TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Large Scale Optimization Methods and Applications in Tensor Optimization

by

Georgios Lourakis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE MASTER OF SCIENCE OF

ELECTRICAL AND COMPUTER ENGINEERING

December 2017

THESIS COMMITTEE

Professor Athanasios P. Liavas, *Thesis Supervisor*
Professor Vassilis Digalakis
Associate Professor George N. Karystinos

Abstract

We consider the problems of nonnegative tensor factorization and completion. Our aim is to derive efficient algorithms that are also suitable for parallel implementation. We adopt the alternating optimization framework and solve each matrix nonnegative least-squares problem via a Nesterov-type algorithm for convex and strongly convex problems. We describe parallel implementations of the algorithms and measure the attained speedup in a multi-core computing environment. It turns out that the derived algorithms are competitive candidates for the solution of very large-scale nonnegative tensor factorization and completion.

Acknowledgements

I would like to thank my family for their support and encouragement. I would also like to thank my supervisor, Professor Athanasios Liavas, for his guidance throughout this work.

Table of Contents

List of Abbreviations

ADMM	Alternating Direction Method of Multipliers
AO	Alternating Optimization
AOO	All-at-Once Optimization
BSUM	Block Successive Bound Minimization
CANDECOMP	Canonical Decomposition
CPD	Canonical Polyadic Decomposition
i.i.d.	independent and identically distributed
KKT	Karush-Kuhn-Tucker
MNLS	Matrix Nonnegative Least-Squares
MPI	Message Passing Interface
MRFE	Maximum Relative Factor Error
MU	Multiplicative Updates
NTC	Nonnegative Tensor Completion
NTF	Nonnegative Tensor Factorization
PARAFAC	Parallel Factor Analysis
RFE	Relative Factorization Error

Chapter 1

Introduction

Tensors are mathematical objects that have recently gained great popularity due to their ability to model multiway data dependencies [1], [2], [3], [4]. Tensor factorization (or decomposition) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization, interpretation and completion. Tensor factorizations are usually computed as solutions of optimization problems [1], [2]. The Canonical Decomposition or Canonical Polyadic Decomposition (CANDECOMP or CPD), also known as Parallel Factor Analysis (PARAFAC), and the Tucker Decomposition are the two most widely used tensor factorization models. In this work, we focus on nonnegative PARAFAC, which, for simplicity, we call Nonnegative Tensor Factorization (NTF).

Alternating Optimization (AO), All-at-Once Optimization (AOO), and Multiplicative Updates (MUs) are among the most commonly used techniques for NTF [2], [5]. Recent work for constrained tensor factorization/completion includes, among others, [6], [7], [8], and [9].

In [6], several NTF algorithms and a detailed convergence analysis have been developed. A general framework for joint matrix/tensor factorization/completion has been developed in [7]. In [8], an Alternating Direction Method of Multipliers (ADMM) algorithm for NTF has been derived, and an architecture for its parallel implementation has been outlined. However, the convergence properties of the algorithm in ill-conditioned cases are not favorable, necessitating additional research towards their improvement. In [9], the authors consider constrained matrix/tensor factorization/completion problems. They adopt the AO framework as outer loop and use the ADMM for solving the inner constrained optimization problem for one matrix factor conditioned on the rest. The ADMM offers significant flexibility, due to its ability to efficiently handle a wide range of constraints.

In [10], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations on a multi-core system have been reported. Related work on parallel algorithms for sparse tensor decomposition includes [11] and [12].

1.1 Contribution

In this work, we focus on very large dense NTF problems and sparse NTC problems. Our aim is to derive efficient NTF and NTC algorithms, suitable for parallel implementation. We adopt the AO framework and solve each matrix nonnegative least-squares (MNLS) problem via a first-order optimal (Nesterov-type) algorithm for L-smooth μ-strongly con-

vex problems.[1] Then, we describe in detail MPI implementations of the AO NTF and AO NTC algorithms and measure the speedup attained in a multi-core environment. We conclude that the proposed algorithms are strong candidates for the solution of very large dense NTF and sparse NTC problems. The results concerning the NTF problems have appeared in [15] and [16].

1.2 Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital bold letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. $\mathbb{R}_+^{I\times J\times K}$ 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, $\mathbf{I}$ denotes the identity matrix of appropriate dimensions, and $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of element-wise nonnegative matrices. The outer product of vectors $\mathbf{a} \in \mathbb{R}^{I\times 1}$, $\mathbf{b} \in \mathbb{R}^{J\times 1}$, and $\mathbf{c} \in \mathbb{R}^{K\times 1}$ is the rank-one tensor $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I\times J\times K}$ with elements $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})(i, j, k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \otimes \mathbf{B}$. The Khatri-Rao (columnwise Kronecker) product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \odot \mathbf{B}$ and the Hadamard (elementwise) product is denoted as $\mathbf{A} \circledast \mathbf{B}$. Finally, inequality $\mathbf{A} \succeq \mathbf{B}$ means that matrix $\mathbf{A} - \mathbf{B}$ is positive semidefinite.

1.3 Structure

In Chapter 2, we present the Nesterov algorithm for set-constrained L-smooth μ-strongly convex optimization problems and derive a Nesterov-type algorithm for the MNLS problem with proximal term. In Chapter 3, we briefly describe the NTF problem, present the associated AO NTF algorithm and describe in detail a parallel implementation. In Chapter 4, we present the AO NTC algorithm and describe a parallel implementation. Finally, Chapter 5 concludes this thesis with some ideas for future work.

[1] We note that a closely related algorithm for the solution of MNLS problems has been used in [13] and [14]; we explain in detail later the performance improvement offered by our approach.

Chapter 2

The Matrix Nonnegative Least Squares problem

In this chapter, we present an optimal first-order algorithm for the solution of L-smooth μ-strongly convex MNLS problems. Optimal first-order methods have recently attracted great research interest because they are strong candidates and, in many cases, the only viable way for the solution of very large optimization problems.

2.1 Optimal first-order methods for L-smooth μ-strongly convex optimization problems

We consider optimization problems of smooth and strongly convex functions and briefly present results concerning their information complexity and the associated first-order optimal algorithms (for a detailed exposition see [17, Chapter 2]).

We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth (that is, differentiable up to a sufficiently high order) convex function, with gradient $\nabla f(\mathbf{x})$ and Hessian $\nabla^2 f(\mathbf{x})$. Our aim is to solve the problem

$$\min_{\mathbf{x}} f(\mathbf{x}), \tag{2.1}$$

within accuracy $\epsilon > 0$. The solution accuracy is defined as follows. If $f^* := \min_{\mathbf{x}} f(\mathbf{x})$, then point $\bar{\mathbf{x}} \in \mathbb{R}^n$ solves problem (2.1) within accuracy ϵ if $f(\bar{\mathbf{x}}) - f^* \leq \epsilon$.

