Parallel Algorithms for Constrained Tensor Factorization via Alternating Direction Method of Multipliers

Athanasios P. Liavas, Member, IEEE, and Nicholas D. Sidiropoulos, Fellow, IEEE

Abstract

Tensor factorization has proven useful in a wide range of applications, from sensor array processing to communications, speech and audio signal processing, and machine learning. With few recent exceptions, all tensor factorization algorithms were originally developed for centralized, in-memory computation on a single machine; and the few that break away from this mold do not easily incorporate practically important constraints, such as non-negativity. A new constrained tensor factorization framework is proposed in this paper, building upon the Alternating Direction Method of Multipliers (ADMoM). It is shown that this simplifies computations, bypassing the need to solve constrained optimization problems in each iteration; and it naturally leads to distributed algorithms suitable for parallel implementation. This opens the door for many emerging big data-enabled applications. The methodology is exemplified using non-negativity as a baseline constraint, but the proposed framework can incorporate many other types of constraints. Numerical experiments are encouraging, indicating that ADMoM-based non-negative tensor factorization (NTF) has high potential as an alternative to state-of-the-art approaches.

I. INTRODUCTION

Tensor factorization\(^1\) has proven useful in a wide range of signal processing applications, such as direction of arrival estimation [2], communication signal intelligence [3], and speech and audio signal separation [4], [5], as well as cross-disciplinary areas, such as community detection in social networks

\(^1\)In the literature, the terms factorization and decomposition are often used interchangeably, even though the latter alludes to exact decomposition, whereas the former may include a residual term.
[6], and chemical signal analysis [7]. More recently, there has been significant activity in applying tensor factorization theory and methods to problems in machine learning research - see [8].

There are two basic tensor factorization models: parallel factor analysis (PARAFAC) [9], [10] also known as canonical decomposition (CANDECOMP) [11], or CP (and CPD) for CANDECOMP-PARAFAC (Decomposition), or canonical polyadic decomposition (CPD, again); and the Tucker3 model [12]. Both are sum-of-outer-products models which historically served as cornerstones for further developments, e.g., block term decomposition [13], and upon which the vast majority of tensor applications have been built. In this paper, we will primarily focus on the CP model.

Whereas for low-enough$^2$ rank CP is already unique ‘on its own,’ any side information can (and should) be used to enhance identifiability and estimation performance in practice. Towards this end, we may exploit known properties of the sought latent factors, such as non-negativity, sparsity, monotonicity, or unimodality [14]. Whereas many of these properties can be handled with existing tensor factorization software, they generally complicate and slow down model fitting.

Unconstrained tensor factorization is already a hard non-convex (multi-linear) problem; even rank-one least-squares tensor approximation is NP-hard [15]. Many tensor factorization algorithms rely on alternating optimization, usually alternating least-squares (ALS), and imposing e.g., non-negativity and/or sparsity entails replacing linear least-squares conditional updates of the factor matrices with non-negative and/or sparse least-squares updates. In addition to ALS, many derivative-based methods have been developed that update all model parameters at once, see [16] and references therein, and [17], [18] for recent work in this direction.

With few recent exceptions, all tensor factorization algorithms were originally developed for centralized, in-memory computation on a single machine. This model of computation is inadequate for emerging big data-enabled applications, where the tensors to be analyzed cannot be loaded on a single machine, the data is more likely to reside in cloud storage, and cloud computing, or some other kind of high performance parallel architecture, must be used for the actual computation.

A carefully optimized Hadoop/MapReduce [19], [20] implementation of the basic ALS CP-decomposition algorithm was developed in [21], which reported 100-fold scaling improvements relative to the prior art. The jist of [21] is to avoid the explicit computation of ‘blown-up’ intermediate matrix products in the ALS algorithm, particularly for sparse tensors, and parallelization is achieved by splitting the computation of outer products. On the other hand, [21] is not designed for high performance computing (e.g., mesh) architectures, and it does not incorporate constraints on the factor matrices.

A random sampling approach was later proposed in [6], motivated by recent progress in randomized

$^2$E.g., relative to the sum of Kruskal-ranks of the latent factor matrices. Looser bounds can be guaranteed almost-surely.
algorithms for matrix algebra. The idea of [6] is to create and analyze multiple randomly sub-sampled parts of the tensor, then combine the results using a common piece of data to anchor the constituent decompositions. The downside of [6] is that it only works for sparse tensors, and it offers no identifiability guarantees - although it usually works well for sparse tensors.

A different approach based on generalized random sampling was recently proposed in [22], [23]. The idea is to create multiple randomly compressed mixtures (instead of sub-sampled parts) of the original tensor, analyze them all in parallel, and then combine the results. The main advantages of [22], [23] over [6] are that i) identifiability can be guaranteed, ii) no sparsity is needed, and iii) there are theoretical scalability guarantees.

Distributed CP decomposition based on the ALS algorithm has been considered in [24], and more recently in [25], which exploit the inherent parallelism in the matrix version of the linear least-squares subproblems to split the computation in different ways, assuming an essentially ‘flat’ architecture for the computing nodes. Regular (e.g., mesh) architectures and constraints on the latent factors are not considered in [24], [25].

In this paper, we develop algorithms for constrained tensor factorization based on Alternating Direction Method of Multipliers (ADMoM). ADMoM has recently attracted renewed interest [26], primarily for solving certain types of convex optimization problems in a distributed fashion. However, it can also be used to tackle non-convex problems, such as non-negative matrix factorization [26], albeit its convergence properties are far less understood in this case. We focus on non-negative CP decompositions as a working problem, due to the importance of the CP model and non-negativity constraints; but our approach can be generalized to many other types of constraints on the latent factors, as well as other tensor factorizations, such as Tucker3, and tensor completion.

The advantages of our approach are as follows. First, during each ADMoM iteration, we avoid the solution of constrained optimization problems, resulting in considerably smaller computational complexity per iteration compared to constrained least-squares based algorithms, such as alternating non-negative least-squares (NALS). Second, our approach leads naturally to distributed algorithms suitable for parallel implementation on regular high-performance computing (e.g., mesh) architectures. Finally, our approach can easily incorporate many other types of constraints on the latent factors, such as sparsity.

Numerical experiments are encouraging, indicating that ADMoM-based NTF has significant potential as an alternative to state-of-the-art approaches.

The rest of the manuscript is structured as follows. In Section II, we present the NTF problem and in Section III we present the general ADMoM framework. In Section IV, we develop ADMoM for NTF, while in Section V we develop distributed ADMoM for large NTF. In Section VI, we test the behavior
of the developed schemes with numerical experiments. Finally, in Section VII, we conclude the paper.

A. Notation

Vectors, matrices, and tensors are denoted by small, capital, and underlined capital bold letters, respectively; for example, \( x, X, \) and \( \underline{X} \). \( \mathbb{R}^{I \times J \times K}_+ \) denotes the set of \((I \times J \times K)\) real non-negative tensors, while \( \mathbb{R}^{I \times J}_+ \) denotes the set of \((I \times J)\) real non-negative matrices. \( || \cdot ||_F \) denotes the Frobenius norm of the tensor or matrix argument, \( A^\dagger \) denotes the Moore-Penrose pseudoinverse of matrix \( A \), and \((A)_+\) denotes the projection of matrix \( A \) onto the set of element-wise non-negative matrices. The outer product of three vectors \( a \in \mathbb{R}^{I_1}, b \in \mathbb{R}^{J_1}, \) and \( c \in \mathbb{R}^{K_1} \) is the rank-one tensor \( a \circ b \circ c \in \mathbb{R}^{I_1 \times J_1 \times K_1} \) with elements \((a \circ b \circ c)(i, j, k) = a(i)b(j)c(k)\). For matrices \( A \) and \( B \), with compatible dimensions, \( A \odot B \) denotes the Khatri-Rao (columnwise Kronecker) product, \( A \oslash B \) denotes the Hadamard (element-wise) product, and \( A \ast B \) denotes the matrix inner product, that is \( A \ast B := \text{trace}(A^TB) = \sum_{i,j} A_{i,j}B_{i,j} \).

