Parallel Algorithms for Constrained Tensor Factorization via the 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 nonnegativity. 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 on regular high-performance computing (e.g., mesh) architectures. This opens the door for many emerging big data-enabled applications. The methodology is exemplified using nonnegativity as a baseline constraint, but the proposed framework can more-or-less readily incorporate many other types of constraints. Numerical experiments are very encouraging, indicating that the ADMoM-based nonnegative tensor factorization (NTF) has high potential as an alternative to state-of-the-art approaches.
I. INTRODUCTION

Tensors (multi-way arrays) \cite{2}, \cite{3}, \cite{4} are data structures indexed by three or more indices, say \((i,j,k,\cdots)\). Matrices are two-way arrays indexed by (row,column) \((i,j)\).

Tensor algebra is similar to matrix algebra in many ways, but there are also some fundamental (and initially surprising) differences. Tensor factorization has proven useful in a wide range of applications, including many signal processing ‘staples’ - such as direction of arrival estimation \cite{5}, communication signal intelligence \cite{6}, and speech and audio signal separation \cite{7}, \cite{8}, as well as cross-disciplinary topical areas, such as community detection in social networks \cite{9}, and chemical signal analysis \cite{10}. More recently, there has been a flurry of activity in applying tensor factorization theory and methods to problems in machine learning research - see \cite{11}.

There are many useful matrix factorizations, ranging from the ubiquitous singular value decomposition (SVD), to Choleski, QR, LU and many others. These differ in the constraints they impose on the constituent factors, but share one important characteristic: they can be viewed as decomposing a given matrix into a sum of outer products (rank-one factors). Note that the terms factorization and decomposition are often used interchangeably, even though the latter alludes to a complete decomposition, whereas the former may include a residual term. Also note that for matrices we have many factorizations for a given rank, because rank decomposition is generically non-unique - even when the rank of the matrix that we decompose is low relative to the maximal possible rank. For tensors, the situation is very different. The reason is that low-rank tensor decomposition is unique under certain conditions \cite{12}–\cite{13}, so we do not have the freedom to rotate the factors. Hence, if we insist on low rank there is essentially only one decomposition; whereas if we desire an orthogonal decomposition (similar to the matrix SVD) we must over-parameterize, and thus lose uniqueness. There are therefore two basic tensor factorization models: parallel factor analysis (PARAFAC) \cite{14}, \cite{15} also known as canonical decomposition (CANDECOMP) \cite{16}, or CP (and CPD) for CANDECOMP-PARAFAC (Decomposition), or canonical polyadic decomposition (CPD, again); and the Tucker3 model. Both are sum-of-outer-products models.

CP is non-orthogonal but unique, under certain conditions; whereas orthogonality comes without loss of generality in Tucker3. CP, where appropriate, can be used for latent parameter estimation; Tucker3 is generally used for data compression and imputation, where uniqueness of the latent parameters is not a concern. In this paper, we will primarily focus on the CP model.

Whereas for low-enough rank CP is already unique ‘on its own,’ any side information we may have
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 [3]. 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 [17]. As a result, most tensor factorization algorithms rely on alternating optimization, usually alternating least squares (ALS), and imposing e.g., non-negativity and/or sparsity entails replacing linear conditional least squares with non-negative and/or sparse least squares problems.

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 [18], [19] implementation of the basic ALS CP-decomposition algorithm was developed in [20], which reported 100-fold scaling improvements relative to the prior art. The jist of [20] 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, [20] 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 [9], motivated by recent progress in randomized algorithms for matrix algebra. The idea of [9] 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 [9] 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 [21], [22]. 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 [21], [22] over [9] 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 [23], and more recently in [24], 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 [23], [24].

In this paper, we develop efficient algorithms for constrained tensor factorization based on the Alternating Direction method of Multipliers (ADMoM). The ADMoM has recently attracted renewed interest [25], primarily for solving certain types of convex optimization problems in a distributed fashion. However, it can also be used to tackle certain non-convex problems, such as non-negative matrix factorization [25], albeit its convergence properties are far less understood in this case. We focus on non-negative CP decomposition (sometimes referred to as non-negative tensor factorization) as a working problem, due to the importance of the CP model and non-negativity constraints; but our approach can be more-or-less readily generalized to many other types of constraints on the latent factors, as well as other tensor factorizations, such as Tucker3.

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 with synthetic data are very encouraging, indicating that the ADMoM-based NTF has high 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 ADMoM. In Section IV we develop the ADMoM for NTF, while in Section V we develop a distributed ADMoM for large NTF. In Section VI we use synthetic data and 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, \( \mathbf{x} \), \( \mathbf{X} \), and \( \underline{\mathbf{X}} \). \( \mathbb{R}^{I\times J\times K}_+ \) denotes the set of \( (I \times J \times K) \) real nonnegative tensors, while \( \mathbb{R}^{I\times J}_+ \) denotes the set of \( (I \times J) \) real nonnegative matrices. \( \| \cdot \|_F \) denotes the Frobenius norm of the tensor or matrix argument, \( \mathbf{A}^\dagger \) denotes the Moore-Penrose pseudoinverse of matrix \( \mathbf{A} \), and \( (\mathbf{A})_+ \) denotes the projection of matrix \( \mathbf{A} \) onto the set of element-wise nonnegative matrices. The outer product of two vectors \( \mathbf{a} \in \mathbb{R}^{I\times 1} \) and \( \mathbf{b} \in \mathbb{R}^{J\times 1} \) is the rank-one matrix \( \mathbf{a} \circ \mathbf{b} \in \mathbb{R}^{I\times J} \) with elements
\((a \circ b)(i, j) = a(i)b(j)\). Similarly, the outer product of three vectors \(a \in \mathbb{R}^{I \times 1}\), \(b \in \mathbb{R}^{J \times 1}\), and \(c \in \mathbb{R}^{K \times 1}\) is the rank-one tensor \(a \circ b \circ c \in \mathbb{R}^{I \times J \times K}\) with elements \((a \circ b \circ c)(i, j, k) = a(i)b(j)c(k)\). Finally, \(A \odot B\) denotes the Khatri-Rao (columnwise Kronecker) product and \(A \oplus B\) denotes the Hadamard (elementwise) product.

## II. NONNEGATIVE TENSOR FACTORIZATION

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

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

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

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

In order to compute estimates of \(A^o, B^o,\) 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

\[
\min_{A,B,C} \quad f_{X}(A,B,C)
\]

subject to \(A \geq O, B \geq O, C \geq O,\)

where \(f\) is a function measuring the quality of the factorization, and the inequalities are element-wise. A common choice of \(f_X\) (motivated via maximum likelihood estimation for Gaussian i.i.d. \(E\)) is

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

Let \(Y = [A, B, C]\) and, for every tensor \(W\), let \(W^{(1)}, W^{(2)},\) and \(W^{(3)}\) denote the matrix unfoldings of \(W\), with respect to the first, second, and third dimension, respectively. Then,

\[
Y^{(1)} = A \ (C \odot B)^T, \ Y^{(2)} = B \ (C \odot A)^T, \ Y^{(3)} = C \ (B \odot A)^T,
\]

and \(f_X\) can be equivalently expressed as

\[
f_{X}(A,B,C) = \frac{1}{2} \left\| X^{(1)} - A \ (C \odot B)^T \right\|_F^2 = \frac{1}{2} \left\| X^{(2)} - B \ (C \odot A)^T \right\|_F^2 = \frac{1}{2} \left\| X^{(3)} - C \ (B \odot A)^T \right\|_F^2.
\]

\(^1\)Note that due to the nonnegativity constraints on the latent factors, \(F\) can be higher than the rank of \(X^o\).