Let $0 < \mu \leq L < \infty$. A smooth convex function f is called L-smooth or, using the notation of [17, p. 66], $f \in \mathcal{S}^{\infty,1}_{0,L}$, if

$$\mathbf{0} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall \mathbf{x} \in \mathbb{R}^n, \tag{2.2}$$

and L-smooth μ-strongly convex, or $f \in \mathcal{S}^{\infty,1}_{\mu,L}$, if

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{2.3}$$

The number of iterations that first-order methods need for the solution of problem (2.1), within accuracy ϵ, is $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ if $f \in \mathcal{S}^{\infty,1}_{0,L}$, and $O\left(\sqrt{\frac{L}{\mu}} \log \frac{1}{\epsilon}\right)$ if $f \in \mathcal{S}^{\infty,1}_{\mu,L}$ [17, Theorem 2.2.2]. The convergence rate in the first case is *sublinear* while, in the second case, it is *linear* and determined by the condition number of the problem, $\mathcal{K} := \frac{L}{\mu}$. Thus, strong convexity is a very important property that should be exploited whenever possible.

An algorithm that achieves this complexity, and, thus, is first-order optimal, appears in Algorithm 1 (see, also [17, p. 80]). This algorithm can handle both the L-smooth case,

Algorithm 1: Nesterov algorithm for L-smooth μ-strongly convex optimization problems

Input: $\mathbf{x}_0 \in \mathbb{R}^n$, μ, L. Set $\mathbf{y}_0 = \mathbf{x}_0$, $\alpha_0 \in (0,1)$, $q = \frac{\mu}{L}$.

1 k-th iteration

2 $\quad \mathbf{x}_{k+1} = \mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)$

3 $\quad \alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$

4 $\quad \beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$

5 $\quad \mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k)$

by setting $q = 0$, and the L-smooth μ-strongly convex case, by setting $q = \frac{\mu}{L} > 0$.

If the problem of interest is the constrained problem

$$\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x}), \tag{2.4}$$

where $\mathbb{X}$ is a closed convex set, then the corresponding optimal algorithm is very much alike Algorithm 1, with the only difference being in the computation of $\mathbf{x}_{k+1}$. We now have that [17, p. 90]

$$\mathbf{x}_{k+1} = \Pi_{\mathbb{X}}\left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right), \tag{2.5}$$

where $\Pi_{\mathbb{X}}(\cdot)$ denotes the Euclidean projection onto set $\mathbb{X}$. The convergence properties of this algorithm are the same as those of Algorithm 1. If the projection onto set $\mathbb{X}$ is easy to compute, then the algorithm is both theoretically optimal and very efficient in practice.

2.2 Nesterov-type algorithm for MNLS with proximal term

In the sequel, we present a Nesterov-type algorithm for the MNLS problem with proximal term. Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{A} \in \mathbb{R}^{m\times r}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$, and consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2. \tag{2.6}$$

The gradient and Hessian of f, at point $\mathbf{A}$, are, respectively,

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} \tag{2.7}$$

and

$$\nabla^2 f(\mathbf{A}) := \frac{\partial^2 f(\mathbf{A})}{\partial \mathrm{vec}(\mathbf{A}) \partial \mathrm{vec}(\mathbf{A})^T} = \mathbf{B}^T\mathbf{B} \otimes \mathbf{I} \succeq \mathbf{0}. \tag{2.8}$$

Let $L := \max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$ and $\mu := \min(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$. If $\mu = 0$ (for example, if $r > n$), then problem (2.6) is L-smooth. If $\mu > 0$, then problem (2.6) is L-smooth μ-strongly convex. A first-order optimal algorithm for the solution of (2.6) can be derived using the approach of Section 2.1. We note that [13] and [14] solved problem (2.6) using a variation of Algorithm 1, which is equivalent to Algorithm 1 with $\mu = 0$. However, if $\mu > 0$, then this algorithm is *not* first-order optimal and, as we shall see later, it performs much worse

Algorithm 2: Nesterov-type algorithm for MNLS with proximal term

Input: $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$, $\mathbf{A}_* \in \mathbb{R}^{m\times r}$
$L = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu = \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$
$\lambda = g(L, \mu)$
$\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
$q = \frac{\mu+\lambda}{L+\lambda}$
$\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
$\alpha_0 = 1$, $k = 0$
while (1) **do**
  $\nabla f_{\mathtt{P}}(\mathbf{Y}_k) = \mathbf{W} + \mathbf{Y}_k\mathbf{Z}$
  if (terminating_condition is TRUE) **then**
    break
  else
    $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda}\nabla f_{\mathtt{P}}(\mathbf{Y}_k)\right)_+$
    $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
    $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
    $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta_{k+1}(\mathbf{A}_{k+1} - \mathbf{A}_k)$
    $k = k + 1$
return $\mathbf{A}_k$.

than the optimal.

We note that the values of L and μ are necessary for the development of the Nesterov-type algorithm, thus, their computation is imperative.[1]

In order to avoid very ill-conditioned problems (and guarantee strong convexity), we introduce a proximal term (as we will see in Section 3.1, under the AO framework) and solve problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f_{\mathtt{P}}(\mathbf{A}) := \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2 + \frac{\lambda}{2}\|\mathbf{A} - \mathbf{A}_*\|_F^2, \tag{2.9}$$

for given $\mathbf{A}_*$ and appropriately chosen λ. We choose λ based on L and μ, and denote this functional dependence as $\lambda = g(L, \mu)$. If $\frac{\mu}{L} \ll 1$, then we may set $\lambda \approx 10\mu$, significantly improving the conditioning of the problem by putting large weight on the proximal term; however, in this case, we expect that the optimal point will be biased towards $\mathbf{A}_*$. Otherwise, we may set $\lambda \lessapprox \mu$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible.

The gradient of $f_{\mathtt{P}}$, at point $\mathbf{A}$, is

$$\nabla f_{\mathtt{P}}(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A} - \mathbf{A}_*). \tag{2.10}$$

The Karush-Kuhn-Tucker (KKT) conditions for problem (2.9) are [13]

$$\nabla f_{\mathtt{P}}(\mathbf{A}) \geq \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \nabla f_{\mathtt{P}}(\mathbf{A}) \circledast \mathbf{A} = \mathbf{0}. \tag{2.11}$$

[1] An alternative to their direct computation is to estimate L using line-search techniques and overcome the computation of μ using heuristic adaptive restart techniques [18]. However, in our case, this alternative is computationally demanding, especially for large-scale problems, and shall not be considered.

These expressions can be used in a terminating condition. For example, we may terminate the algorithm if

$$\min_{i,j}\left([\nabla f_{\mathrm{P}}(\mathbf{A})]_{i,j}\right) > -\delta_1, \quad \max_{i,j}\left(\left|[\nabla f_{\mathrm{P}}(\mathbf{A}) \circledast \mathbf{A}]_{i,j}\right|\right) < \delta_2, \tag{2.12}$$

for small positive real numbers δ_1 and δ_2. Of course, other criteria, based, for example, on the (relative) change of the cost function can be used in terminating conditions.

A Nesterov-type algorithm for the solution of the MNLS problem with proximal term (2.9) is given in Algorithm 2. For notational convenience, we denote Algorithm 2 as

$$\mathbf{A}_{\mathrm{opt}} = \mathrm{Nesterov\_MNLS}(\mathbf{X}, \mathbf{B}, \mathbf{A}_*).$$

Computational complexity of Algorithm 2

Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mnr)$ and $O(rn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(r^3)$ operations. $\nabla f_{\mathrm{P}}(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration with cost $O(mr^2)$, $O(mr)$, and $O(mr)$ arithmetic operations, respectively.