II. NON-NEGATIVE TENSOR FACTORIZATION

Let tensor \( X^o \in \mathbb{R}^{I \times J \times K}_+ \) admit a non-negative\(^3\) CP decomposition of order \( F \)

\[
X^o = [A^o, B^o, C^o] = \sum_{f=1}^{F} a^o_f \circ c^o_f \circ c^o_f,
\]

where \( A^o = [a^o_1 \cdots a^o_F] \in \mathbb{R}_+^{I \times F}, B^o = [b^o_1 \cdots b^o_F] \in \mathbb{R}_+^{J \times F}, \) and \( C^o = [c^o_1 \cdots c^o_F] \in \mathbb{R}_+^{K \times F} \). We observe a noisy version of \( X^o \) expressed as

\[
\underline{X} = X^o + E.
\]

In order to estimate \( A^o, B^o, \) and \( C^o \), we compute matrices \( A \in \mathbb{R}_+^{I \times F}, B \in \mathbb{R}_+^{J \times F}, \) and \( C \in \mathbb{R}_+^{K \times F} \) that solve the optimization problem

\[
\begin{align*}
& \min_{A, B, C} f_X(A, B, C) \\
& \text{subject to } A \geq 0, \ B \geq 0, \ C \geq 0,
\end{align*}
\]

where \( f \) is a function measuring the quality of the factorization, \( 0 \) is the zero matrix of appropriate dimensions, and the inequalities are element-wise. A common choice for \( f_X \), motivated via maximum likelihood estimation for \( E \) with Gaussian independent and identically distributed (i.i.d.) elements, is

\[
f_X(A, B, C) = \frac{1}{2} ||\underline{X} - [A, B, C]||_F^2.
\]

\(^3\)Note that, due to the non-negativity constraints on the latent factors, \( F \) can be higher than the rank of \( X^o \).
Let $\mathbf{W} = [\mathbf{A}, \mathbf{B}, \mathbf{C}]$ and $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \text{and } \mathbf{W}^{(3)}$ be the matrix unfoldings of $\mathbf{W}$, with respect to the first, second, and third dimension, respectively. Then,

$$
\mathbf{W}^{(1)} = \mathbf{A} (\mathbf{C} \odot \mathbf{B})^T,
$$

$$
\mathbf{W}^{(2)} = \mathbf{B} (\mathbf{C} \odot \mathbf{A})^T,
$$

$$
\mathbf{W}^{(3)} = \mathbf{C} (\mathbf{B} \odot \mathbf{A})^T,
$$

and $f_X$ can be equivalently expressed as

$$
f_X(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \| \mathbf{X}^{(1)} - \mathbf{A} (\mathbf{C} \odot \mathbf{B})^T \|_F^2
$$

$$
= \frac{1}{2} \| \mathbf{X}^{(2)} - \mathbf{B} (\mathbf{C} \odot \mathbf{A})^T \|_F^2
$$

$$
= \frac{1}{2} \| \mathbf{X}^{(3)} - \mathbf{C} (\mathbf{B} \odot \mathbf{A})^T \|_F^2.
$$

These expressions are the basis for ALS-type CP optimization, because they enable simple linear least-squares updating of one matrix given the other two. Using NALS for each update step is a popular approach for the solution of (1), but non-negativity brings a significant computational burden relative to plain ALS and also complicates the development of parallel algorithms for NTF. It is worth noting that the above expressions will also prove useful during the development of the ADMoM-based NTF algorithm.

### III. ADMoM

ADMoM is a technique for the solution of optimization problems of the form [26]

$$
\min_{\mathbf{x}, \mathbf{z}} \quad f(\mathbf{x}) + g(\mathbf{z})
$$

subject to $\mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{z} = \mathbf{c},$

where $\mathbf{x} \in \mathbb{R}^{n_1}, \mathbf{z} \in \mathbb{R}^{n_2}, \mathbf{A} \in \mathbb{R}^{m \times n_1}, \mathbf{B} \in \mathbb{R}^{m \times n_2}, \mathbf{c} \in \mathbb{R}^m, f : \mathbb{R}^{n_1} \to \mathbb{R}, \text{ and } g : \mathbb{R}^{n_2} \to \mathbb{R}.$

The augmented Lagrangian for problem (5) is

$$
L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^T (\mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{z} - \mathbf{c}) + \frac{\rho}{2} \| \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{z} - \mathbf{c} \|_2^2,
$$

where $\rho > 0$ is a penalty parameter. Assuming that at time instant $k$ we have computed $\mathbf{z}^k$ and $\mathbf{y}^k$, which comprise the state of the algorithm, the $(k + 1)$-st iteration of ADMoM is\footnote{Note that $\mathbf{y}^{kT}$ is shorthand notation for $(\mathbf{y}^k)^T$.}

$$
\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}} \left( f(\mathbf{x}) + \mathbf{y}^{kT} \mathbf{A} \mathbf{x} + \frac{\rho}{2} \| \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{z}^k - \mathbf{c} \|_2^2 \right)
$$

$$
\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \left( g(\mathbf{z}) + \mathbf{y}^{kT} \mathbf{B} \mathbf{z} + \frac{\rho}{2} \| \mathbf{A} \mathbf{x}^{k+1} + \mathbf{B} \mathbf{z} - \mathbf{c} \|_2^2 \right)
$$

$$
\mathbf{y}^{k+1} = \mathbf{y}^k + \rho \left( \mathbf{A} \mathbf{x}^{k+1} + \mathbf{B} \mathbf{z}^{k+1} - \mathbf{c} \right).
$$
It can be shown that, under certain conditions (among them convexity of \( f \) and \( g \)), ADMoM converges in a certain sense (see [26] for an excellent review of ADMoM, including some convergence analysis results). However, ADMoM can be used even when problem (5) is non-convex. In this case, we use ADMoM with the goal of reaching a good local minimum [26]. Note that this is all we can realistically hope for anyway, irrespective of approach or algorithm used, since tensor factorization is NP-hard [15].

A. ADMoM for set-constrained optimization

Let us consider the set-constrained optimization problem

\[
\min_{x} \quad f(x)
\]

subject to \( x \in \mathcal{X} \),

where \( x \in \mathbb{R}^n \) and \( \mathcal{X} \subseteq \mathbb{R}^n \) is a closed convex set. At first sight, this problem does not seem suitable for ADMoM. However, if we introduce variable \( z \in \mathbb{R}^n \), we can write the equivalent problem

\[
\min_{x,z} \quad f(x) + g(z)
\]

subject to \( x - z = 0 \),

where \( g \) is the indicator function of set \( \mathcal{X} \), that is,

\[
g(z) := \begin{cases} 
0, & z \in \mathcal{X}, \\
\infty, & z \notin \mathcal{X}.
\end{cases}
\]

Then it becomes clear that (7) can be solved via ADMoM. Assuming that at time instant \( k \) we have computed \( z^k \) and \( y^k \), the \((k+1)\)-st iteration of ADMoM is [26]

\[
\begin{align*}
\mathbf{x}^{k+1} &= \arg\min_{x} \left( f(x) + y^{kT}x + \frac{\rho}{2}\|x - z^k\|^2_2 \right) \\
\mathbf{z}^{k+1} &= \arg\min_{z} \left( g(z) - y^{kT}z + \frac{\rho}{2}\|x^{k+1} - z\|^2_2 \right) \\
&= \Pi_{\mathcal{X}} \left( \mathbf{x}^{k+1} + \frac{1}{\rho}y^k \right) \\
\mathbf{y}^{k+1} &= y^k + \rho (\mathbf{x}^{k+1} - \mathbf{z}^{k+1}),
\end{align*}
\]

where \( \Pi_{\mathcal{X}} \) denotes projection (in the Euclidean norm) onto \( \mathcal{X} \).

IV. ADMoM for NTF

In this section, we adopt the approach of subsection III-A and develop an ADMoM-based NTF algorithm. At first, we must put the NTF problem (1) into ADMoM form. Towards this end, we introduce auxiliary variables \( \mathbf{\bar{A}} \in \mathbb{R}^{I \times F} \), \( \mathbf{\bar{B}} \in \mathbb{R}^{J \times F} \), and \( \mathbf{\bar{C}} \in \mathbb{R}^{K \times F} \) and consider the equivalent optimization problem

\[
\begin{align*}
\min_{\mathbf{A},\mathbf{\bar{A}},\mathbf{B},\mathbf{\bar{B}},\mathbf{C}} \quad & f_{X}(\mathbf{A},\mathbf{B},\mathbf{C}) + g(\mathbf{\bar{A}}) + g(\mathbf{\bar{B}}) + g(\mathbf{\bar{C}}) \\
\text{subject to} \quad & \mathbf{A} - \mathbf{\bar{A}} = 0, \; \mathbf{B} - \mathbf{\bar{B}} = 0, \; \mathbf{C} - \mathbf{\bar{C}} = 0,
\end{align*}
\]
\[ L_{\rho}(A, B, C, \tilde{A}, \tilde{B}, \tilde{C}, Y_A, Y_B, Y_C) = f_X(A, B, C) + g(\tilde{A}) + g(\tilde{B}) + g(\tilde{C}) \]
\[ + Y_A^{*}(A - \tilde{A}) + \frac{\rho_A}{2} \|A - \tilde{A}\|_F^2 \]
\[ + Y_B^{*}(B - \tilde{B}) + \frac{\rho_B}{2} \|B - \tilde{B}\|_F^2 \]
\[ + Y_C^{*}(C - \tilde{C}) + \frac{\rho_C}{2} \|C - \tilde{C}\|_F^2. \]

(10)

where, for any matrix argument \( M \),
\[ g(M) := \begin{cases} 
0, & \text{if } M \geq 0, \\
\infty, & \text{otherwise}
\end{cases} \]  

(9)

We introduce the dual variables \( Y_A \in \mathbb{R}^{I \times F} \), \( Y_B \in \mathbb{R}^{J \times F} \), and \( Y_C \in \mathbb{R}^{K \times F} \), and the vector of penalty terms \( \rho := [\rho_A \rho_B \rho_C]^T \). The augmented Lagrangian is given in (10), at the top of this page.