September 9, 2014 DRAFT
These expressions are the basis for alternating least squares (ALS)-type CP optimization, because they enable simple linear least squares updating of one matrix given the other two. Using nonnegative least squares for each update step (nonnegative ALS, or NALS) is the most popular approach for the solution of (1), but nonnegativity brings a significant computational burden relative to plain ALS, and it 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. In this work, we derive ADMoM-based algorithms for NTF whose complexity per iteration is small compared to NALS and, perhaps more importantly, they are suitable for parallel implementation.

III. The ADMoM

The ADMoM is an iterative technique for the solution of optimization problems of the form

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

subject to \( Ax + Bz = b \),

where \( x \in \mathbb{R}^{n_1}, \ z \in \mathbb{R}^{n_2}, \ A \in \mathbb{R}^{m \times n_1}, \ B \in \mathbb{R}^{m \times n_2}, \ b \in \mathbb{R}^{m}, \) and \( f: \mathbb{R}^{n_1} \rightarrow \mathbb{R}, \) and \( g: \mathbb{R}^{n_2} \rightarrow \mathbb{R}. \)

The augmented Lagrangian for problem (4) is

\[
L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - b) + \frac{\rho}{2} \|Ax + Bz - b\|_2^2.
\]

A modification which often leads to simplified expressions is to introduce the scaled dual variable \( u = \frac{1}{\rho}y. \)

Then, the augmented Lagrangian becomes

\[
L_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2} \|Ax + Bz - b + u\|_2^2 - \frac{\rho}{2} \|u\|_2^2.
\]

Assuming that at time instant \( k \) we have computed \( z^k \) and \( u^k \), which comprise the state of the algorithm, the \((k + 1)\)-st iteration of the (scaled dual variable) ADMoM is as follows

\[
x^{k+1} = \arg\min_x \left( f(x) + \frac{\rho}{2} \|Ax + Bz^k - b + u^k\|_2^2 \right)
\]

\[
z^{k+1} = \arg\min_z \left( g(z) + \frac{\rho}{2} \|Ax^{k+1} + Bz - b + u^k\|_2^2 \right)
\]

\[
u^{k+1} = u^k + Ax^{k+1} + Bz^{k+1} - b.
\]

It can be shown that, under certain conditions (among them, convexity of \( f \) and \( g \)), the ADMoM converges in a certain sense (see [25] for an excellent review of the ADMoM, including some convergence analysis results). However, the ADMoM can be used even when problem (4) is not convex. In this case, we can use the ADMoM with the goal of reaching a good local minimum [25]. Note that this is all we can realistically hope for anyway, irrespective of approach or algorithm used, as tensor factorization is NP-hard [17].
A. ADMoM for set-constrained optimization

Let us consider the set-constrained optimization problem

\[
\begin{align*}
\min_{x} & \quad f(x) \\
\text{subject to} & \quad x \in \mathcal{X},
\end{align*}
\]

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

\[
\begin{align*}
\min_{x, z} & \quad f(x) + g(z) \\
\text{subject to} & \quad x - z = 0,
\end{align*}
\]

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

\[
g(z) = I_{\mathcal{X}}(z) = \begin{cases} 0, & z \in \mathcal{X}, \\ \infty, & z \not\in \mathcal{X}, \end{cases}
\]

and then it becomes clear that (III-A) can be solved via the ADMoM. Assuming that at time instant \( k \) we have computed \( z^k \) and the scaled dual variable \( u^k \), the \((k+1)\)-st iteration of the ADMoM is as follows

\[
\begin{align*}
x^{k+1} &= \operatorname{argmin}_x \left( f(x) + \frac{\rho}{2} \|x - z^k + u^k\|^2_2 \right) \\
z^{k+1} &= \operatorname{argmin}_z \left( g(z) + \frac{\rho}{2} \|z^{k+1} - z + u^k\|^2_2 \right) \\
&= \Pi_{\mathcal{X}}(x^{k+1} + u^k) \\
u^{k+1} &= u^k + x^{k+1} - z^{k+1},
\end{align*}
\]

where \( \Pi_{\mathcal{X}} \) is the projection operator onto set \( \mathcal{X} \).

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 (I) into ADMoM form. Towards this end, we introduce variables \( \tilde{A} \in \mathbb{R}^{I \times F}, \tilde{B} \in \mathbb{R}^{J \times F}, \) and \( \tilde{C} \in \mathbb{R}^{K \times F} \) and consider the equivalent optimization problem

\[
\begin{align*}
\min_{A, \tilde{A}, B, \tilde{B}, C, \tilde{C}} & \quad f_X(A, B, C) + g(\tilde{A}) + g(\tilde{B}) + g(\tilde{C}) \\
\text{subject to} & \quad A - \tilde{A} = O, \quad B - \tilde{B} = O, \quad C - \tilde{C} = O,
\end{align*}
\]

where, for any matrix argument \( M \),

\[
g(M) = \begin{cases} 0, & \text{if } M \geq O, \\ \infty, & \text{otherwise}. \end{cases}
\]
If we introduce the scaled dual variables $U_A \in \mathbb{R}^{I \times F}$, $U_B \in \mathbb{R}^{J \times F}$, and $U_C \in \mathbb{R}^{K \times F}$, we can express the augmented Lagrangian as

$$L_\rho(A, B, C, \tilde{A}, \tilde{B}, \tilde{C}, U_A, U_B, U_C) = f_\mathcal{X}(A, B, C) + g(\tilde{A}) + g(\tilde{V}) + g(\tilde{C})$$

$$+ \frac{\rho}{2} \| A - \tilde{A} + U_A \|_F^2 - \frac{\rho}{2} \| U_A \|_F^2$$

$$+ \frac{\rho}{2} \| B - \tilde{B} + U_B \|_F^2 - \frac{\rho}{2} \| U_B \|_F^2$$

$$+ \frac{\rho}{2} \| C - \tilde{C} + U_C \|_F^2 - \frac{\rho}{2} \| U_C \|_F^2. $$

(7)