Chapter 3

Nonnegative Tensor Factorization

3.1 Problem Formulation

Let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}_+^{I\times J\times K}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{A}^o, \mathbf{B}^o, \mathbf{C}^o]\!] = \sum_{r=1}^{R} \mathbf{a}_r^o \circ \mathbf{b}_r^o \circ \mathbf{c}_r^o, \tag{3.1}$$

where $\mathbf{A}^o = [\mathbf{a}_1^o \ \cdots \ \mathbf{a}_R^o] \in \mathbb{R}_+^{I\times R}$, $\mathbf{B}^o = [\mathbf{b}_1^o \ \cdots \ \mathbf{b}_R^o] \in \mathbb{R}_+^{J\times R}$, and $\mathbf{C}^o = [\mathbf{c}_1^o \ \cdots \ \mathbf{c}_R^o] \in \mathbb{R}_+^{K\times R}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Estimates of $\mathbf{A}^o$, $\mathbf{B}^o$, and $\mathbf{C}^o$ can be obtained by computing matrices $\mathbf{A} \in \mathbb{R}_+^{I\times R}$, $\mathbf{B} \in \mathbb{R}_+^{J\times R}$, and $\mathbf{C} \in \mathbb{R}_+^{K\times R}$ that solve the optimization problem

$$\min_{\mathbf{A}\geq\mathbf{0},\mathbf{B}\geq\mathbf{0},\mathbf{C}\geq\mathbf{0}} \quad f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}), \tag{3.2}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization and the inequalities are element-wise. A common choice for $f_{\boldsymbol{\mathcal{X}}}$ is

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \right\|_F^2 . \tag{3.3}$$

If $\boldsymbol{\mathcal{Y}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, then its matrix unfoldings, with respect to the first, second, and third mode, are given by [3]

$$\mathbf{Y}_{\mathbf{A}} = \mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^T, \ \mathbf{Y}_{\mathbf{B}} = \mathbf{B}\,(\mathbf{C} \odot \mathbf{A})^T, \ \mathbf{Y}_{\mathbf{C}} = \mathbf{C}\,(\mathbf{B} \odot \mathbf{A})^T.$$

Thus, $f_{\boldsymbol{\mathcal{X}}}$ can be expressed as

$$\begin{aligned} f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}) &= \frac{1}{2} \left\| \mathbf{X}_{\mathbf{A}} - \mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^T \right\|_F^2 \\ &= \frac{1}{2} \left\| \mathbf{X}_{\mathbf{B}} - \mathbf{B}\,(\mathbf{C} \odot \mathbf{A})^T \right\|_F^2 \\ &= \frac{1}{2} \left\| \mathbf{X}_{\mathbf{C}} - \mathbf{C}\,(\mathbf{B} \odot \mathbf{A})^T \right\|_F^2 . \end{aligned} \tag{3.4}$$

These expressions form the basis for the AO NTF in the sense that, if we fix two matrix factors, we can update the third by solving an MNLS problem. For reasons related with the conditioning of the MNLS problems, we propose to add a proximal term. More specifically, if $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$ are the estimates of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively, after the k-th AO

iteration, then $\mathbf{A}_{k+1}$ is computed as

$$\mathbf{A}_{k+1} := \underset{\mathbf{A}\geq\mathbf{0}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{X}_{\mathbf{A}} - \mathbf{A}(\mathbf{C}_k \odot \mathbf{B}_k)^T \right\|_F^2 + \frac{\lambda_k^{\mathbf{A}}}{2} \|\mathbf{A} - \mathbf{A}_k\|_F^2, \tag{3.5}$$

where $\lambda_k^{\mathbf{A}} \geq 0$ determines the weight assigned to the proximal term. If $(\mathbf{C}_k \odot \mathbf{B}_k)$ is a well-conditioned matrix, then it is reasonable to put small weight on the proximal term and compute $\mathbf{A}_{k+1}$ that leads to a large decrease of the cost function $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}_k, \mathbf{C}_k)$. If, on the other hand, $(\mathbf{C}_k \odot \mathbf{B}_k)$ is an ill-conditioned matrix, then it is reasonable to put large weight on the proximal term, leading to a better conditioned problem and easy computation of $\mathbf{A}_{k+1}$ that improves the fit in $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}_k, \mathbf{C}_k)$ but is not very far from $\mathbf{A}_k$. This is the strategy we shall follow for the solution of problem (3.2) (see also [6], [19]).

The computational efficiency of the AO NTF heavily depends on the algorithm we use for the solution of problem (3.5). In this work, we adopt the approach of Nesterov for the solution of L-smooth μ-strongly convex problems. The derived algorithm is optimal under the (worst-case) black-box first-order oracle framework [17, Chapter 2] and is very efficient in practice. Furthermore, it leads to an AO NTF algorithm that is suitable for parallel implementation.

3.2 Nesterov Based AO NTF

In Algorithm 3, we present the Nesterov-based AO NTF. We start from point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, MNLS problems with proximal terms, based on the previous estimates.

Algorithm 3: Nesterov-based AO NTF

Input: $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0 > \mathbf{0}$, $\mathbf{B}_0 > \mathbf{0}$, $\mathbf{C}_0 > \mathbf{0}$.

1 Set $k = 0$
2 **while** (1) **do**
3 $\quad \mathbf{A}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_{\mathbf{A}}, (\mathbf{C}_k \odot \mathbf{B}_k), \mathbf{A}_k)$
4 $\quad \mathbf{B}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_{\mathbf{B}}, (\mathbf{C}_k \odot \mathbf{A}_{k+1}), \mathbf{B}_k)$
5 $\quad \mathbf{C}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{X}_{\mathbf{C}}, (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}), \mathbf{C}_k)$
6 $\quad (\mathbf{A}_{k+1}^{\mathcal{N}}, \mathbf{B}_{k+1}^{\mathcal{N}}, \mathbf{C}_{k+1}^{\mathcal{N}}) = \text{Normalize}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$
7 $\quad$ **if** (terminating_condition is TRUE) **then** break; **endif**
8 $\quad (\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1}) = \text{Accelerate}(\mathbf{A}_{k+1}^{\mathcal{N}}, \mathbf{A}_k^{\mathcal{N}}, \mathbf{B}_{k+1}^{\mathcal{N}}, \mathbf{B}_k^{\mathcal{N}}, \mathbf{C}_{k+1}^{\mathcal{N}}, \mathbf{C}_k^{\mathcal{N}})$
9 $\quad k = k + 1$
10 **return** $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

For later use, we note that the most demanding computations during the update of factor matrix $\mathbf{A}_k$ via the Nesterov-type MNLS algorithm are (see line 3 of Algorithm 2)

$$\begin{aligned} \widetilde{\mathbf{W}}_{\mathbf{A}} &:= -\mathbf{X}_{\mathbf{A}} (\mathbf{C}_k \odot \mathbf{B}_k), \\ \widetilde{\mathbf{Z}}_{\mathbf{A}} &:= (\mathbf{C}_k \odot \mathbf{B}_k)^T (\mathbf{C}_k \odot \mathbf{B}_k) \\ &= (\mathbf{C}_k^T \mathbf{C}_k) \circledast (\mathbf{B}_k^T \mathbf{B}_k). \end{aligned} \tag{3.6}$$

Analogous quantities are computed for the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$.