The ADMoM for problem (8) is as follows:

\[ (A^{k+1}, B^{k+1}, C^{k+1}) = \arg\min_{A, B, C} (f_X(A, B, C) \]
\[ + Y_A^{k} \cdot A + \frac{\rho_A}{2} \|A - \tilde{A}^{k}\|_F^2 \]
\[ + Y_B^{k} \cdot B + \frac{\rho_B}{2} \|B - \tilde{B}^{k}\|_F^2 \]
\[ + Y_C^{k} \cdot C + \frac{\rho_C}{2} \|C - \tilde{C}^{k}\|_F^2 \) \]

(11)

\[ \tilde{A}^{k+1} = \left( A^{k+1} + \frac{1}{\rho_A} Y_A^{k} \right) \]
\[ \tilde{B}^{k+1} = \left( B^{k+1} + \frac{1}{\rho_B} Y_B^{k} \right) \]
\[ \tilde{C}^{k+1} = \left( C^{k+1} + \frac{1}{\rho_C} Y_C^{k} \right) \]
\[ Y_A^{k+1} = Y_A^{k} + \rho_A (A^{k+1} - \tilde{A}^{k+1}) \]
\[ Y_B^{k+1} = Y_B^{k} + \rho_B (B^{k+1} - \tilde{B}^{k+1}) \]
\[ Y_C^{k+1} = Y_C^{k} + \rho_C (C^{k+1} - \tilde{C}^{k+1}) \].

The minimization problem in the first line of (11) is non-convex. Using the equivalent expressions for \( f_X \) in (4), we propose the alternating optimization scheme of (12), at the top of the next page. The updates of (12) can be executed either for a predetermined number of iterations, or until convergence.\(^5\) We observe that, during each ADMoM iteration, we avoid the solution of constrained optimization problems. This seems favorable, especially in the cases where the size of the problem is (very) large.

\(^5\) In our implementations, we execute these updates once per ADMoM iteration.
\[ A^{k+1} = \arg\min_{A} \left( \frac{1}{2} \| X^{(1)} - A(C^{k} \odot B^{k})^T \|_F^2 + Y_{A}^k \ast A + \frac{\rho_{A}}{2} \| A - \tilde{A}^{k} \|_F^2 \right) \]
\[ = \left( X^{(1)}(C^{k} \odot B^{k}) + \rho_{A} \tilde{A}^{k} - Y_{A}^{k} \right) \left( (C^{k} \odot B^{k})^T(C^{k} \odot B^{k}) + \rho_{A} I_F \right)^{-1}, \]
\[ B^{k+1} = \arg\min_{B} \left( \frac{1}{2} \| X^{(2)} - B(C^{k} \odot A^{k+1})^T \|_F^2 + Y_{B}^k \ast B + \frac{\rho_{B}}{2} \| B - \tilde{B}^{k} \|_F^2 \right) \]
\[ = \left( X^{(2)}(C^{k} \odot A^{k+1}) + \rho_{B} \tilde{B}^{k} - Y_{B}^{k} \right) \left( (C^{k} \odot A^{k+1})^T(C^{k} \odot A^{k+1}) + \rho_{B} I_F \right)^{-1}, \]
\[ C^{k+1} = \arg\min_{C} \left( \frac{1}{2} \| X^{(3)} - C(B^{k+1} \odot A^{k+1})^T \|_F^2 + Y_{C}^k \ast C + \frac{\rho_{C}}{2} \| C - \tilde{C}^{k} \|_F^2 \right) \]
\[ = \left( X^{(3)}(B^{k+1} \odot A^{k+1}) + \rho_{C} \tilde{C}^{k} - Y_{C}^{k} \right) \left( (B^{k+1} \odot A^{k+1})^T(B^{k+1} \odot A^{k+1}) + \rho_{C} I_F \right)^{-1}. \]  

We note that \( A^{k}, B^{k}, \) and \( C^{k} \) are not necessarily non-negative. They become non-negative (or, at least, their negative elements become very small) upon convergence. On the other hand, \( \tilde{A}^{k}, \tilde{B}^{k}, \) and \( \tilde{C}^{k} \) are by construction non-negative.

A. Computational complexity per iteration

Each ADMoM iteration consists of simple matrix operations. Thus, rough estimates of its computational complexity can be easily derived (of course, accurate estimates can be derived after fixing the algorithms that implement the matrix operations).

A rough estimate for the computational complexity of the update of \( A^{k} \) (see the first update in (12)) can be derived as follows:

1) \( O(nF + IF) \) for the computation of the term \( X^{(1)}(C^{k} \odot B^{k}) + \rho_{A} \tilde{A}^{k} - Y_{A}^{k} \), where \( n \) is the number of nonzero elements of tensor \( X \) (also of matrix \( X^{(1)} \)). Note that \( n = IJK \) for dense tensors, but for sparse tensors \( n \ll IJK \). This is because the product \( X^{(1)}(C^{k} \odot B^{k}) \) can be computed with \( 3nF \) flops, by exploiting sparsity and the structure of the Khatri-Rao product \([27],[28],[29]\). With \( 5nF \) flops, it is possible to parallelize this computation \([21]\). Efficient (in terms of favorable memory access pattern) in-place computation of all three products needed for the update of \( A^{k}, B^{k}, C^{k} \) from a single copy of \( X \) has been recently considered in \([30]\), which also features potential flop gains as a side-benefit.

2) \( O((K + J)F^2) \) for the computation of the term \( (C^{k} \odot B^{k})^T(C^{k} \odot B^{k}) + \rho_{A} I_F \), and \( O(F^3) \) for its Cholesky decomposition. This is because \( (C^{k} \odot B^{k})^T(C^{k} \odot B^{k}) = \left( (C^{k})^T C^{k} \right) \otimes \left( (B^{k})^T B^{k} \right) \).

3) \( O(F^2I) \) for the computation of the system solution that gives the updated value \( A^{k+1} \).

Analogous estimates can be derived for the updates of \( B^{k} \) and \( C^{k} \). Finally, the updates of the auxiliary and dual variables require, in total, \( O((I + J + K)F) \) arithmetic operations.
B. Convergence

Let \( Z := (A, B, C, \tilde{A}, \tilde{B}, \tilde{C}, Y_A, Y_B, Y_C) \). It can be proven that \( Z \) is a Karush-Kuhn-Tucker (KKT) point for the NTF problem (8) if

\[
\begin{align*}
(X^{(1)} - A(C \odot B)^T)(C \odot B) - Y_A &= 0 \\
(X^{(2)} - B(C \odot A)^T)(C \odot A) - Y_B &= 0 \\
(X^{(3)} - C(B \odot A)^T)(B \odot A) - Y_C &= 0 \\
A - \tilde{A} &= 0, \quad B - \tilde{B} = 0, \quad C - \tilde{C} = 0 \\
Y_A &\leq 0, \quad Y_B \leq 0, \quad Y_C \leq 0 \\
Y_A \odot \tilde{A} &= 0, \quad Y_B \odot \tilde{B} = 0, \quad Y_C \odot \tilde{C} = 0.
\end{align*}
\]  

(13)

Proposition 1: Let \( \{Z^k\} \) be a sequence generated by ADMoM for NTF that satisfies condition

\[
\lim_{k \to \infty} (Z^{k+1} - Z^k) = 0.
\]

(14)

Then, any accumulation point of \( \{Z^k\} \) is a KKT point of problem (8). Consequently, any accumulation point of \( \{A^k, B^k, C^k\} \) is a KKT point of problem (1).