The ADMoM for this problem is as follows:

$$(A^{k+1}, B^{k+1}, C^{k+1}) = \text{argmin}_{A,B,C} \left( f_\mathcal{X}(A, B, C) \right.$$}

$$+ \frac{\rho}{2} \| A - \tilde{A}^k + U_A^k \|_F^2$$

$$+ \frac{\rho}{2} \| B - \tilde{B}^k + U_B^k \|_F^2$$

$$+ \frac{\rho}{2} \| C - \tilde{C}^k + U_C^k \|_F^2 \right)$$

(8)

$$\tilde{A}^{k+1} = \left( A^{k+1} + U_A^k \right)_+$$

$$\tilde{B}^{k+1} = \left( B^{k+1} + U_B^k \right)_+$$

$$\tilde{C}^{k+1} = \left( C^{k+1} + U_C^k \right)_+$$

$$U_A^{k+1} = U_A^k + A^{k+1} - \tilde{A}^{k+1}$$

$$U_B^{k+1} = U_B^k + B^{k+1} - \tilde{B}^{k+1}$$

$$U_C^{k+1} = U_C^k + C^{k+1} - \tilde{C}^{k+1}.$$

The minimization problem in the first line of (8) is NP-hard. Using the equivalent expressions for $f_\mathcal{X}$ in (2), we propose the alternating optimization scheme

$$A^{k+1} = \text{argmin}_A \left( \frac{1}{2} \| X^{(1)} - A(C^k \odot B^k)^T \|_F^2 + \frac{\rho}{2} \| A - \tilde{A}^k + U_A^k \|_F^2 \right)$$

$$= \left( X^{(1)} (C^k \odot B^k) + \rho(\tilde{A}^k - U_A^k) \right) \left( (C^k \odot B^k)^T (C^k \odot B^k) + \rho I_F \right)^{-1},$$

$$B^{k+1} = \text{argmin}_B \left( \frac{1}{2} \| X^{(2)} - B(C^k \odot A^{k+1})^T \|_F^2 + \frac{\rho}{2} \| B - \tilde{B}^k + U_B^k \|_F^2 \right)$$

$$= \left( X^{(2)} (C^k \odot A^{k+1}) + \rho(\tilde{B}^k - U_B^k) \right) \left( (C^k \odot A^{k+1})^T (C^k \odot A^{k+1}) + \rho I_F \right)^{-1},$$

$$C^{k+1} = \text{argmin}_C \left( \frac{1}{2} \| X^{(3)} - C(B^{k+1} \odot A^{k+1})^T \|_F^2 + \frac{\rho}{2} \| C - \tilde{C}^k + U_C^k \|_F^2 \right)$$

$$= \left( X^{(3)} (B^{k+1} \odot A^{k+1}) + \rho(\tilde{C}^k - U_C^k) \right) \left( (B^{k+1} \odot A^{k+1})^T (B^{k+1} \odot A^{k+1}) + \rho I_F \right)^{-1}.$$
Algorithm 1 The ADMoM algorithm for NTF

\begin{align*}
A^{k+1} &= \left( X^{(1)} (C^k \odot B^k) + \rho(\tilde{A}^k - U_A^k) \right) \left( (C^k \odot B^k)^T (C^k \odot B^k) + \rho I_F \right)^{-1} \\
B^{k+1} &= \left( X^{(2)} (C^k \odot A^{k+1}) + \rho(\tilde{B}^k - U_B^k) \right) \left( (C^k \odot A^{k+1})^T (C^k \odot A^{k+1}) + \rho I_F \right)^{-1} \\
C^{k+1} &= \left( X^{(3)} (B^{k+1} \odot A^{k+1}) + \rho(\tilde{C}^k - U_C^k) \right) \left( (B^{k+1} \odot A^{k+1})^T (B^{k+1} \odot A^{k+1}) + \rho I_F \right)^{-1} \\
\tilde{A}^{k+1} &= (A^{k+1} + U_A^k) + \\
\tilde{B}^{k+1} &= (B^{k+1} + U_B^k) + \\
\tilde{C}^{k+1} &= (C^{k+1} + U_C^k) + \\
U_A^{k+1} &= U_A^k + A^{k+1} - \tilde{A}^{k+1} \\
U_B^{k+1} &= U_B^k + B^{k+1} - \tilde{B}^{k+1} \\
U_C^{k+1} &= U_C^k + C^{k+1} - \tilde{C}^{k+1}
\end{align*}

The updates of (9) can be executed either for a predetermined number of iterations, or until convergence. We observe that, during each ADMoM iteration, we avoid the solution of constrained optimization problems. This seems favourable, especially in the cases where the size of the problem is (very) large.

Assuming that at the \( k \)-th iteration we have computed \( B^k, C^k, \tilde{A}^k, \tilde{B}^k, \tilde{C}^k, U_A^k, U_B^k, \) and \( U_C^k \), which constitute the state of the algorithm, the \((k + 1)\)-st iteration of the ADMoM for NTF is summarized in Algorithm 1.
A. Convergence analysis

Let \( W = (A, B, C, \tilde{A}, \tilde{B}, \tilde{C}, U_A, U_B, U_C) \). It can be shown that \( W \) is a Karush-Kuhn-Tucker (KKT) point for the NTF problem (5) if

\[
\begin{align*}
(X^{(1)} - A(C \odot B)^T)(C \odot B) - \rho U_A &= 0 \\
(X^{(2)} - B(C \odot A)^T)(C \odot A) - \rho U_B &= 0 \\
(X^{(3)} - C(B \odot A)^T)(B \odot A) - \rho U_C &= 0 \\
A - \tilde{A} &= 0, \\
B - \tilde{B} &= 0, \\
C - \tilde{C} &= 0 \\
U_A &\leq 0, \quad U_B \leq 0, \quad U_C \leq 0 \\
U_A \odot \tilde{A} &= 0, \quad U_B \odot \tilde{B} = 0, \quad U_C \odot \tilde{C} = 0.
\end{align*}
\] (10)

**Proposition 1:** Let \( \{W_k\} \) be a sequence generated by the ADMoM for NTF that satisfies the condition

\[
\lim_{k \to \infty} (W_{k+1} - W_k) = 0. \tag{11}
\]

Then, any accumulation point of \( \{W_k\} \) is a KKT point of problem (5). 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 [26] and is omitted here. □

Proposition 1 implies that whenever \( \{W_k\} \) converges, it converges to a KKT point.

B. Initialization

Before the start of the algorithm, we must initialize its state, that is, give values to \( B^0, C^0, \tilde{A}^0, \tilde{B}^0, \tilde{C}^0, U_A^0, U_B^0, \) and \( U_C^0 \). A trivial approach is to give nonnegative random values to \( B^0 \) and \( C^0 \) and set the rest to zero. However, a good initialization significantly improves the behavior of the algorithm. In the sequel, we describe an algebraic initialization approach that is useful when two modes are tall relative to tensor rank; see [28] for detailed conditions and a discussion. More sophisticated algebraic initialization procedures are possible even when the rank exceeds the dimension of every mode, however these are more involved, and would detract from our focus here.

Consider an \( I \times J \times K \) tensor \( \mathbf{X} \) of rank \( F < \min(I, J) \), with \( K \geq 2 \), and its slabs perpendicular to the third mode \( X_k = \mathbf{X}(:, :, k) \) (using Matlab notation : to denote all indices along a certain mode). It