After the updates of the factor matrices, we use two functions which have been proven very useful in our experiments, in the sense that they significantly reduce the number of outer iterations necessary to reach convergence.

Function "Normalize" normalizes each column of $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$ to unit Euclidean norm, putting all the power on the respective columns of $\mathbf{A}_{k+1}$. We denote its output as $\mathbf{A}_{k+1}^{\mathcal{N}}$, $\mathbf{B}_{k+1}^{\mathcal{N}}$ and $\mathbf{C}_{k+1}^{\mathcal{N}}$.

Function "Accelerate" implements an acceleration mechanism. The development of efficient acceleration mechanisms is a very important research topic, see, for example, [20], [21], but is beyond the scope of this thesis. In our experiments, we adopted the simple acceleration technique used in the function `parafac` of the n-way toolbox [22], which is briefly described as follows.

At iteration $k+1 > k_0$, after the computation and normalization of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$, we compute

$$\mathbf{A}_{\text{new}} = \mathbf{A}_k^{\mathcal{N}} + s_{k+1}(\mathbf{A}_{k+1}^{\mathcal{N}} - \mathbf{A}_k^{\mathcal{N}}), \tag{3.7}$$

where s_{k+1} is a small positive number; a simple choice for s_{k+1} is $s_{k+1} = (k+1)^{\frac{1}{n}}$, where n is initialized as $n = 3$ and its value may change as the algorithm progresses. In an analogous manner, we compute $\mathbf{B}_{\text{new}}$ and $\mathbf{C}_{\text{new}}$. If $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}_{\text{new}}, \mathbf{B}_{\text{new}}, \mathbf{C}_{\text{new}}) \leq f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$, then the acceleration step is successful, and we set $\mathbf{A}_{k+1} = \mathbf{A}_{\text{new}}$, $\mathbf{B}_{k+1} = \mathbf{B}_{\text{new}}$, and $\mathbf{C}_{k+1} = \mathbf{C}_{\text{new}}$. If the acceleration step fails, then it is ignored and we set $\mathbf{A}_{k+1} = \mathbf{A}_{k+1}^{\mathcal{N}}$, $\mathbf{B}_{k+1} = \mathbf{B}_{k+1}^{\mathcal{N}}$, and $\mathbf{C}_{k+1} = \mathbf{C}_{k+1}^{\mathcal{N}}$ as input to the next AO update. If the acceleration step fails for n_0 iterations, then we set $n = n + 1$, thus, decreasing the exponent of the acceleration step. Typical values of k_0 and n_0 are $k_0 = 5$ and $n_0 = 5$.

It has been shown in [19] that the AO NTF algorithm with proximal term falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (3.2).

We can use various termination criteria for the AO NTF algorithm based, for example, on the (relative) change of the cost function and/or the latent factors.

3.3 Parallel Implementation

In this section, we assume that we have at our disposal $p = p_{\mathbf{A}} \times p_{\mathbf{B}} \times p_{\mathbf{C}}$ processing elements and describe a parallel implementation of the Nesterov-based AO NTF algorithm, which has been motivated by the medium-grained approach of [11].[1] The p processors form a three-dimensional Cartesian grid and are denoted as $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, $i_{\mathbf{B}} = 1, \ldots, p_{\mathbf{B}}$, and $i_{\mathbf{C}} = 1, \ldots, p_{\mathbf{C}}$.

[1] We note that both the single-core and the multi-core implementations solve the same problem, thus problems that are identifiable in single-core environments remain identifiable in multi-core environments and the solutions, in both cases, are practically the same.

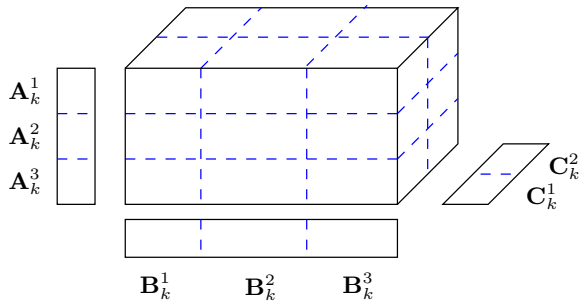

Figure 3.1: Tensor $\boldsymbol{\mathcal{X}}$, factors $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$, and their partitioning for $p_{\mathbf{A}} = p_{\mathbf{B}} = 3$ and $p_{\mathbf{C}} = 2$.

3.3.1 Variable partitioning and data allocation

In order to describe the parallel implementation, we introduce certain partitionings of the factor matrices and the tensor matricizations. We partition the factor matrix $\mathbf{A}_k$ into $p_{\mathbf{A}}$ block rows as

$$\mathbf{A}_k = \left[\begin{array}{ccc} \left(\mathbf{A}_k^1\right)^T & \cdots & \left(\mathbf{A}_k^{p_{\mathbf{A}}}\right)^T \end{array} \right]^T, \tag{3.8}$$

with $\mathbf{A}_k^{i_{\mathbf{A}}} \in \mathbb{R}^{\frac{I}{p_{\mathbf{A}}} \times R}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$. We partition accordingly the matricization $\mathbf{X}_{\mathbf{A}}$ and get

$$\mathbf{X}_{\mathbf{A}} = \left[\begin{array}{ccc} \left(\mathbf{X}_{\mathbf{A}}^1\right)^T & \cdots & \left(\mathbf{X}_{\mathbf{A}}^{p_{\mathbf{A}}}\right)^T \end{array} \right]^T, \tag{3.9}$$

with $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}} \in \mathbb{R}^{\frac{I}{p_{\mathbf{A}}} \times JK}$. In a similar manner, we partition $\mathbf{B}_k$ and $\mathbf{X}_{\mathbf{B}}$ into $p_{\mathbf{B}}$ block rows, each of size $\frac{J}{p_{\mathbf{B}}} \times R$ and $\frac{J}{p_{\mathbf{B}}} \times IK$, respectively, and $\mathbf{C}_k$ and $\mathbf{X}_{\mathbf{C}}$ into $p_{\mathbf{C}}$ block rows, each of size $\frac{K}{p_{\mathbf{C}}} \times R$ and $\frac{K}{p_{\mathbf{C}}} \times IJ$, respectively.

We partition tensor $\boldsymbol{\mathcal{X}}$ into p subtensors, according to the partitioning of the factor matrices (see Figure 3.1), and allocate its parts to the various processors, so that processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ receives subtensor $\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, defined as

$$\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}} := \boldsymbol{\mathcal{X}}\left((i_{\mathbf{A}}-1)\frac{I}{p_{\mathbf{A}}}+1 : i_{\mathbf{A}}\frac{I}{p_{\mathbf{A}}}, (i_{\mathbf{B}}-1)\frac{J}{p_{\mathbf{B}}}+1 : i_{\mathbf{B}}\frac{J}{p_{\mathbf{B}}}, (i_{\mathbf{C}}-1)\frac{K}{p_{\mathbf{C}}}+1 : i_{\mathbf{C}}\frac{K}{p_{\mathbf{C}}}\right). \tag{3.10}$$

We assume that, at the end of the k-th outer AO iteration,

(a) processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ knows $\mathbf{A}_k^{i_{\mathbf{A}}}$, $\mathbf{B}_k^{i_{\mathbf{B}}}$, and $\mathbf{C}_k^{i_{\mathbf{C}}}$;

(b) *all* processors know $\mathbf{A}_k^T\mathbf{A}_k$, $\mathbf{B}_k^T\mathbf{B}_k$, and $\mathbf{C}_k^T\mathbf{C}_k$.