Proof: The proof follows closely the steps of the proof of Proposition 2.1 of [31] and is omitted.\(^6\)

Proposition 1 implies that, whenever \( \{Z^k\} \) converges, it converges to a KKT point. We will further discuss ADMoM convergence from a practical point of view in the section with the numerical experiments.

C. Stopping criteria

The primal residual for variable \( A^k \) is defined as

\[
P_A^k := A^k - \tilde{A}^k,
\]

(15)

while quantity

\[
D_{\tilde{A}}^k := \rho_A(\tilde{A}^k - \tilde{A}^{k-1})
\]

(16)

can be viewed as a dual feasibility residual (see [26, Section 3.3]). We analogously define \( P_B^k, D_B^k, P_C^k, \) and \( D_C^k \).

We stop the algorithm if all primal and dual residuals are sufficiently small. More specifically, we introduce small positive constants \( \epsilon^{\text{abs}} \) and \( \epsilon^{\text{rel}} \) and consider \( P_A^k \) and \( D_{\tilde{A}}^k \) small if

\[
\|P_A^k\|_F \leq \sqrt{IF} \epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\left\{\|A^k\|_F, \|\tilde{A}^k\|_F\right\},
\]

(17)

\[
\|D_{\tilde{A}}^k\|_F \leq \sqrt{IF} \epsilon^{\text{abs}} + \epsilon^{\text{rel}} \|Y_A^k\|_F.
\]

(18)

\(^6\)See report [32] for a detailed proof.
Analogous conditions apply for the other residuals. Reasonable values for $\epsilon_{rel}$ are $\epsilon_{rel} \lesssim 10^{-3}$, while the value of $\epsilon_{abs}$ depends on the scale of the values of the latent factors.

We note that stopping criteria (17) and (18) involve quantities of the size of the latent factors which, in most cases, is small compared to the size of the tensor. Thus, their computation, even during every ADMoM iteration, is not computationally demanding.

D. Varying penalty parameters

We have found very useful in practice to vary the values of each one of the penalty parameters, $\rho_A$, $\rho_B$, and $\rho_C$, depending on the size of the corresponding primal and dual residuals (see [26, Section 3.4]). More specifically, the penalty parameters $\rho^{k+1}_M$ for $M = A, B, C$, are updated as follows:

$$
\rho^{k+1}_M = \begin{cases} 
\tau^{\text{incr}} \rho^k_M, & \text{if } \|P^k_M\|_F > \mu \|D^k_M\|_F, \\
\rho^k_M/\tau^{\text{decr}}, & \text{if } \|D^k_M\|_F > \mu \|P^k_M\|_F, \\
\rho^k_M, & \text{otherwise},
\end{cases}
$$

where $\mu > 1$, $\tau^{\text{incr}} > 1$, and $\tau^{\text{decr}} > 1$ are the adaptation parameters. Large values of $\rho_M$ place large penalty on violations of primal feasibility, leading to small primal residuals, while small values of $\rho_M$ tend to reduce the dual residuals.

E. ADMoM for tensor factorization with structural constraints

ADMoM can easily handle certain structural constraints on the latent factors [26], [33]. For example, if we want to solve an NTF problem with the added constraint that the number of nonzero elements of $A$ is lower than or equal to a given number $c_A$, then we can adopt an approach similar to that followed in Section IV with the only difference being that, instead of using $g(\tilde{A})$ defined in (9), we use $g_{c_A}(\tilde{A})$ where, for any matrix argument $M$,

$$
g_{c}(M) := \begin{cases} 
0, & \text{if } M \geq 0 \text{ and } \|M\|_0 \leq c, \\
\infty, & \text{otherwise},
\end{cases}
$$

The only difference between the ADMoM for this case and the one presented in (11) and (12) is in the update of $\tilde{A}^k$. More specifically, instead of using projection onto the set of non-negative matrices, we must use projection onto the set of non-negative matrices with at most $c_A$ nonzero elements, which can be easily computed through sorting of the elements of $\tilde{A}^k$. Using analogous arguments, we can incorporate into our ADMoM framework box or other set constraints on the latent factors. The development of the corresponding ADMoM is almost trivial if projection onto the constraint set is easy.

Thorough study of ADMoM-based algorithms for tensor factorization and/or completion with more complicated structural constraints is a topic of future research.
V. DISTRIBUTED ADMoM FOR LARGE NTF

In this section, we assume that all dimensions of tensor $X$ are large and derive an ADMoM-based NTF that is suitable for parallel implementation. Of course, our framework can handle the cases where only one or two of the dimensions of $X$ are large.

A. Matrix unfoldings in terms of partitioned matrix factors

Let $W = [A, B, C]$, and $A$, $B$, and $C$ be partitioned as

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_{N_A} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ \vdots \\ B_{N_B} \end{bmatrix}, \quad C = \begin{bmatrix} C_1 \\ \vdots \\ C_{N_C} \end{bmatrix},$$

with $A_{n_A} \in \mathbb{R}^{I_{n_A} \times F}$, for $n_A = 1, \ldots, N_A$, $\sum_{n_A=1}^{N_A} I_{n_A} = I$, $B_{n_B} \in \mathbb{R}^{J_{n_B} \times F}$, for $n_B = 1, \ldots, N_B$, $\sum_{n_B=1}^{N_B} J_{n_B} = J$, and $C_{n_C} \in \mathbb{R}^{K_{n_C} \times F}$, for $n_C = 1, \ldots, N_C$, $\sum_{n_C=1}^{N_C} K_{n_C} = K$.

We first derive partitionings of the matrix unfoldings of $W$ in terms of (the blocks of) matrices $A$, $B$, and $C$. Towards this end, we write $W^{(1)}$ as

$$W^{(1)} = A(C \odot B)^T$$

$$= \begin{bmatrix} A_1 \\ \vdots \\ A_{N_A} \end{bmatrix} \left( \begin{bmatrix} C_1 \\ \vdots \\ C_{N_C} \end{bmatrix} \odot B \right)^T$$

$$= \begin{bmatrix} A_1 \\ \vdots \\ A_{N_A} \end{bmatrix} \left( \begin{bmatrix} C_1 \odot B \\ \vdots \\ C_{N_C} \odot B \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} A_1 \\ \vdots \\ A_{N_A} \end{bmatrix} \left( \begin{bmatrix} (C_1 \odot B)^T \\ \vdots \\ (C_{N_C} \odot B)^T \end{bmatrix} \right).$$

Thus, $W^{(1)}$ can be partitioned as

$$W^{(1)} = \begin{bmatrix} W^{(1)}_{1,1} & \cdots & W^{(1)}_{1,N_C} \\ \vdots & \ddots & \vdots \\ W^{(1)}_{N_A,1} & \cdots & W^{(1)}_{N_A,N_C} \end{bmatrix},$$

where the $(n_A, n_C)$-th block of $W^{(1)}$ is equal to the $I_{n_A} \times (J K_{n_C})$ matrix $W^{(1)}_{n_A,n_C} = A_{n_A} (C_{n_C} \odot B)^T$, for $n_A = 1, \ldots, N_A$ and $n_C = 1, \ldots, N_C$. 

Similarly, it can be shown that $W^{(2)}$ can be partitioned into blocks $W_{n_Bn_C}^{(2)} = B_{n_B}(C_{n_C} \odot A)^T$, of dimensions $J_{n_B} \times (IK_{n_C})$, for $n_B = 1, \ldots, N_B$ and $n_C = 1, \ldots, N_C$, and $W^{(3)}$ can be partitioned into blocks $W_{n_Cn_B}^{(3)} = C_{n_C}(B_{n_B} \odot A)^T$, of dimensions $K_{n_C} \times (IJK_{n_B})$, for $n_C = 1, \ldots, N_C$ and $n_B = 1, \ldots, N_B$.

If we partition $X^{(1)}$, $X^{(2)}$, and $X^{(3)}$ accordingly, then we can write
\begin{align*}
    f_X(A, B, C) &= \sum_{n_A=1}^{N_A} \sum_{n_C=1}^{N_C} \frac{1}{2} \|X_{n_An_C}^{(1)} - A_{n_A}(C_{n_C} \odot B)^T\|_F^2 \\
    &= \sum_{n_B=1}^{N_B} \sum_{n_C=1}^{N_C} \frac{1}{2} \|X_{n_Bn_C}^{(2)} - B_{n_B}(C_{n_C} \odot A)^T\|_F^2 \tag{22}
    \\
    &= \sum_{n_C=1}^{N_C} \sum_{n_B=1}^{N_B} \frac{1}{2} \|X_{n_Cn_B}^{(3)} - C_{n_C}(B_{n_B} \odot A)^T\|_F^2.
\end{align*}

These expressions will be fundamental for the development of the distributed ADMoM for large NTF.