See the report [27] for a detailed proof.
can be easily shown that
\[ X_k = A D_k(C) B^T, \quad k \in \{1, 2, \ldots, K\}, \]
where \( D_k(C) \) is a diagonal matrix holding the \( k \)-th row of \( C \) on its diagonal. Create two random mixtures of these \( K \) slabs as follows
\[
Y_1 = \sum_{k=1}^{K} w_1(k) X_k, \\
Y_2 = \sum_{k=1}^{K} w_2(k) X_k,
\]
where \( w_1, w_2 \) are random i.i.d. (the marginal distribution can be Gaussian or uniform in \([0,1]\)). Then
\[
Y_1 = AD_1B^T, \\
Y_2 = AD_2B^T,
\]
where \( D_1, D_2 \) are diagonal matrices. Due to the scaling ambiguity, we can absorb \( D_1 \) in \( B^T \), and simply use \( D \) in place of \( D_2 \). Hence we obtain the equivalent model
\[
Y_1 = AB^T, \\
Y_2 = ADB^T,
\]
Taking the SVD of \([Y_1^T Y_2^T]^T\), and splitting the matrix of left singular vectors, \( U \), in two \((I \times F)\) pieces \( U_1 \) and \( U_2 \), we obtain (assuming \( A \) and \( B \) are full column rank)
\[
U_1 = AM, \\
U_2 = ADM,
\]
Solving the first for \( A \) and substituting in the second we further obtain
\[
U_2 = U_1 M^{-1} DM,
\]
or, equivalently,
\[
U_1^T U_2 = M^{-1} DM.
\]
Thus, \( M^{-1} \) holds the eigenvectors of \( U_1^T U_2 \). It follows that \( M^{-1} \) can be found up to column permutation and scaling. Using \( M^{-1} \), \( A \) can be found (subject to the same indeterminacies) from \( U_1 = AM \); then \( B \) can be found by solving \( Y_1 = AB^T \), and finally \( C \) can be found by solving a linear least squares problem involving the Khatri-Rao product of \( B \) and \( A \), all subject to column permutation and scaling that carry over from the eigen-decomposition of \( U_1^T U_2 \).
C. ADMoM for NTF with sparsity or structural constraints

It is well understood that the ADMoM can handle certain structural constraints \[25\], \[29\]. For example, if we want to solve an NTF problem with the added constraint that the number of the nonzero elements of \(A\) should be smaller than 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 (6), we must use \(g_{c_A}(\tilde{A})\), with

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

(12)

The only difference in the ADMoM is in the update of \(\tilde{A}^k\). Instead of using projection onto the set of nonnegative matrices, we must use projection onto the set of nonnegative matrices with at most \(c_A\) nonzero elements; this projection can be easily computed through, for example, sorting of the elements of \(\tilde{A}^k\). Using analogous arguments, we can easily incorporate into the ADMoM framework many structural constraints. The development and the study of specific algorithms remain topics of future research.

V. ADMoM FOR VERY LARGE NTFs

In this section, we assume that all dimensions of tensor \(X\) are very large and derive an ADMoM-based NTF that is suitable for parallel implementation. Of course, cases where only one or two of the dimensions of \(X\) are very large can be handled similarly.\[3\]

We first partition matrices \(A\), \(B\), and \(C\) as follows:

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

(13)

where \(A_{n_A} \in \mathbb{R}^{I_{n_A} \times F}\), 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 \(n_C = 1, \ldots, N_C\), \(\sum_{n_C=1}^{N_C} K_{n_C} = K\).

A. Matrix unfoldings in terms of partitioned matrix factors

Let \(Y = [A, B, C]\). It turns out that, in order to derive the distributed ADMoM for very large NTFs, we must derive partitionings of the matrix unfoldings, \(Y^{(1)}, Y^{(2)},\) and \(Y^{(3)}\), in terms of (the blocks of) matrices \(A\), \(B\), and \(C\).

\[\text{See }[27]\text{ for a detailed discussion of the case where only one dimension is very large.}\]
We start by partitioning $Y^{(1)}$ as

$$
Y^{(1)} = \begin{bmatrix} Y_{1,1}^{(1)} & \cdots & Y_{1,N_c}^{(1)} \\ \vdots & \ddots & \vdots \\ Y_{N_A,1}^{(1)} & \cdots & Y_{N_A,N_C}^{(1)} \end{bmatrix},
$$

where $Y_{n_A,n_C}^{(1)} \in \mathbb{R}^{I_{n_A} \times (JK_{n_C})}$, for $n_A = 1, \ldots, N_A$ and $n_C = 1, \ldots, N_C$. In order to get explicit expressions for the blocks of $Y^{(1)}$ in terms of (the blocks of) $A$, $B$, and $C$, we express $Y^{(1)}$ as

$$
Y^{(1)} = A(C \odot B)^T = \begin{bmatrix} A_1 \\ \vdots \\ A_{N_A} \end{bmatrix} \left( \begin{bmatrix} C_1 \\ \vdots \\ C_{N_C} \end{bmatrix} \odot B \right)^T
$$

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

Thus, the $(n_A, n_C)$-th block of $Y^{(1)}$, for $n_A = 1, \ldots, N_A$ and $n_C = 1, \ldots, N_C$, is equal to the $I_{n_A} \times (JK_{n_C})$ matrix $Y_{n_A,n_C}^{(1)} = A_{n_A}(C_{n_C} \odot B)^T$.

Similarly, it can be shown that $Y^{(2)}$ can be partitioned into blocks $Y_{n_B,n_C}^{(2)} = B_{n_B}(C_{n_C} \odot A)^T$, of dimensions $J_{n_B} \times (IK_{n_C})$, for $n_B = 1, \ldots, N_B$ and $n_C = 1, \ldots, N_C$. Finally, $Y^{(3)}$ can be partitioned into blocks $Y_{n_C,n_B}^{(3)} = C_{n_C}(B_{n_B} \odot A)^T$, of dimensions $K_{n_C} \times (IJ_{n_B})$, 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 $f_X$ can be written as

$$
f_X(A,B,C) = \sum_{n_A=1}^{N_A} \sum_{n_C=1}^{N_C} \frac{1}{2} \|X_{n_A,n_C}^{(1)} - A_{n_A}(C_{n_C} \odot B)^T\|^2_F
$$

$$
= \sum_{n_B=1}^{N_B} \sum_{n_C=1}^{N_C} \frac{1}{2} \|X_{n_B,n_C}^{(2)} - B_{n_B}(C_{n_C} \odot A)^T\|^2_F 
$$

$$
= \sum_{n_C=1}^{N_C} \sum_{n_B=1}^{N_B} \frac{1}{2} \|X_{n_C,n_B}^{(3)} - C_{n_C}(B_{n_B} \odot A)^T\|^2_F. \quad (14)
$$

These expressions will be fundamental for the development of the distributed ADMoM for very large NTFs.
B. The ADMoM for very large NTFs

In order to put the very large NTF problem into a form suitable for the ADMoM, we introduce variables $\tilde{A} = [\tilde{A}_1^T \cdots \tilde{A}_{N_A}^T]^T$, with $\tilde{A}_{n_A} \in \mathbb{R}^{I_{n_A} \times F}$, for $n_A = 1, \ldots, N_A$, $\tilde{B} = [\tilde{B}_1^T \cdots \tilde{B}_{N_B}^T]^T$, with $\tilde{B}_{n_B} \in \mathbb{R}^{J_{n_B} \times F}$, for $n_B = 1, \ldots, N_B$, and $\tilde{C} = [\tilde{C}_1^T \cdots \tilde{C}_{N_C}^T]^T$, with $\tilde{C}_{n_C} \in \mathbb{R}^{K_{n_C} \times F}$, for $n_C = 1, \ldots, N_C$, and consider the equivalent problem

$$\min_{A, \tilde{A}, B, \tilde{B}, C} \ f_X(A, B, C) + \sum_{n_A=1}^{N_A} g(\tilde{A}_{n_A})$$
$$+ \sum_{n_B=1}^{N_B} g(\tilde{B}_{n_B}) + \sum_{n_C=1}^{N_C} g(\tilde{C}_{n_C})$$
subject to
$$A_{n_A} - \tilde{A}_{n_A} = 0, \quad n_A = 1, \ldots, N_A,$$
$$B_{n_B} - \tilde{B}_{n_B} = 0, \quad n_B = 1, \ldots, N_B,$$
$$C_{n_C} - \tilde{C}_{n_C} = 0, \quad n_C = 1, \ldots, N_C.$$