3.3.2 Communication groups

We define certain communication groups, also known as communicators [23], over subsets of the p processors, which are used for the efficient collaborative implementation of specific computational tasks, as explained in detail later.

First, we define $p_{\mathbf{A}}$ two-dimensional processor groups, each involving the $p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors $p_{i_{\mathbf{A}},:,:}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$ (horizontal layers), with the $i_{\mathbf{A}}$-th processor group used for the collaborative update of $\mathbf{A}_k^{i_{\mathbf{A}}}$. Similarly, we define groups $p_{:,i_{\mathbf{B}},:}$, for $i_{\mathbf{B}} =$

$1, \ldots, p_{\mathbf{B}}$, and $p_{:,:,i_{\mathbf{C}}}$, for $i_{\mathbf{C}} = 1, \ldots, p_{\mathbf{C}}$, which are used for the collaborative update of $\mathbf{B}_k^{i_{\mathbf{B}}}$ and $\mathbf{C}_k^{i_{\mathbf{C}}}$, respectively.

We define $p_{\mathbf{B}} \times p_{\mathbf{C}}$ one-dimensional processor groups, each involving the $p_{\mathbf{A}}$ processors $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$. Each of these groups is used for the collaborative computation of $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$. Similarly, we define groups $p_{i_{\mathbf{A}},:,i_{\mathbf{C}}}$ and $p_{i_{\mathbf{A}},i_{\mathbf{B}},:}$, which are used for the collaborative computation of $\mathbf{B}_{k+1}^T \mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}^T \mathbf{C}_{k+1}$, respectively.

3.3.3 Factor update implementation

We describe in detail the update of $\mathbf{A}_k$, which is achieved via the parallel updates of $\mathbf{A}_k^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, and consists of the following stages:

1. Processors $p_{i_{\mathbf{A}},:,:}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, collaboratively compute the $\frac{I}{p_{\mathbf{A}}} \times R$ matrix

$$\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}} = -\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}} (\mathbf{C}_k \odot \mathbf{B}_k), \tag{3.11}$$

   and the result is scattered among the processors in the group; thus, each processor in the group receives $\frac{I}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}}$ successive rows of $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$. Term $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$ can be computed collaboratively because

$$\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}} (\mathbf{C}_k \odot \mathbf{B}_k) = \sum_{i_{\mathbf{B}}=1}^{p_{\mathbf{B}}} \sum_{i_{\mathbf{C}}=1}^{p_{\mathbf{C}}} \mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}} (\mathbf{C}_k^{i_{\mathbf{C}}} \odot \mathbf{B}_k^{i_{\mathbf{B}}}), \tag{3.12}$$

   where $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ is the matricization of $\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, with respect to the first mode. Processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ knows $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, $\mathbf{B}_k^{i_{\mathbf{B}}}$, and $\mathbf{C}_k^{i_{\mathbf{C}}}$, and computes the corresponding term of (3.12). The sum is computed and scattered among processors $p_{i_{\mathbf{A}},:,:}$ via a reduce-scatter operation.

2. Each processor in the group $p_{i_{\mathbf{A}},:,:}$ uses the scattered part of $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$, $\widetilde{\mathbf{Z}}_{\mathbf{A}} = \mathbf{C}_k^T \mathbf{C}_k \circledast \mathbf{B}_k^T \mathbf{B}_k$, and $\mathbf{A}_k^{i_{\mathbf{A}}}$, and computes the updated part of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, via the `while` loop of the Nesterov MNLS algorithm.

3. The updated parts of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ are all-gathered at the processors of the group $p_{i_{\mathbf{A}},:,:}$, so that *all* processors in the group learn the updated $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$.

4. By applying an all-reduce operation to $\left(\mathbf{A}_{k+1}^{i_{\mathbf{A}}}\right)^T \mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, on each of the single-dimensional processor groups $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$, for $i_{\mathbf{B}} = 1, \ldots, p_{\mathbf{B}}$ and $i_{\mathbf{C}} = 1, \ldots, p_{\mathbf{C}}$, *all* p processors learn $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$.[2]

The updates of $\mathbf{B}_k$ and $\mathbf{C}_k$ are implemented by following analogous steps.

The Euclidean norms of the columns of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$ appear on the diagonals of $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}^T \mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}^T \mathbf{C}_{k+1}$, which are known to all processors. Thus, no communication is necessary for the normalization of the updated matrix factors.

[2] In the cases where $R \gtrapprox \frac{I}{p_{\mathbf{A}}}$ it seems preferable to compute $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$ via an all-gather operation on terms $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, on each of the single-dimensional processor groups $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$. However, in this work, we mainly focus on small-rank factorizations, thus, in our communication cost analysis and experiments we do not present results for this alternative.

After the normalization step of the $(k+1)$-st AO iteration, processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ knows the parts of the normalized factors, that is, $\mathbf{A}_{k+1}^{i_{\mathbf{A}}\mathcal{N}}$, $\mathbf{B}_{k+1}^{i_{\mathbf{B}}\mathcal{N}}$, $\mathbf{C}_{k+1}^{i_{\mathbf{C}}\mathcal{N}}$, as well as $\mathbf{A}_{k}^{i_{\mathbf{A}}\mathcal{N}}$, $\mathbf{B}_{k}^{i_{\mathbf{B}}\mathcal{N}}$, and $\mathbf{C}_{k}^{i_{\mathbf{C}}\mathcal{N}}$, and can compute $\mathbf{A}_{\text{new}}^{i_{\mathbf{A}}}$, $\mathbf{B}_{\text{new}}^{i_{\mathbf{B}}}$, and $\mathbf{C}_{\text{new}}^{i_{\mathbf{C}}}$ (see (3.7)). The computation of the cost function $f_{\boldsymbol{\mathcal{X}}}$ at points $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$ and $(\mathbf{A}_{\text{new}}, \mathbf{B}_{\text{new}}, \mathbf{C}_{\text{new}})$ is implemented collaboratively. Each processing element computes its local contribution and, via an all-reduce operation over the whole processor grid, the values of the cost function are computed and become known to all processors, thus, all processors make the same decision regarding the success or failure of the acceleration step.

3.3.4 Communication cost

We focus on the parallel updates of $\mathbf{A}_{k}^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, and present results concerning the associated communication cost. Analogous results hold for the updates of $\mathbf{B}_{k}^{i_{\mathbf{B}}}$ and $\mathbf{C}_{k}^{i_{\mathbf{C}}}$.

We assume that an m-word message is transferred from one process to another with communication cost $t_s + t_w m$, where t_s is the latency, or startup time for the data transfer, and t_w is the word transfer time [23].

Communication occurs at three algorithm execution points.