B. Distributed ADMoM for large NTF

In order to put the large NTF problem into ADMoM form, we introduce auxiliary variables $\bar{A} = [\bar{A}^T_1 \cdots \bar{A}^T_{N_A}]^T$, with $\bar{A}_{n_A} \in \mathbb{R}^{I_{n_A} \times F}$, for $n_A = 1, \ldots, N_A$, $\bar{B} = [\bar{B}^T_1 \cdots \bar{B}^T_{N_B}]^T$, with $\bar{B}_{n_B} \in \mathbb{R}^{J_{n_B} \times F}$, for $n_B = 1, \ldots, N_B$, and $\bar{C} = [\bar{C}^T_1 \cdots \bar{C}^T_{N_C}]^T$, with $\bar{C}_{n_C} \in \mathbb{R}^{K_{n_C} \times F}$, for $n_C = 1, \ldots, N_C$, and consider the equivalent problem
\begin{align*}
    \min_{A, \bar{A}, B, \bar{B}, C} & \quad f_X(A, B, C) + \sum_{n_A=1}^{N_A} g(\bar{A}_{n_A}) + \sum_{n_B=1}^{N_B} g(\bar{B}_{n_B}) + \sum_{n_C=1}^{N_C} g(\bar{C}_{n_C}) \\
    \text{subject to} & \quad A_{n_A} - \bar{A}_{n_A} = 0, \quad n_A = 1, \ldots, N_A, \\
    & \quad B_{n_B} - \bar{B}_{n_B} = 0, \quad n_B = 1, \ldots, N_B, \\
    & \quad C_{n_C} - \bar{C}_{n_C} = 0, \quad n_C = 1, \ldots, N_C. \tag{23}
\end{align*}

If we introduce dual variables $Y_A = [Y^T_{A_1} \cdots Y^T_{A_{N_A}}]^T$, with $Y_{A_{n_A}} \in \mathbb{R}^{I_{n_A} \times F}$, for $n_A = 1, \ldots, N_A$, $Y_B = [Y^T_{B_1} \cdots Y^T_{B_{N_B}}]^T$, with $Y_{B_{n_B}} \in \mathbb{R}^{J_{n_B} \times F}$, for $n_B = 1, \ldots, N_B$, and $Y_C = [Y^T_{C_1} \cdots Y^T_{C_{N_C}}]^T$, with $Y_{C_{n_C}} \in \mathbb{R}^{K_{n_C} \times F}$, for $n_C = 1, \ldots, N_C$, the augmented Lagrangian is written as in (24), at the top of the next page.

\textsuperscript{7}An extension of the above partitioning scheme to higher order tensors appears in Appendix A.
The ADMoM for this problem is as follows:

\[
(A^{k+1}, B^{k+1}, C^{k+1}) = \arg\min_{A,B,C} \left( f_X(A, B, C) + \sum_{n_A=1}^{N_A} g(\tilde{A}_{n_A}) + \sum_{n_B=1}^{N_B} g(\tilde{B}_{n_B}) + \sum_{n_C=1}^{N_C} g(\tilde{C}_{n_C}) \right)
\]

\[
+ \sum_{n_A=1}^{N_A} \left( Y_{A_{n_A}}^k \ast (A_{n_A} - \tilde{A}_{n_A}^k) + \frac{\rho_A}{2} \| A_{n_A} - \tilde{A}_{n_A}^k \|_F^2 \right)
\]

\[
+ \sum_{n_B=1}^{N_B} \left( Y_{B_{n_B}}^k \ast (B_{n_B} - \tilde{B}_{n_B}^k) + \frac{\rho_B}{2} \| B_{n_B} - \tilde{B}_{n_B}^k \|_F^2 \right)
\]

\[
+ \sum_{n_C=1}^{N_C} \left( Y_{C_{n_C}}^k \ast (C_{n_C} - \tilde{C}_{n_C}^k) + \frac{\rho_C}{2} \| C_{n_C} - \tilde{C}_{n_C}^k \|_F^2 \right)
\]

(25)

The minimization problem in the first line of (25) is non-convex. Based on (22), we propose the alternating optimization scheme given in (26) at the next page.

Again, during each ADMoM iteration, we avoid the solution of constrained optimization problems. Furthermore, and more importantly, having computed all algorithm quantities at iteration \(k\), the updates of \(A_{n_A}^k\), for \(n_A = 1, \ldots, N_A\), are independent and can be computed in parallel. Then, we can compute in parallel the updates of \(B_{n_B}^k\), for \(n_B = 1, \ldots, N_B\), and, finally, the updates of \(C_{n_C}^k\), for \(n_C = 1, \ldots, N_C\).
\[
A_{n_A}^{k+1} = \underset{A_{n_A}}{\text{argmin}} \left( \frac{1}{2} \left\| X_{n_A,n_C}^{(1)} - A_{n_A} (C_{n_C}^k \odot B^k)^T \right\|_F^2 \right) + Y_{A_{n_A}}^k * A_{n_A} + \frac{\rho_A}{2} \left\| A_{n_A} - \tilde{A}_{n_A}^k \right\|_F^2 \\
= \left( \frac{1}{2} \sum_{n_C=1}^{N_C} X_{n_A,n_C}^{(1)} (C_{n_C}^k \odot B^k) + \rho_A \tilde{A}_{n_A}^k - Y_{A_{n_A}}^k \right) \left( \frac{1}{2} \sum_{n_C=1}^{N_C} (C_{n_C}^k \odot B^k)^T (C_{n_C}^k \odot B^k) + \rho_A I_F \right)^{-1}, \text{ for } n_A = 1, \ldots, N_A,
\]

\[
B_{n_B}^{k+1} = \underset{B_{n_B}}{\text{argmin}} \left( \frac{1}{2} \left\| X_{n_B,n_C}^{(2)} - B_{n_B} (C_{n_C}^k \odot A^{k+1})^T \right\|_F^2 \right) + Y_{B_{n_B}}^k * B_{n_B} + \frac{\rho_B}{2} \left\| B_{n_B} - \tilde{B}_{n_B}^k \right\|_F^2 \\
= \left( \frac{1}{2} \sum_{n_C=1}^{N_C} X_{n_B,n_C}^{(2)} (C_{n_C}^k \odot A^{k+1}) + \rho_B \tilde{B}_{n_B}^k - Y_{B_{n_B}}^k \right) \left( \frac{1}{2} \sum_{n_C=1}^{N_C} (C_{n_C}^k \odot A^{k+1})^T (C_{n_C}^k \odot A^{k+1}) + \rho_B I_F \right)^{-1}, \text{ for } n_B = 1, \ldots, N_B,
\]

\[
C_{n_C}^{k+1} = \underset{C_{n_C}}{\text{argmin}} \left( \frac{1}{2} \left\| X_{n_C,n_B}^{(3)} - C_{n_C} (B_{n_B}^k \odot A^{k+1})^T \right\|_F^2 \right) + Y_{C_{n_C}}^k * C_{n_C} + \frac{\rho_C}{2} \left\| C_{n_C} - \tilde{C}_{n_C}^k \right\|_F^2 \\
= \left( \frac{1}{2} \sum_{n_B=1}^{N_B} X_{n_C,n_B}^{(3)} (B_{n_B}^k \odot A^{k+1}) + \rho_C \tilde{C}_{n_C}^k - Y_{C_{n_C}}^k \right) \left( \frac{1}{2} \sum_{n_B=1}^{N_B} (B_{n_B}^k \odot A^{k+1})^T (B_{n_B}^k \odot A^{k+1}) + \rho_C I_F \right)^{-1}, \text{ for } n_C = 1, \ldots, N_C.
\]

(26)

We note that we can solve problem (23) using the centralized ADMoM of Section IV. In fact, if we initialize the corresponding quantities of the two algorithms with the same values, then the two algorithms evolve in exactly the same way. As a result, the study (for example, convergence analysis and/or numerical behavior) of one of them is sufficient for the characterization of both.

Thus, via the distributed ADMoM, we simply uncover the inherent parallelism in the updates of the blocks of \(A^k, B^k,\) and \(C^k\). In Appendix B, we present a detailed proof of the equivalence of these two forms of ADMoM NTF.