(15)

If we introduce the scaled dual variables $U_A = [U_{A,1}^T \cdots U_{A,N_A}^T]^T$, with $U_{A,n_A} \in \mathbb{R}^{I_{n_A} \times F}$, for $n_A = 1, \ldots, N_A$, $U_B = [U_{B,1}^T \cdots U_{B,N_B}^T]^T$, with $U_{B,n_B} \in \mathbb{R}^{J_{n_B} \times F}$, for $n_B = 1, \ldots, N_B$, and $U_C = [U_{C,1}^T \cdots U_{C,N_C}^T]^T$, with $U_{C,n_C} \in \mathbb{R}^{K_{n_C} \times F}$, for $n_C = 1, \ldots, N_C$, the augmented Lagrangian is written as

$$L_\rho(A, B, C, \tilde{A}, \tilde{B}, \tilde{C}, U_A, U_B, U_C) = f_X(A, B, C) + \sum_{n_A=1}^{N_A} g(\tilde{A}_{n_A}) + \sum_{n_B=1}^{N_B} g(\tilde{B}_{n_B}) + \sum_{n_C=1}^{N_C} g(\tilde{C}_{n_C})$$
$$+ \frac{\rho}{2} \sum_{n_A=1}^{N_A} \left( \|A_{n_A} - \tilde{A}_{n_A} + U_{A,n_A}\|_F^2 - \|U_{A,n_A}\|_F^2 \right)$$
$$+ \frac{\rho}{2} \sum_{n_B=1}^{N_B} \left( \|B_{n_B} - \tilde{B}_{n_B} + U_{B,n_B}\|_F^2 - \|U_{B,n_B}\|_F^2 \right)$$
$$+ \frac{\rho}{2} \sum_{n_C=1}^{N_C} \left( \|C_{n_C} - \tilde{C}_{n_C} + U_{C,n_C}\|_F^2 - \|U_{C,n_C}\|_F^2 \right).$$

(16)
The ADMoM for this problem is:

\[(A^{k+1}, B^{k+1}, C^{k+1}) = \arg\min_{A,B,C} (f_X(A, B, C) + \rho_2 N^A \sum_{n=1}^{N_A} ||A_{nA} - \tilde{A}_{nA}^{k+1} + U_{A nA}^k||_F^2 + \rho_2 N^B \sum_{n=1}^{N_B} ||B_{nB} - \tilde{B}_{nB}^{k+1} + U_{B nB}^k||_F^2 + \rho_2 N^C \sum_{n=1}^{N_C} ||C_{nC} - \tilde{C}_{nC}^{k+1} + U_{C nC}^k||_F^2)\] (17)

\[\tilde{A}_{nA}^{k+1} = (A_{nA}^{k+1} + U_{A nA}^k) + , n_A = 1, \ldots, N_A,\]

\[\tilde{B}_{nB}^{k+1} = (B_{nB}^{k+1} + U_{B nB}^k) + , n_B = 1, \ldots, N_B,\]

\[\tilde{C}_{nC}^{k+1} = (C_{nC}^{k+1} + U_{C nC}^k) + , n_C = 1, \ldots, N_C\]

\[U_{A nA}^{k+1} = U_{A nA}^k + A_{nA}^{k+1} - \tilde{A}_{nA}^{k+1} + , n_A = 1, \ldots, N_A,\]

\[U_{B nB}^{k+1} = U_{B nB}^k + B_{nB}^{k+1} - \tilde{B}_{nB}^{k+1} + , n_B = 1, \ldots, N_B,\]

\[U_{C nC}^{k+1} = U_{C nC}^k + C_{nC}^{k+1} - \tilde{C}_{nC}^{k+1} + , n_C = 1, \ldots, N_C.\]
The minimization problem in the first line of (17) is NP-hard. Based on (14), we propose the alternating optimization scheme

\[ A_{n_A}^{k+1} = \arg\min_{A_{n_A}} \left( \sum_{n_C=1}^{N_C} \frac{1}{2} \| X_{n_A,n_C}^{(1)} - A_{n_A} (C_{n_C}^k \odot B^k)^T \|_F^2 + \frac{\rho}{2} \| A_{n_A} - \tilde{A}_{n_A}^k + U_{A_{n_A}}^k \|_F^2 \right) \]

\[ = \left( \sum_{n_C=1}^{N_C} X_{n_A,n_C}^{(1)} (C_{n_C}^k \odot B^k) \right) + \rho \left( \tilde{A}_{n_A}^k - U_{A_{n_A}}^k \right) \]

\[ B_{n_B}^{k+1} = \arg\min_{B_{n_B}} \left( \sum_{n_C=1}^{N_C} \frac{1}{2} \| X_{n_B,n_C}^{(2)} - B_{n_B} (C_{n_C}^k \odot A_{k+1}^k)^T \|_F^2 + \frac{\rho}{2} \| B_{n_B} - \tilde{B}_{n_B}^k + U_{B_{n_B}}^k \|_F^2 \right) \]

\[ = \left( \sum_{n_C=1}^{N_C} X_{n_B,n_C}^{(2)} (C_{n_C}^k \odot A_{k+1}^k) \right) + \rho \left( \tilde{B}_{n_B}^k - U_{B_{n_B}}^k \right) \]

(18)

\[ C_{n_C}^{k+1} = \arg\min_{C_{n_C}} \left( \sum_{n_B=1}^{N_B} \frac{1}{2} \| X_{n_C,n_B}^{(3)} - C_{n_C} (B_{n_B}^{k+1} \odot A_{k+1}^k)^T \|_F^2 + \frac{\rho}{2} \| C_{n_C} - \tilde{C}_{n_C}^k + U_{C_{n_C}}^k \|_F^2 \right) \]

\[ = \left( \sum_{n_B=1}^{N_B} X_{n_C,n_B}^{(3)} (B_{n_B}^{k+1} \odot A_{k+1}^k) \right) + \rho \left( \tilde{C}_{n_C}^k - U_{C_{n_C}}^k \right) \]

\[ \left( \sum_{n_B=1}^{N_B} (B_{n_B}^{k+1} \odot A_{k+1}^k)^T (B_{n_B}^{k+1} \odot A_{k+1}^k) \right) + \rho I \right)^{-1} \]

, for \( n_B = 1, \ldots, N_B \),

\[ C_{n_C}^{k+1} = \arg\min_{C_{n_C}} \left( \sum_{n_B=1}^{N_B} \frac{1}{2} \| X_{n_C,n_B}^{(3)} - C_{n_C} (B_{n_B}^{k+1} \odot A_{k+1}^k)^T \|_F^2 + \frac{\rho}{2} \| C_{n_C} - \tilde{C}_{n_C}^k + U_{C_{n_C}}^k \|_F^2 \right) \]

\[ = \left( \sum_{n_B=1}^{N_B} X_{n_C,n_B}^{(3)} (B_{n_B}^{k+1} \odot A_{k+1}^k) \right) + \rho \left( \tilde{C}_{n_C}^k - U_{C_{n_C}}^k \right) \]

\[ \left( \sum_{n_B=1}^{N_B} (B_{n_B}^{k+1} \odot A_{k+1}^k)^T (B_{n_B}^{k+1} \odot A_{k+1}^k) \right) + \rho I \right)^{-1} \]

, for \( n_C = 1, \ldots, N_C \).