1. The $\frac{I}{p_{\mathbf{A}}} \times R$ matrix $\widetilde{\mathbf{W}}_{\mathbf{A}}^{i_{\mathbf{A}}}$ is computed and scattered among the $p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors of group $p_{i_{\mathbf{A}},:,:}$, using a reduce-scatter operation, with communication cost [23, §4.2]

$$\mathcal{C}_1^{\mathbf{A}} = t_s \left(p_{\mathbf{B}} \ + p_{\mathbf{C}} - 2\right) + t_w \frac{IR}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}} \left(p_{\mathbf{B}} p_{\mathbf{C}} - 1\right).$$

2. Processors $p_{i_{\mathbf{A}},:,:}$ learn the updated $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ through an all-gather operation on its updated parts, each of dimension $\frac{I}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}} \times R$, with communication cost [23, §4.2]

$$\mathcal{C}_2^{\mathbf{A}} = t_s \left(p_{\mathbf{B}} \ + p_{\mathbf{C}} - 2\right) + t_w \frac{IR}{p_{\mathbf{A}} p_{\mathbf{B}} p_{\mathbf{C}}} \left(p_{\mathbf{B}} p_{\mathbf{C}} - 1\right).$$

3. Finally, $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$ is computed by using an all-reduce operation on quantities

$$\left(\mathbf{A}_{k+1}^{i_{\mathbf{A}}}\right)^T \mathbf{A}_{k+1}^{i_{\mathbf{A}}}, \quad i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}},$$

on each single-dimensional processor group $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$, with communication cost [23, §4.3]

$$\mathcal{C}_3^{\mathbf{A}} = \left(t_s + t_w R^2\right) \log_2 p_{\mathbf{A}}. \tag{3.13}$$

The communication that takes place during the acceleration step involves scalar quantities and, thus, is ignored.

When we are dealing with large messages, the t_w terms dominate the communication

Table 3.1: Average, over 10 realizations, `cputime` and maximum relative factor error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for true latent factors with i.i.d. entries, uniform in $[0, 1]$

Size	R	σ_N^2	AO-Nesterov		`sdf_nls`		`parafac`	
			`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$
$1000 \times 100 \times 100$	15	10^{-2}	29	80	56	79	44	85
		10^{-4}	27	10	52	13	53	8
	50	10^{-2}	77	89	217	91	191	91
		10^{-4}	76	13	221	24	251	9
$500 \times 500 \times 100$	15	10^{-2}	63	35	126	37	72	42
		10^{-4}	64	5	132	10	105	4
	50	10^{-2}	119	39	347	43	250	42
		10^{-4}	124	8	331	20	327	5
$300 \times 300 \times 300$	15	10^{-2}	72	27	84	27	70	38
		10^{-4}	71	5	87	7	106	3
	50	10^{-2}	114	31	171	32	230	34
		10^{-4}	119	8	174	13	279	4

cost. Thus, if we ignore the startup time, the total communication time is

$$\begin{aligned} \mathcal{C}^{\mathbf{A}} &= t_w \left(\frac{2IR}{p_{\mathbf{A}}\, p_{\mathbf{B}}\, p_{\mathbf{C}}} \left(p_{\mathbf{B}} p_{\mathbf{C}} - 1\right) + R^2 \log_2 p_{\mathbf{A}} \right) \\ &\approx t_w \left(\frac{2IR}{p_{\mathbf{A}}} + R^2 \log_2 p_{\mathbf{A}} \right) \\ &\approx \frac{2IR\, t_w}{p_{\mathbf{A}}}, \end{aligned} \tag{3.14}$$

with the second approximation being accurate for $R \ll \frac{I}{p_{\mathbf{A}}}$. The presence of $p_{\mathbf{A}}$ in the denominator of the last expression of (3.14) implies that our implementation is scalable in the sense that, if we double I, then we can have (approximately) the same communication cost per processor by doubling $p_{\mathbf{A}}$.

Analogous results hold for the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$.

3.4 Numerical Experiments

3.4.1 Matlab environment

In this subsection, we test the effectiveness of the Nesterov-based AO NTF algorithm with numerical experiments performed in Matlab.

At first, we compare the performance of the algorithm we propose in Algorithm 2 for the solution of the MNLS problem (2.6) with that of the algorithm proposed in [13] and [14] (for the moment, we ignore the proximal term, thus, we put $\lambda = 0$ in Algorithm 2). As we mentioned in subsection 2.2, if matrix $\mathbf{B}^T\mathbf{B}$ is rank deficient, that is, if $\mu = 0$, then both algorithms have practically the same behavior. However, if $\mathbf{B}^T\mathbf{B}$ is full-rank, then the two algorithms exhibit different behavior. In order to illustrate their difference, we perform the following experiment. We generate random matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times r}$ with $m = 300$, $n = 200$, and $r = 100$, with independent and identically distributed (i.i.d) elements, taking values uniformly at random in the interval $[0, 1]$. Then,

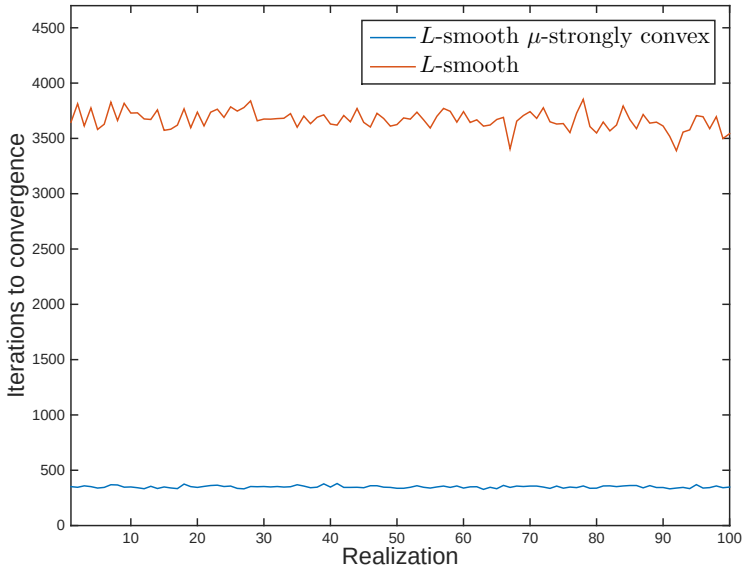

Figure 3.2: Number of iterations to convergence for (blue line) Algorithm 2, with $\lambda = 0$, and (red line) algorithm of [13].

Table 3.2: Average, over 10 realizations, `cputime` and maximum relative factor error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for true latent factors with correlated entries

Size	R	σ_N^2	Bottleneck	AO-Nesterov		`sdf_nls`		`parafac`	
				`cputime`	`MRFE`	`cputime`	`MRFE`	`cputime`	`MRFE`
$300 \times 300 \times 300$	50	10^{-4}	$\mathbf{A}$	132	0.0074	194	0.0075	356	0.0073
			$\mathbf{A}, \mathbf{B}$	204	0.0116	254	0.0195	412	0.0122
			$\mathbf{A}, \mathbf{B}, \mathbf{C}$	271	0.0206	370	0.1007	779	0.0168

we solve problem (2.6) with the two algorithms, starting from the same random point. The terminating conditions are determined by parameters $\delta_1 = \delta_2 = 10^{-3}$. In Figure 3.2, we plot the number of iterations needed by the two algorithms to converge over 100 independent realizations. We observe that the Nesterov-type algorithm which exploits strong convexity is much more efficient than the algorithm which does not. Thus, in the sequel, we shall not present performance results involving the algorithm of [13].