C. A parallel implementation of ADMoM for large NTF

In the sequel, we briefly describe a simple implementation of ADMoM for large NTF on a mesh-type architecture. In order to keep the presentation simple, we assume that (1) \(N_A = N_B = N_C = N\) and (2) each of the matrix unfoldings \(X^{(1)}, X^{(2)},\) and \(X^{(3)}\) has been split into \(N^2\) blocks, with their \((i,j)\)-th
blocks stored at the \((i, j)\)-th processing element, for \(i, j = 1, \ldots, N\) (for related results in the matrix factorization context see [34]).

In Figure 1, we depict the data flow for the computation of the blocks of \(A^{k+1}\). The inputs to the \(N\) top processing elements are \(C_n^k\), for \(n = 1, \ldots, N\), as well as \(B^k\), which is common input to all top processing elements. Each processing element uses its inputs and memory contents and computes certain partial matrix sums. The communications between the processing elements are local and involve either the forwarding of the terms \(C_n^k\), for \(n = 1, \ldots, N\), and \(B^k\) (top-down communication), or the forwarding of the partial sums \(\sum_{i=1}^{j} X_{n,i}^{(1)}(C_l^k \odot B^k)\) and \(\sum_{i=1}^{j}(C_l^k \odot B^k)^T(C_l^k \odot B^k)\) (left-right communication), of dimensions \(\frac{N}{N \times F} \) and \(F \times F\), respectively. The computation of \(A_{n+1}^{k+1}\), for \(n = 1, \ldots, N\), amounts to solution of \(\rho\) systems of linear equations with common coefficient matrix and takes place at the rightmost computing elements.

Then, using a similar strategy, we can compute the blocks of \(B^{k+1}\) and, finally, the blocks of \(C^{k+1}\). The updates of the auxiliary and dual variables are very simple and can be performed locally (see at the rightmost computing elements of Figure 1).

As we see in Figure 1, in order to compute the blocks of the \(A^{k+1}\), we use the appropriate blocks of \(C_n^k\), for \(n = 1, \ldots, N\), as well as the *whole* matrix \(B^k\). When the size of \(B^k\) is not very large, the communication cost is *not* prohibitive (analogous arguments holds for the computation of the blocks of \(B^{k+1}\) and \(C^{k+1}\)). Of course, if one or more latent factors are very large, the communication cost significantly increases.
Concerning the distributed implementation of ADMoM for large tensor factorization with structural constraints other than the non-negativity of the latent factors, we note that, if the size of the latent factors is not very large, the communication cost of gathering together the blocks of the auxiliary variables $\tilde{A}^k$, $\tilde{B}^k$, and $\tilde{C}^k$, is not prohibitive, enabling the computation of more complicated non-separable projections, like, for example, projection onto the set of non-negative matrices with a certain maximum number of non-negative elements.

Actual implementation of the distributed ADMoM for large NTF will depend on the specific parallel architecture and programming environment used. Since our aim in this paper is to introduce the basic methodology and computational framework, we leave those customizations and performance tune-ups, which are further away from the signal processing core, for follow-up work to be reported in the high-performance computing literature.

VI. NUMERICAL EXPERIMENTS

A. Comparison of ADMoM with NALS and NLS

In our numerical experiments, we compare ADMoM NTF with (1) NALS NTF, as implemented in the parafac routine of the N-way toolbox for Matlab [35] and (2) NTF using the nonlinear least-squares solvers (NLS), as implemented in the sdf_nls routine of tensorlab [36] (with the non-negativity option turned on in both cases). In all cases, we use random initialization. More specifically, the initialization of the ADMoM NTF is as follows. We give non-negative random values to $B^0$ and $C^0$ and zero values to the other state variables of the algorithm, namely, $\tilde{A}^0$, $\tilde{B}^0$, $\tilde{C}^0$, $Y_{A^0}$, $Y_{B^0}$, and $Y_{C^0}$.

In extensive numerical experiments, we have observed that the relative performance of the algorithms depends on the size and rank of the tensor as well as the additive noise power. Thus, we consider 12 different scenarios, corresponding to the combinations of the following cases:

1) one, two, or three tensor dimensions are large;
2) rank $F$ is small or large;
3) additive noise is weak or strong.

For each scenario, we generate $R = 50$ realizations of tensor $X$ as follows. We generate random matrices $A^0$, $B^0$, and $C^0$ with i.i.d. $U[0,1]$ elements (using the rand command of Matlab) and construct $X = [A^0, B^0, C^0] + N$, where $N$ consists of i.i.d. $\mathcal{N}(0, \sigma^2_N)$ elements. For each realization, we solve the NTF problem with (1) NALS (parafac), (2) NLS (sdf_nls), and (3) ADMoM.

We designed our experiments so that, upon convergence, all algorithms achieve practically the same relative factorization error. Towards this end, we set the values of the stopping parameters as follows: the

---

8In certain cases, it may be possible to employ algebraic initialization schemes (e.g., see [16] and references therein), but for NTF we observed that these perform (very) well only in (very) high SNR cases.
parameter Options(1) of parafac is set to Options(1) = 10^{-5}, the parameter TolFun of sdf.nls is set to TolFun = 10^{-8}, and the ADMoM stopping parameters are set to $\epsilon_{\text{abs}} = 10^{-4}$ and $\epsilon_{\text{rel}} = 10^{-4}$.

In all cases, the initial values of the ADMoM penalty terms are $\rho_M = 1$, for $M = A, B, C$, while the ADMoM penalty term adaptation parameters are $\mu = 8$, $\tau_{\text{incr}} = 4$, $\tau_{\text{decr}} = 2$.

In practice, convergence properties of ADMoM NTF depend on the (random) initialization point. In some cases, convergence may be quite fast while, in others, it may be quite slow. As we shall see in the sequel, this phenomenon seems more prominent in the cases where rank $F$ is large. In order to overcome the slow convergence properties associated with bad initial points, we adopted the following strategy. We execute ADMoM NTF for up to $n_{\text{max}} = 400$ iterations (we have observed that, in the great majority of the cases in the scenarios we examined, this number of iterations is sufficient for convergence when we start from a good initial point). If ADMoM does not converge within $n_{\text{max}}$ iterations, then we restart it from another random initial point; we repeat this procedure until ADMoM converges.

Before proceeding, we mention that all the algorithms converged in all the realizations we run.

Since an accurate statement about the computational complexity per iteration of parafac is not easy, the metric we used for comparison of the algorithms is the cpuTime of Matlab. Despite the fact that cpuTime is strongly dependent on the computer hardware and the actual algorithm implementation, we

\[\begin{array}{cccccccccc}
\text{Size} & F & \sigma_N^2 & \text{mean(RFE)} & \text{NALS mean(t)} & \text{NLS mean(t)} & \text{ADMoM mean(t)} & \text{NALS std(t)} & \text{NLS std(t)} & \text{ADMoM std(t)} \\
3000 \times 50 \times 50 & 3 & 10^{-2} & 0.2156 & 12.4010 & 17.2584 & 7.1806 & 1.4770 & 6.9348 & 3.8647 \\
& & 10^{-4} & 0.0221 & 16.6500 & 16.5962 & 7.5098 & 1.9625 & 3.3589 & 4.2499 \\
& 10^{-2} & 0.0260 & 212.0598 & 115.7030 & 110.4128 & 11.4579 & 8.0409 & 91.4882 \\
& 10^{-4} & 0.0026 & 270.6674 & 117.2152 & 148.5052 & 11.5324 & 11.3154 & 149.0542 \\
400 \times 400 \times 50 & 3 & 10^{-2} & 0.2175 & 8.6174 & 4.7124 & 8.0710 & 0.9264 & 1.4596 & 4.4952 \\
& 10^{-4} & 0.0222 & 11.0670 & 4.8916 & 8.3614 & 1.2075 & 1.6443 & 3.1833 \\
& 10^{-2} & 0.0260 & 71.1950 & 31.7546 & 106.1362 & 8.2119 & 3.2162 & 76.5743 \\
& 10^{-4} & 0.0026 & 92.4838 & 31.1690 & 94.5734 & 7.9280 & 3.7062 & 88.1671 \\
200 \times 200 \times 200 & 5 & 10^{-2} & 0.1400 & 10.9142 & 4.1756 & 12.5190 & 1.2783 & 0.6915 & 2.1334 \\
& 10^{-4} & 0.0143 & 14.2882 & 4.1070 & 12.6184 & 2.5817 & 0.9059 & 2.1616 \\
& 10^{-2} & 0.0260 & 55.0806 & 16.1838 & 33.9268 & 4.4886 & 1.3649 & 12.2687 \\
& 10^{-4} & 0.0026 & 70.0238 & 16.7624 & 32.1670 & 6.5737 & 1.4152 & 10.0182
\end{array}\]

\[\text{TABLE I}
\text{MEAN RELATIVE FACTORIZATION ERROR AND MEAN AND STANDARD DEVIATION OF cpuTime, IN sec, FOR NALS, NLS AND ADMoM NTF.}\]
Fig. 2. cputime for $I = 3000$, $J = K = 50$, $F = 3$, and $\sigma_N^2 = 10^{-2}$. NALS (blue solid line), NLS (green dashed line), ADMoM (red dotted-dashed line).