Assuming that we have computed the state of the algorithm at time \( k \), the \((k + 1)\)-st iteration of the ADMoM for large NTF is given as Algorithm 2, where \( n_A = 1, \ldots, N_C \), \( n_B = 1, \ldots, N_B \), and \( n_C = 1, \ldots, N_C \).

We observe that, during each ADMoM iteration, we avoid the solution of constrained optimization problems. Furthermore, and, perhaps, 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 implemented in parallel. After the computation of \( A_{k+1}^k \), the updates of \( B_{n_B}^k \), for \( n_B = 1, \ldots, N_B \), are also independent and can be implemented in parallel as well. Finally, after the computation of \( B_{k+1}^k \), we can compute in parallel the updates of \( C_{n_C}^k \), for \( n_C = 1, \ldots, N_C \).

We note that we can solve problem (15) using the ‘centralized’ ADMoM of Section IV. The difference is that, using the distributed ADMoM developed in this section, we uncover the inherent parallelism...
in the updates of the blocks of \( \mathbf{A}^k \), \( \mathbf{B}^k \), and \( \mathbf{C}^k \). If we initialize the corresponding quantities of the two algorithms with the same values, then both algorithms evolve in exactly the same way. Thus, the study (for example, convergence analysis and/or numerical behavior) of one of them is sufficient for the characterization of both.

C. A parallel implementation of the ADMoM for very large NTFs

In this subsection, we describe an implementation of the ADMoM for very large NTFs on a mesh-type architecture. In order to keep the presentation simple, we assume that \( N_A = N_B = N_C = N \). We assume that the matrix unfoldings \( \mathbf{X}^{(1)} \), \( \mathbf{X}^{(2)} \), and \( \mathbf{X}^{(3)} \) have been split into \( N^2 \) blocks of appropriate dimensions, with the \((i,j)\)-th block stored at the \((i,j)\)-th processing element, for \( i, j = 1, \ldots, N \) (see Figure 1).

Each ADMoM iteration consists of three stages. During the first stage, we use the blocks of \( \mathbf{X}^{(1)} \), as well as quantities from the previous ADMoM iteration, and compute blocks \( \mathbf{A}^{k+1} \), for \( n = 1, \ldots, N \). The first stage ends with the computation of \( \tilde{\mathbf{A}}^{k+1} \) and \( \mathbf{U}^{k+1} \), for \( n = 1, \ldots, N \). During the second stage, we use the blocks of \( \mathbf{X}^{(2)} \), as well as quantities from the previous ADMoM iteration and the
Fig. 2. Computation of $A_{n}^{k+1}$, for $n = 1, \ldots, N$.

newly computed $A^{k+1}$, and compute $B_{n}^{k+1}$, $B_{n}^{k+1}$, and $U_{B_{n}}^{k+1}$, for $n = 1, \ldots, N$. Finally, during the third stage, we use the blocks of $X^{(3)}$, quantities from the previous ADMoM iteration as well as $A^{k+1}$ and $B^{k+1}$, and compute $C_{n}^{k+1}$, $\tilde{C}_{n}^{k+1}$, and $U_{C_{n}}^{k+1}$, for $n = 1, \ldots, N$. 

September 9, 2014 DRAFT
In Figure 2, we depict the data flow for the computation of $A^{k+1}$. Each processing element contains the appropriate block of $X^{(1)}$. The inputs to the $N$ top processing elements are the $C_n^k$, 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 $\sum_{l=1}^{j} X_{n,l}^{(1)} (C_l^k \odot B^k)$ and $\sum_{l=1}^{j} (C_l^k \odot B^k)^T (C_l^k \odot B^k)$ (left-right communication), of dimensions $\frac{1}{N} \times F$ and $F \times F$, respectively, which are necessary for the computation of $A_n^{k+1}$. The computation of $A_n^{k+1}$, for $n = 1, \ldots, N$, amounts to the solution of $\rho$ systems of linear equations with common coefficient matrix and takes place at the right-side of the structure. Variables $\tilde{A}_n^k$ and $U_{A_n}^k$, for $n = 1, \ldots, N$, are used for the computation of $A_n^{k+1}$; then, they are updated locally.

After the computation of the blocks of $A^{k+1}$, we can compute the blocks of $B^{k+1}$ (see Figure 3) and then the blocks of $C^{k+1}$ (see Figure 4).

As we see in Figures 2, 3, in order to update the blocks of the $A^{k+1}$, $B^{k+1}$, and $C^{k+1}$, we use the appropriate blocks $B_n^k$ and $C_n^k$, for $n = 1, \ldots, N$, as well as the whole matrices $B^k$ and $A^{k+1}$. The communication of these matrices does not have prohibitive cost, especially in the cases where the rank $F$ is (relatively) small.
Fig. 4. Computation of $C_{n}^{k+1}$, for $n = 1, \ldots, N$.

Actual implementation of the distributed ADMoM for very large NTFs will depend on the specific parallel architecture and programming environment used. Architecture-specific customization and performance optimization can make a big difference (e.g., with respect to memory caching and communication patterns). Since the main aim and contribution of the present 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

In our numerical experiments, we use synthetic data and compare the ADMoM NTF with the NALS NTF, as implemented in the state-of-art nway toolbox for Matlab, by Rasmus Bro. More specifically, we use the parafac routine of the nway toolbox, with the nonnegativity option turned on. In our experience, this is the best available NTF algorithm for up to moderate-size NTF problems. parafac incorporates many improvements relative to plain NALS, including good initialization and computational improvements such as warm-started block-pivoting based fast nonnegative least squares, as opposed to slower nonnegative column updates for the latent factor matrices. Since an accurate statement about the computational complexity per iteration of parafac is not easy, the metric we use for both algorithms is...
the cputime, as computed by Matlab. Despite the fact that this metric depends on the actual algorithm implementation, parafac has been under development and refinement for almost 20 years, and we carefully coded our ADMoM-based algorithm. Hence, the comparison of cputime provides a rough but useful comparative complexity assessment of the algorithms.

In each experiment, we fix the additive noise variance, $\sigma^2_N$, and generate $N$ realizations of tensor $X$ as follows. We generate random matrices $A^o$, $B^o$, and $C^o$ with i.i.d. $\mathcal{U}[0,1]$ elements (with the rand command of MATLAB) and synthesize the noiseless tensor $X^o$ as the sum of outer products of the respective columns of $A^o$, $B^o$, and $C^o$. The noisy tensor $X$ is given by $X = (X^o + N)_+$, where $N$ consists of i.i.d. elements, distributed according to $\mathcal{N}(0, \sigma^2_N)$. For each realization, we solve the NTF problem using the NALS algorithm, as implemented via the parafac routine in the nway toolbox, with nonnegativity turned on for all three modes. Then, we solve the same NTF problem using the ADMoM.