Next, we compare the performance of a Matlab implementation of the proposed algorithm with routines `parafac` of the n-way toolbox [22] and `sdf_nls` of tensorlab [24]. Our aim is to provide some general observations about the difficulty of the problems and the behavior of the algorithms and not a strict ranking of the algorithms.[3]

The `parafac` routine essentially implements an AO NTF algorithm, where each MNLS problem is solved via the function `fastnnls`, which is based on [25, §23.3]. It also incorporates the normalization and acceleration schemes briefly described in Section 3.2. The `sdf_nls` routine for NTF first applies a "squaring" transformation to the problem variables [26] and then solves an unconstrained problem via an AOO-based Gauss-Newton method.

[3]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.

Table 3.3: `cputime` and relative factorization error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for real-world data

Size	R	AO-Nesterov		`sdf_nls`		`parafac`	
		`cputime`	`RFE`	`cputime`	`RFE`	`cputime`	`RFE`
$1021 \times 1343 \times 33$	10	127	0.2361	1984	0.2483	156	0.2349
	20	409	0.1741	2660	0.2183	421	0.1738
	30	518	0.1446	3072	0.2202	547	0.1444

In our experiments with synthetic data, we focus on the `cputime` and the Maximum, over the three latent factors, Relative Factor Error (MRFE), which is computed via function `cpd_err` of tensorlab.

In the numerical experiments we present in this subsection, we choose the parameter values that determine the terminating conditions so that all algorithms achieve (approximately) the same average MRFEs (of course, this is not always possible with one set of parameter values). Thus, we set Tol $= 10^{-5}$ for `parafac`, TolFun $= 10^{-9}$ for `sdf_nls`, and δ_1 and δ_2, which determine the terminating conditions for the Nesterov-based MNLS, are set to $\delta_1 = \delta_2 = 10^{-2}$. The outer iterations of the Nesterov-based AO NTF terminate if the relative changes of the normalized latent factors become sufficiently small, that is,

$$\frac{\|\mathbf{M}_{k+1}^{\mathcal{N}} - \mathbf{M}_{k}^{\mathcal{N}}\|_F}{\|\mathbf{M}_{k}^{\mathcal{N}}\|_F} < \text{tol}_{\text{AO}}, \text{ for } \mathbf{M} = \mathbf{A}, \mathbf{B}, \mathbf{C}, \tag{3.15}$$

where $\text{tol}_{\text{AO}} = 10^{-4}$.

The proximal parameter λ is computed as

$$\lambda := g(L, \mu) = \begin{cases} 10\mu, & \text{if } \frac{L}{\mu} > 10^6, \\ \mu, & \text{if } 10^6 > \frac{L}{\mu} > 10^4, \\ \frac{\mu}{10}, & \text{if } 10^4 > \frac{L}{\mu}. \end{cases} \tag{3.16}$$

All algorithms start from the same triple of random matrices, $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$, which have i.i.d. elements, uniformly distributed in $[0, 1]$.

True latent factors with i.i.d. elements

We start with synthetic data by assuming that the true latent factors consist of i.i.d. elements, uniformly distributed in $[0, 1]$. The additive noise is zero-mean white Gaussian with variance σ_N^2.

In Table 3.1, we present the average, over 10 realizations, `cputime` and MRFE for various tensor "shapes," ranks $R = 15, 50$, and noise variances $\sigma_N^2 = 10^{-2}, 10^{-4}$. We observe that the Nesterov-based AO NTF is very competitive in all cases, in the sense that it converges fast, achieving very good accuracy in most of the cases.

True latent factors with correlated elements

It is well-known that, if some columns of (at least) one latent factor are almost collinear, convergence of the AO algorithm tends to be slow (these cases are known as "bottlenecks")

[20]. In the sequel, we test the behavior of the three algorithms in cases with one, two, and three bottlenecks. More specifically, we generate the true latent factors with i.i.d. elements as before and we create a single "bottleneck" by modifying the last two columns of one latent factor so that each becomes highly correlated with another column of the same latent factor (the correlation coefficient is larger that 0.98). In an analogous way, we generate double and triple "bottlenecks."

In Table 3.2, we focus on the case $I = J = K = 300$, $R = 50$, $\sigma_N^2 = 10^{-4}$, and present the average, over 10 realizations, `cputime` and MRFE. We observe that the problems become more difficult as the number of bottlenecks increases, in the sense that both the `cputime` and the MRFE increase as the number of bottlenecks increases. Again, the Nesterov-based AO NTF algorithm is very efficient in all cases. Analogous observations have been made in extensive numerical experiments with other tensor shapes and noise levels.

Real-world data

In order to test the behavior of the aforementioned algorithms with real-world data, we use the tensor with size $1021 \times 1343 \times 33$ derived from the hyperspectral image "Souto_Wood_Pile" [27]. Since, in this case, the true latent factors are unknown, we focus on the `cputime` and the Relative Factorization Error (RFE), defined as

$$\texttt{RFE}(\mathbf{A}, \mathbf{B}, \mathbf{C}) := \frac{\|\boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_F}{\|\boldsymbol{\mathcal{X}}\|_F}.$$

In Table 3.3, we present the average `cputime` and `RFE` for ranks $R = 10, 20, 30$. The averages are with respect to the initial points $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$, which are random with i.i.d. elements uniformly distributed in $[0, 1]$, and are computed over 5 realizations. We observe that the Nesterov-based AO NTF is very efficient in these cases as well.

3.4.2 Parallel environment - MPI

We now present results obtained from the MPI implementation described in detail in Section 3.3. The program is executed on a DELL PowerEdge R820 system with SandyBridge - Intel(R) Xeon(R) CPU $E5 - 4650v2$ (in total, 16 nodes with 40 cores each at 2.4 Gz) and 512 GB RAM per node. The matrix operations are implemented using routines of the C++ library Eigen [28]. We assume a noiseless tensor $\boldsymbol{\mathcal{X}}$, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$. The terminating conditions for MNLS are determined by values $\delta_1 = \delta_2 = 10^{-2}$.

The AO terminates at iteration k if (recall that tensor $\boldsymbol{\mathcal{X}}$ is noiseless)

$$\texttt{RFE}(\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k) < 10^{-3}.$$

We test the behavior of our implementation for various tensor sizes and rank $R = 15, 50, 100$. The performance metric we compute is the speedup attained using $p = p_{\mathbf{A}} \times p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors.