Fig. 3. cputime for $I = 3000$, $J = K = 50$, $F = 30$ and $\sigma_N^2 = 10^{-2}$. NALS (blue solid line), NLS (green dashed line), ADMoM (red dotted-dashed line).
TABLE II
MEAN RELATIVE FACTORIZATION ERROR AND MEAN AND STANDARD DEVIATION OF CPU TIME, IN SEC, FOR NALS, NLS AND ADMoM NTF FOR $I = 10^4$, $J = K = 50$, $F = 10$, AND $\sigma^2_N = 10^{-2}$.

<table>
<thead>
<tr>
<th></th>
<th>NALS mean(RFE)</th>
<th>NLS mean(RFE)</th>
<th>ADMoM mean(RFE)</th>
<th>NALS mean(t)</th>
<th>NLS mean(t)</th>
<th>ADMoM mean(t)</th>
<th>NALS std(t)</th>
<th>NLS std(t)</th>
<th>ADMoM std(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0752</td>
<td>0.0758</td>
<td>0.0751</td>
<td>116.0722</td>
<td>201.6556</td>
<td>30.6460</td>
<td>19.8483</td>
<td>38.5401</td>
<td>16.5823</td>
</tr>
</tbody>
</table>

feel that it is a useful metric for the assessment of the relative efficiency of the algorithms.\textsuperscript{10} The reason is that we used carefully developed, publicly available Matlab toolbox implementations of the baseline algorithms, and we carefully coded our ADMoM NTF implementation.

In Table I, we present the mean and standard deviation of cpu time, in seconds, denoted as mean(t) and std(t), respectively, for NALS, NLS, and ADMoM. We also present the mean relative factorization error (which is common to all algorithms up to four decimal digits), defined as

$$\text{mean}(\text{RFE}) := \frac{1}{n} \sum_{k=1}^{n} \frac{\|X_k - [A_k, B_k, C_k]\|_F}{\|X_k\|_F},$$

where $X_k$ is the $k$-th noisy tensor realization and $A_k$, $B_k$, and $C_k$ are the factors returned by a factorization algorithm. Our observations are as follows:

1) There is no clear winner. Certainly, for high ranks, NLS has very good behavior.

2) In general, both NALS and NLS have more predictable behavior than ADMoM. Especially for high ranks, the cpu time of our implementation of ADMoM has large variance.

3) For small ranks, ADMoM looks more competitive and, in the cases where one dimension is much larger than the other two, it behaves very well (we shall say more on this later).

In order to get a better feeling of the behavior of the three algorithms, we plot their cpu time, along the 50 realizations we used to obtain the averages of Table I, for two different scenarios. In Figure 2, we consider the case for $I = 3000$, $J = K = 50$, $F = 3$ and $\sigma^2_N = 10^{-2}$. We observe that the behavior of the algorithms is stable, in the sense that there is a clear ordering among the three algorithms, with no large variations. In Figure 3, we keep the dimensions and the noise power the same as before and increase the rank to $F = 30$. We observe that the variance of ADMoM cpu time has significantly increased, while both NALS and NLS show stable behavior. When ADMoM starts from a good initial point, it converges faster than NALS and NLS while, when it starts from bad initial points, it needs one or more restarts.

In order to check if ADMoM maintains its advantage over NALS and NLS in the cases where one dimension is very large, compared with the other two, and the rank is relatively small, we performed an experiment with $I = 10^4$, $J = K = 50$, $F = 10$, and $\sigma^2_N = 10^{-2}$. However, in this case, we used\textsuperscript{10}For our experiments, we run Matlab 2014a on a MacBook Pro with a 2.5 GHz Intel Core i7 Intel processor and 16 GB RAM.
somewhat relaxed stopping conditions for all algorithms; more specifically, we used $\text{Options}(1) = 10^{-3}$, $\text{TolFun} = 10^{-6}$, $\epsilon_{\text{abs}} = 10^{-3}$, and $\epsilon_{\text{rel}} = 10^{-3}$. In Table II, we present the mean relative factorization errors and the mean and standard deviation of $\text{cputime}$. As we can see, both NALS and NLS are slightly less accurate than ADMoM, in terms of relative factorization error, which means that their stopping criteria are more relaxed. In terms of $\text{cputime}$, we see that ADMoM is much faster than both NALS and NLS. In Figure 4, we plot the $\text{cputime}$ of the three algorithms for the 50 realizations of the experiment. Again, we see the significant difference between ADMoM and both NALS and NLS. We note that if we had used as values of the stopping parameters those of our initial experiments, then the gain of ADMoM, compared with NALS and NLS, would have been much greater. However, we believe that we have made clear that, in this case, ADMoM has a clear advantage. We have made analogous observations for larger $I$.

B. A closer look at ADMoM

In order to get a more detailed view of the convergence properties of ADMoM, we return to the scenario with $I = 3000$, $J = K = 50$, $F = 30$, and $\sigma^2_N = 10^{-2}$, whose $\text{cputime}$ we plot in Figure 3. We recall that, in order to converge in this case, ADMoM needed often restarts. In Figure 5, we plot the total number of ADMoM iterations, denoted as $\text{iters}$, and the number of ADMoM iterations during its final way to convergence, which is equal to $\text{mod(\text{iters}, n_{\text{max}})}$. As expected, $\text{iters}$ is compatible with
the corresponding cpu-time (see the red line in Figure 3). Quantity $\text{mod}(\text{iters}, n_{\text{max}})$ shows how many iterations are required for convergence if ADMoM always starts from good initial points. We observe that $\text{mod}(\text{iters}, n_{\text{max}})$ is quite stable around its mean, which is approximately equal to 320. This gives an estimate of the fastest possible ADMoM convergence in this case.

C. **ADMoM NTF with under- and over-estimated rank**

In the sequel, we consider ADMoM behavior in the cases where we under- or over-estimate the true rank, in both noisy and noiseless cases. Towards this end, we fix $I = J = K = 100$ and $F = 30$, and investigate ADMoM with exact rank as well as with rank under- and over-estimated by 1. We expect that, in this case, all versions of ADMoM may need restarts. In the sequel, we examine the influence of under- and over-estimating the rank on (1) factorization accuracy and (2) number of restarts. Of course, in under-modeled cases, we expect that the relative factorization error will be higher than that of the true rank case. However, we know nothing in advance about ADMoM behavior in over-modeled cases. In order to get insight into these issues, we perform the following experiment. We set stopping parameters $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-4}$ and run each of the three versions of ADMoM for $n_{\text{max}} = 500$ iterations. For each version, we proceed as follows: if it converges within $n_{\text{max}}$ iterations, we stop; otherwise, we restart, and repeat until convergence. Thus, finally, the number of iterations for each ADMoM version will be a multiple of $n_{\text{max}}$. For the computation of the trajectory of the mean relative factorization error we use
only the last $n_{\text{max}}$ values; in this way, we avoid the influence of bad initial points. However, we keep count of the restarts of each version and, thus, can assess the time it needs to achieve convergence.

In Figure 6, we plot the average relative factorization errors (computed over 50 realizations in the way we mentioned before), versus the iteration number, for $\sigma_N^2 = 10^{-2}$. As was expected, the ADMoM version with...
under-estimated rank converges to a higher relative factorization error. We observe that the average relative factorization errors for ADMoM with exact rank and rank over-estimated by 1 follow almost the same trajectory. The average numbers of restarts for the three ADMoM versions are $\text{mean}(\text{restarts}_{\text{exact}}) = 1.08$, $\text{mean}(\text{restarts}_{\text{under}}) = 1.22$, and $\text{mean}(\text{restarts}_{\text{over}}) = 1.92$. Thus, in the cases of over-estimated rank, we finally achieve a relative factorization error trajectory as good as in the exact rank case, but we may need more restarts and, thus, more time. This implies that the probability of bad initial points may increase.