We define the relative error between matrices $M$ and $N$ as

$$
E(M, N) = \frac{\|M - N\|_F}{\|M\|_F},
$$

and the relative factorization error as

$$
E(X, [A, B, C]) = \frac{\|X - [A, B, C]\|_F}{\|X\|_F}.
$$

We note that, at the $k$-th ADMoM iteration, $A^k$, $B^k$, and $C^k$ are not necessarily nonnegative. They become nonnegative (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 nonnegative.

The ADMoM is terminated at iteration $k$ if either $k$ is larger than the maximum number of ADMoM iterations, $n_{\text{max}}$, or the following conditions hold true

$$
\max \left( E(A^k, \tilde{A}^k), E(B^k, \tilde{B}^k), E(C^k, \tilde{C}^k) \right) \leq \epsilon_1,
$$

$$
\max \left( E(\tilde{A}^k, \tilde{A}^{k-1}), E(\tilde{B}^k, \tilde{B}^{k-1}), E(\tilde{C}^k, \tilde{C}^{k-1}) \right) \leq \epsilon_2,
$$

where $\epsilon_1$ and $\epsilon_2$ are small positive real numbers. Condition (21) guarantees that the primal residuals of the optimization problem (5) are sufficiently small. It can be shown [27] that condition (22) guarantees that the dual residuals of the optimization problem (5) are sufficiently small.

Upon convergence, the relative factorization errors $E \left( X, [A^k, B^k, C^k] \right)$, $E \left( X, \left[ (A^k)_+, (B^k)_+, (C^k)_+ \right] \right)$, and $E \left( \tilde{X}, \left[ \tilde{A}^k, \tilde{B}^k, \tilde{C}^k \right] \right)$ are very close to each other. Under ‘normal’ algorithm behaviour, we expect that the distances between these terms decrease with increasing $k$. However, this does not always happen. In fact, we have observed that if the distance between $E \left( X, [A^k, B^k, C^k] \right)$ and $E \left( X, \left[ (A^k)_+, (B^k)_+, (C^k)_+ \right] \right)$
increases, for a number of successive iterations, then algorithm convergence is usually very slow. In these
cases, we have found it useful to restart the ADMoM from a random initial point, that is, give nonnegative
random values to $B^k$ and $C^k$ and set the rest of the algorithm matrix variables to zero. We illustrate this
phenomenon in Fig. 5, where we plot two pairs of factorization relative errors for $I = J = K = 100,$
$F = 10,$ $\sigma^2_N = 0.01,$ $\epsilon_1 = 10^{-2.5},$ and $\epsilon_2 = 10^{-3}.$ With the blue lines, we plot $E(\mathbf{X}, [A^k, B^k, C^k])$
while with the green lines we plot $E(\mathbf{X}, [(A^k)_, (B^k)_, (C^k)_)],)$ versus the iteration number $k.$ With
the solid lines, we plot the relative factorization errors corresponding to a realization with ‘abnormal’
behavior while with the dashed lines we plot the relative factorization errors corresponding to the same
realization with random restart. Looking at the solid lines, we observe that, at iteration $k = 6,$ the two
relative factorization errors start deviating, with the deviation persisting for a large number of successive
iterations, and then approach each other very slowly. As a result, the ADMoM converges very slowly; in
fact, in this case, it did not converge after $n_{\text{max}} = 250$ iterations. Looking at the dashed lines, we observe
that, after restarting the ADMoM, after $N_a = 15$ successive relative factorization error deviation steps,
the two relative factorization errors approach each other fast; in this case, the algorithm converged after
$k = 209$ iterations. It is not always true that one algorithm restart is sufficient for algorithm convergence
(after a ‘reasonable’ number of iterations). We have observed that the probability of ‘abnormal’ ADMoM
behavior depends on the scenario; usually, it increases with increasing $F.$

In Table I we present the mean cpu time for the NALS, NALS$_{\text{cpu}},$ and the ADMoM, ADMoM$_{\text{cpu}}.$ We also present the corresponding mean relative factorization errors, NALS$_{\text{re}}$ and ADMoM$_{\text{re}},$ for various values
of the tensor dimensions, $I, J, K,$ ranks $F,$ and noise variances $\sigma^2_N.$ More specifically, we consider cases
where

1) one, two, or three tensor dimensions are ‘large’, with respect to the others;
2) the rank $F$ is ‘small’ or ‘large’;
3) the additive noise is ‘weak’ or ‘strong’.

The number of Monte-Carlo realizations (over which we take averages) is $N = 100,$ the penalty parameter
is $\rho = 1,$ the maximum number of ADMoM iterations is $n_{\text{max}} = 250,$ and the number of successive
relative factorization error deviations for algorithm restart is $N_a = 25.$ Especially in the cases where
‘abnormal’ algorithm behavior is more likely, we found it useful to replace (21) with the following
condition:

$$\frac{E(\mathbf{X}, [(A^k)_, (B^k)_, (C^k)_+])}{E(\mathbf{X}, [A^k, B^k, C^k])} \leq 1 + \epsilon_3,$$

(23)
where $\epsilon_3$ is a small positive real number.\footnote{In our experiments, we used $\epsilon_2 = 10^{-2}$ and $\epsilon_3 = 5 \times 10^{-3}$.}

Our observations are as follows:

1) There is no clear winner, in terms of \texttt{cpu time}. In some cases, the NALS is faster, while in other cases the ADMoM is faster. In the cases with ‘large’ $F$, the NALS is usually (much) faster than the ADMoM, while in the cases with ‘small’ $F$, the ADMoM is usually more efficient than the NALS.

2) Both algorithms attain approximately the same factorization accuracy. The cases where there exists a noticeable difference between the two mean relative factorization errors are those in which the ADMoM did not fully converge after $n_{\max}$ iterations. For example, in the seventh line of Table II (that is, for $I = J = 300$, $K = 20$, $F = 15$, and $\sigma_N^2 = 10^{-2}$), the difference in the mean relative factorization errors is caused by the fact that the ADMoM, after one or more restarts, did not fully converge in three out of 100 realizations.

3) In the ‘weak noise’ cases, the ADMoM initialization of subsection IV-B is effective. This was expected, since, in these cases, the initial eigen-decomposition based estimates are accurate.

\footnote{With the appropriate tuning of parameters $\epsilon_1$ and $\epsilon_3$, the corresponding terminating criteria lead to ‘similar’ algorithm behaviour. However, since we found it easier to tune $\epsilon_3$, in this work we use (23) instead of (21).}
TABLE I
PERFORMANCE MEASURES OF ADMoM AND NALS NTF.