In Figure 7, we plot the same quantities for noiseless data. Again, the ADMoM behavior in the under-modeled case is as expected. Interestingly, we observe that there is no relative factorization error floor neither for the exact rank nor for the over-estimated by 1 rank case. Reasonably, after a certain precision level, the over-modeled case converges slower. The average numbers of restarts for the three ADMoM versions are $\text{mean}(\text{restarts}_{\text{exact}}) = 1.14$, $\text{mean}(\text{restarts}_{\text{under}}) = 1.54$, and $\text{mean}(\text{restarts}_{\text{over}}) = 1.06$.

We have observed similar behavior for more drastic rank under- and over-estimation.

D. ADMoM for NTF with box-linear constraints

In our final experiment, we briefly consider NTF for the case where two of the latent factors, say $A$ and $B$, are non-negative while $C$ is subject to box-linear constraints in the sense that each row of $C$ is a probability mass function, that is, has non-negative elements with sum equal to 1.

The only difference between the ADMoM for this case and the ADMoM for NTF is that, instead of
computing $C^{k+1}$ as the solution of an unconstrained least-squares problem, we compute it as the solution of linearly constrained least-squares; note that both cases exhibit closed-form solutions.

In Figure 8, we illustrate the behavior of ADMoM in this case by plotting the trajectories of the norms of the average (over 50 realizations) relative estimation errors of the latent factors, as computed by function `cpderr` of tensorlab, versus the iteration number, for a noiseless case with $I = J = 100$, $K = 50$ and $F = 5$. We observe that ADMoM works to very high precision.

E. Discussion

Our numerical results are encouraging and suggest that, in many cases, ADMoM NTF can efficiently achieve close to state-of-the-art factorization accuracy. The fact that ADMoM is suitable for high-performance parallel implementation (the first NTF algorithm with this property, as far as we know) can only increase its potential. Thus, we believe that it will be a valuable tool in the NTF toolbox.

Obviously, in order to fully uncover the pros and cons of ADMoM NTF, more extensive experimentation is required. But our intention in this paper is to give the fundamental ideas and some basic performance metrics. Experiments with real-world data (using ADMoM for tensor completion and factorization) as well as constraints well beyond non-negativity are ongoing work.

A weak point of the version of ADMoM we developed in this manuscript is the high `cpu time` variance in cases of high rank. The improvement of the behavior of ADMoM in these cases remains a very interesting problem. To achieve this goal, it might be possible to combine elements of NLS and ADMoM and derive a more efficient algorithm. However, more research efforts are needed in this direction.

As we mentioned, if the centralized and the distributed algorithms start from the same initial point, they evolve in exactly the same way. Thus, distributed ADMoM inherits the convergence properties of centralized ADMoM.

VII. Conclusion

Motivated by emerging big data applications, involving multi-way tensor data, and the ensuing need for scalable tensor factorization tools, we developed a new constrained tensor factorization framework based on the ADMoM. We used non-negative factorization of third order tensors as an example to work out the main ideas, but our approach can be generalized to higher order tensors, many other types of constraints on the latent factors, as well as other tensor factorizations and tensor completion. Our numerical experiments were encouraging, indicating that, in many cases, the ADMoM-based NTF has high potential as an alternative to the state-of-the-art and, in some cases, it may become state-of-the-art. The fact that it is naturally amenable to parallel implementation can only increase its potential. The improvement of its behavior in the high rank cases remains a very interesting problem.
APPENDIX A

EXTENSION TO HIGHER ORDER TENSORS

In this appendix, we highlight how our approach can be extended to higher order tensors. We focus on fourth-order tensors, with the general case being obvious. If \( \mathbf{W} = [\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}] \), then its matrix unfoldings satisfy relations

\[
\begin{align*}
\mathbf{W}^{(1)} &= \mathbf{A} \left( \mathbf{D} \odot (\mathbf{C} \odot \mathbf{B}) \right)^T, \\
\mathbf{W}^{(2)} &= \mathbf{B} \left( \mathbf{D} \odot (\mathbf{C} \odot \mathbf{A}) \right)^T, \\
\mathbf{W}^{(3)} &= \mathbf{C} \left( \mathbf{D} \odot (\mathbf{B} \odot \mathbf{A}) \right)^T, \\
\mathbf{W}^{(4)} &= \mathbf{D} \left( \mathbf{C} \odot (\mathbf{B} \odot \mathbf{A}) \right)^T.
\end{align*}
\]

Partitioning matrices \( \mathbf{A}, \mathbf{B}, \mathbf{C}, \) and \( \mathbf{D} \) as in subsection V-A, we obtain that matrix \( \mathbf{W}^{(1)} \) can be partitioned into \( N_A \times N_D \) blocks, with the \((i,j)\)-th block being equal to

\[
\mathbf{W}^{(1)}_{i,j} = \mathbf{A}_i \left( \mathbf{D}_j \odot (\mathbf{C}_i \odot \mathbf{B}) \right)^T.
\]

Analogous partitionings apply to the other matrix unfoldings. Then, development of ADMoM NTF (centralized and distributed) is rather easy.

APPENDIX B

ON THE EQUIVALENCE OF THE CENTRALIZED AND THE DISTRIBUTED ADMoM NTF

A simple proof of the equivalence of the centralized and the distributed ADMoM NTF is as follows. We focus on the update of \( \mathbf{A}^k \) of the centralized algorithm and the updates of its blocks, \( \mathbf{A}_{n_A}^k \), for \( n_A = 1, \ldots, N_A \), of the distributed algorithm, and prove that they are equivalent. We remind that

\[
\begin{align*}
\mathbf{A}^{k+1} &= \left( \mathbf{X}^{(1)}(\mathbf{C}^k \odot \mathbf{B}^k) + \rho_{\mathbf{A}} \tilde{\mathbf{A}}^k - \mathbf{Y}_A^k \right) \\
&\quad \left( (\mathbf{C}^k \odot \mathbf{B}^k)^T(\mathbf{C}^k \odot \mathbf{B}^k) + \rho_{\mathbf{A}} \mathbf{I} \right)^{-1}.
\end{align*}
\]

Using the partitionings of \( \mathbf{C}^k \odot \mathbf{B}^k \) (see subsection V-A), it can be shown that

\[
(\mathbf{C}^k \odot \mathbf{B}^k)^T(\mathbf{C}^k \odot \mathbf{B}^k) = \sum_{n_C=1}^{N_C} (\mathbf{C}^k_{n_C} \odot \mathbf{B}^k)^T(\mathbf{C}^k_{n_C} \odot \mathbf{B}^k).
\]

Rewriting the update of \( \mathbf{A}^k \) in terms of partitioned matrices, we obtain

\[
\begin{align*}
\begin{bmatrix}
\mathbf{A}_1^{k+1} \\
\vdots \\
\mathbf{A}_{N_A}^{k+1}
\end{bmatrix}
&= \left( \begin{bmatrix}
\mathbf{X}_1^{(1)} \\
\vdots \\
\mathbf{X}_{N_A}^{(1)}
\end{bmatrix}
\begin{bmatrix}
\mathbf{C}_1^k \odot \mathbf{B}^k \\
\vdots \\
\mathbf{C}_{N_A}^k \odot \mathbf{B}^k
\end{bmatrix}
\right)
+ \left( \begin{bmatrix}
\rho_{\mathbf{A}} \tilde{\mathbf{A}}_1^k - \mathbf{Y}_A^k \\
\vdots \\
\rho_{\mathbf{A}} \tilde{\mathbf{A}}_{N_A}^k - \mathbf{Y}_A^{k, -1}
\end{bmatrix}
\right)
\times \left( \sum_{n_C=1}^{N_C} (\mathbf{C}^k_{n_C} \odot \mathbf{B}^k)^T(\mathbf{C}^k_{n_C} \odot \mathbf{B}^k) + \rho_{\mathbf{A}} \mathbf{I} \right)^{-1}.
\end{align*}
\]
If we focus on a certain block of $A^{k+1}$ in (27), then we obtain the corresponding update of the distributed algorithm (see (26)). We observe that the matrix inverse in the second line of (27) is common to all blocks, and should be computed once.

Analogous statements hold for the updates of $B^k$ and $C^k$. The equivalence of the updates of the rest of the variables is trivial.

Thus, in fact, using the partitionings of subsection V-A, the distributed ADMoM simply uncovered the inherent parallelism of the centralized ADMoM.

REFERENCES