| $I$  | $J$  | $K$ | $F$ | $\sigma_N^2$ | NALS cpu | ADMoM cpu | NALS re | ADMoM re |
|------|------|-----|-----|--------------|----------|-----------|---------|----------|
| 1000 | 30   | 30  | 3   | $10^{-2}$    | 8.5371   | 4.1979    | 0.2100  | 0.2108   |
| 1000 | 30   | 30  | 3   | $10^{-4}$    | 11.4243  | 2.0268    | 0.0218  | 0.0220   |
| 1000 | 30   | 30  | 15  | $10^{-2}$    | 65.7790  | 52.98     | 0.0505  | 0.0523   |
| 1000 | 20   | 20  | 15  | $10^{-4}$    | 84.7642  | 46.7645   | 0.0051  | 0.0053   |
| 300  | 300  | 20  | 3   | $10^{-2}$    | 9.4612   | 2.5945    | 0.2113  | 0.2117   |
| 300  | 300  | 20  | 3   | $10^{-4}$    | 13.1442  | 2.9707    | 0.0221  | 0.221    |
| 300  | 300  | 20  | 15  | $10^{-2}$    | 46.6018  | 69.3164   | 0.0509  | 0.539    |
| 300  | 300  | 20  | 15  | $10^{-4}$    | 62.2833  | 45.2131   | 0.0051  | 0.0079   |
| 100  | 100  | 100 | 3   | $10^{-2}$    | 4.2589   | 3.5961    | 0.2098  | 0.2104   |
| 100  | 100  | 100 | 3   | $10^{-4}$    | 5.6161   | 2.4235    | 0.0223  | 0.0224   |
| 100  | 100  | 100 | 15  | $10^{-2}$    | 20.88210 | 109.1450  | 0.0508  | 0.0509   |
| 100  | 100  | 100 | 15  | $10^{-4}$    | 26.6969  | 89.1963   | 0.0510  | 0.0610   |

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

As we mentioned, if the ‘centralized’ and the ‘distributed’ ADMoM algorithms start from the same initial point, then they evolve in exactly the same way. Thus, their convergence properties are exactly the same.

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 NTF as an example to work out the main ideas, but our approach can be generalized to many other types of constraints on the latent factors, as well as other tensor factorizations. The main advantages of our approach are that it bypasses the need to solve constrained optimization problems in each iteration, thus significantly reducing per-iteration complexity; and that it is naturally amenable to parallel implementation on regular high-performance computing (e.g., mesh) architectures. Our numerical experiments were very encouraging, indicating that the ADMoM-based NTF has high...
potential as an alternative to state-of-the-art approaches. Customization and performance optimization for specific computing architectures and parallel programming interfaces are topics of on-going work that will be reported elsewhere.

REFERENCES

[1] A. P. Liavas and N. D. Sidiropoulos, “Parallel algorithms for large-scale constrained tensor decomposition,” in Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, 2015 (to be submitted).
[2] T. Kolda and B. Bader, “Tensor decompositions and applications,” SIAM REVIEW, vol. 51, no. 3, pp. 455–500, 2009.
[3] A. Smilde, R. Bro, P. Geladi, and J. Wiley, Multi-way analysis with applications in the chemical sciences. Wiley, 2004.
[4] P. Kroonenberg, Applied multiway data analysis. Wiley, 2008.
[5] N. Sidiropoulos, R. Bro, and G. Giannakis, “Parallel factor analysis in sensor array processing,” IEEE Transactions on Signal Processing, vol. 48, no. 8, pp. 2377–2388, 2000.
[6] N. Sidiropoulos, G. Giannakis, and R. Bro, “Blind PARAFAC receivers for DS-CDMA systems,” IEEE Transactions on Signal Processing, vol. 48, no. 3, pp. 810–823, 2000.
[7] D. Nion, K. Mokios, N. Sidiropoulos, and A. Potamianos, “Batch and adaptive PARAFAC-based blind separation of convolutive speech mixtures,” IEEE Transactions on Audio, Speech, and Language Processing, vol. 18, no. 6, pp. 1193–1207, 2010.
[8] C. Fevotte and A. Ozerov, “Notes on nonnegative tensor factorization of the spectrogram for audio source separation: Statistical insights and towards self-clustering of the spatial cues,” in Exploring Music Contents, ser. Lecture Notes in Computer Science, S. Ystad, M. Aramaki, R. Kronland-Martinet, and K. Jensen, Eds. Springer Berlin, 2011, vol. 6684, pp. 102–115.
[9] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, “Parcube: Sparse parallelizable tensor decompositions.” in ECML/PKDD (1), ser. Lecture Notes in Computer Science, P. A. Flach, T. D. Bie, and N. Cristianini, Eds., vol. 7523. Springer, 2012, pp. 521–536.
[10] R. Bro and N. Sidiropoulos, “Least squares regression under unimodality and non-negativity constraints,” Journal of Chemometrics, vol. 12, pp. 223–247, 1998.
[11] A. Cichocki, D. Mandic, C. Caiafa, A.-H. Phan, G. Zhou, Q. Zhao, and L. De Lathauwer, “Multiway Component Analysis: Tensor Decompositions for Signal Processing Applications,” IEEE Signal Processing Magazine, 2014 (to appear).
[12] J. Kruskal, “Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics,” Linear Algebra and its Applications, vol. 18, no. 2, pp. 95–138, 1977.
[13] L. Chiantini and G. Ottaviani, “On generic identifiability of 3-tensors of small rank,” SIAM. J. Matrix Anal. & Appl., vol. 33, no. 3, pp. 1018–1037, 2012.
[14] R. Harshman, “Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multimodal factor analysis,” UCLA Working Papers in Phonetics, vol. 16, pp. 1–84, 1970.
[15] ——, “Determination and proof of minimum uniqueness conditions for PARAFAC-1,” UCLA Working Papers in Phonetics, vol. 22, pp. 111–117, 1972.
[16] J. Carroll and J. Chang, “Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart-Young decomposition,” Psychometrika, vol. 35, no. 3, pp. 283–319, 1970.
[17] C. Hillar and L.-H. Lim, “Most Tensor Problems are NP-hard,” 2009. [Online]. Available: http://arxiv.org/abs/0911.1393
[18] Apache, “Hadoop.” [Online]. Available: http://hadoop.apache.org/

[19] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[20] U. Kang, E. Papalexakis, A. Harpole, and C. Faloutsos, “Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries,” in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012, pp. 316–324.

[21] N. Sidiropoulos, E. Papalexakis, and C. Faloutsos, “A Parallel Algorithm for Big Tensor Decomposition Using Randomly Compressed Cubes (PARACOMP),” in Proc. IEEE ICASSP 2014, May 4-9, Florence, Italy.

[22] ———, “Parallel Randomly Compressed Cubes: A Scalable Distributed Architecture for Big Tensor Decomposition,” IEEE Signal Processing Magazine, Sep. 2014.

[23] A. de Almeida and A. Kibangou, “Distributed computation of tensor decompositions in collaborative networks,” in Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2013 IEEE 5th International Workshop on, Dec 2013, pp. 232–235.

[24] ———, “Distributed large-scale tensor decomposition,” in Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, May 2014.

[25] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Found. Trends Mach. Learn., vol. 3, no. 1, pp. 1–122, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1561/2200000016

[26] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, “An Alternating Direction Algorithm for Matrix Competion with Nonnegative Factors,” Frontiers of Mathematics in China, vol. 51, no. 2, pp. 365–384, 2010.

[27] A. P. Liavas and N. D. Sidiropoulos, “Parallel Algorithms for Constrained Tensor Factorization via the Alternating Direction Method of Multipliers,” Technical Report, 2014.

[28] J. M. F. Ten Berge and J. N. Tendeiro, “The link between sufficient conditions by Harshman and by Kruskal for uniqueness in Candecomp/Parafac,” Journal of Chemometrics, vol. 23, no. 7-8, pp. 321–323, 2009. [Online]. Available: http://dx.doi.org/10.1002/cem.1204

[29] L. Xu, B. Yu, and Y. Zhang, “An Alternating Direction and Projection Algorithm for Structure-enforced Matrix Factorization,” 2013. [Online]. Available: http://www.caam.rice.edu/~yzhang/reports/tr1311.pdf