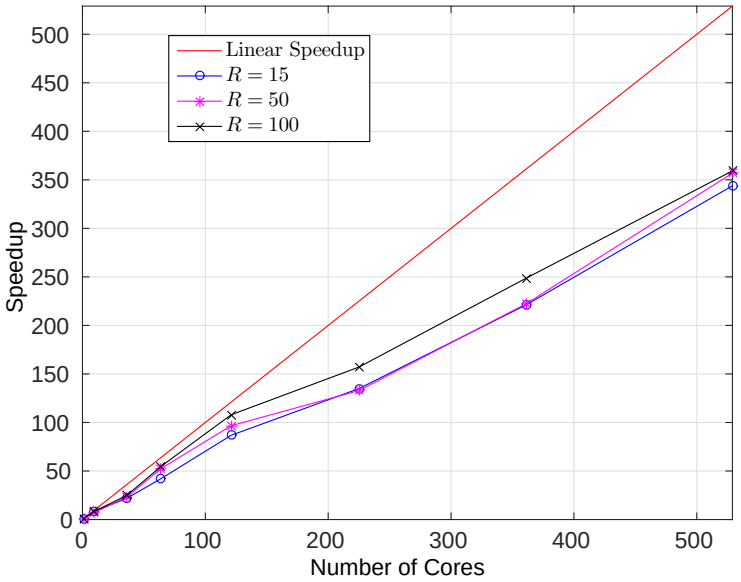

Figure 3.3: Speedup achieved for a $2000 \times 2000 \times 2000$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

In Figures 3.3-3.6, we plot the speedup for the following cases (in all cases with synthetic data, the tensor $\boldsymbol{\mathcal{X}}$ has eight billion entries):

1. Cubic tensor: we set $I = J = K = 2000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = p_{\mathbf{C}} = \sqrt[3]{p}$, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

2. One large dimension: we set $I = 400$, $J = 400$, $K = 50000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = 1$, $p_{\mathbf{C}} = p$, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

3. Two large dimensions: we set $I = 5000$, $J = 320$, $K = 5000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{C}} = \sqrt{p}$, $p_{\mathbf{B}} = 1$, for $p = 1, 9, 36, 64, 121, 225, 361, 529$.

4. Finally, we use the hyperspectral image from the previous section, with $I = 1021$, $J = 1343$, $K = 33$, and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = \sqrt{p}$, $p_{\mathbf{C}} = 1$, for $p = 1, 9, 36, 64$.

We observe that, in all cases, we attain significant speedup, which is rather insensitive to the tensor shape and rank.

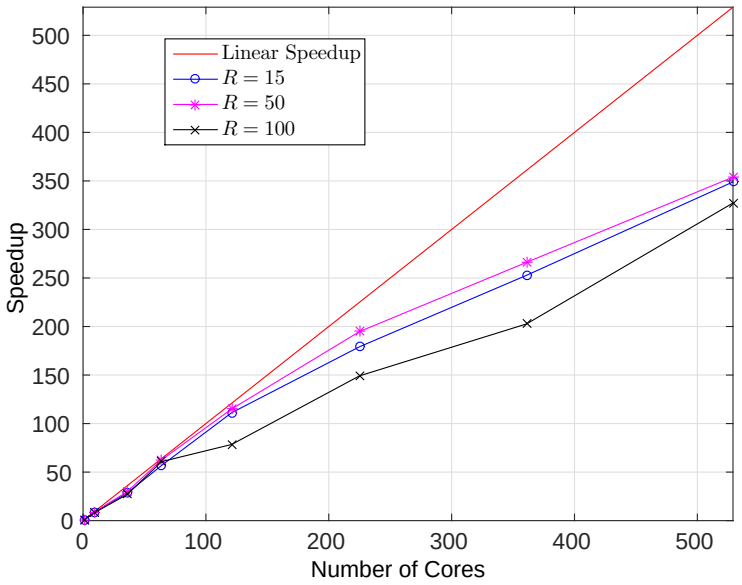

Figure 3.4: Speedup achieved for a $400 \times 400 \times 50000$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343, 512$.

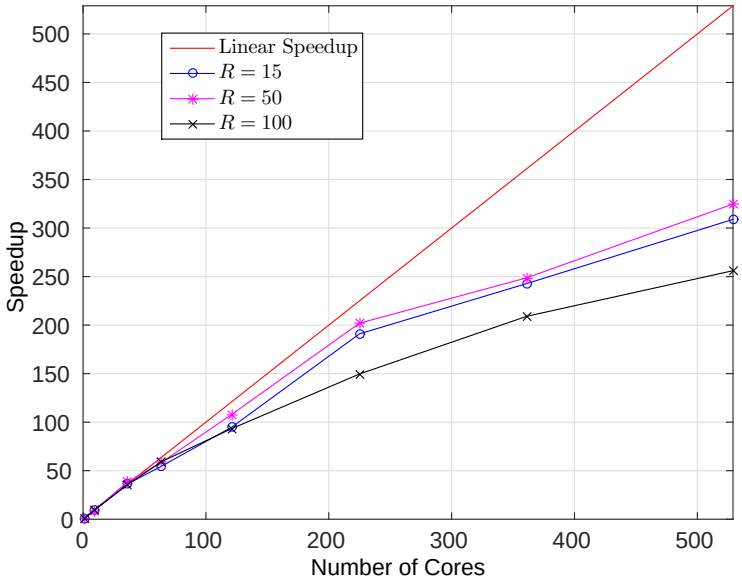

Figure 3.5: Speedup achieved for a $5000 \times 320 \times 5000$ tensor with p cores, for $p = 1, 9, 36, 64, 121, 225, 361, 529$.

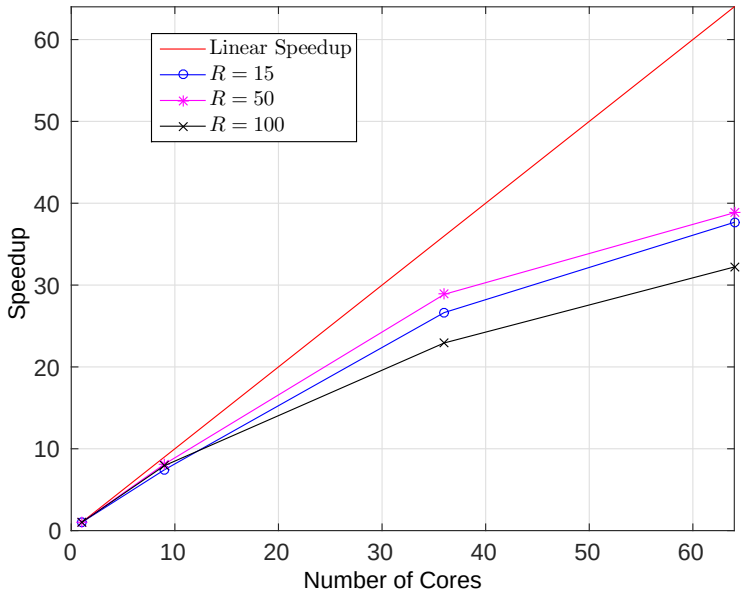

Figure 3.6: Speedup achieved for the hyperspectral image "Souto_Wood_Pile" [27] with p cores, for $p = 1, 9, 36, 64$.

Chapter 4

Nonnegative Tensor Completion

The problem of tensor completion arises in many modern applications, such as machine learning, signal processing, and scientific computing, where we want to estimate missing values in multi-way data, using only the available elements and structural properties of the data.

Completion problems are closely related to recommendation problems, which can be viewed as completing a partially observable user-item matrix whose entries are ratings. Matrix factorization was empirically shown to be a better model than traditional nearest-neighbour based approaches in the Netflix Prize competition [29].

In our case, we model rating data as tensors and use contextual information, such as time and location, for enhancing the recommendation quality. Similar to the matrix case, we employ factorization techniques to provide accurate recommendations. Other approaches for contextual recommendations use context as a means to pre-filter or post-filter the recommendations made [30].

4.1 Problem Formulation

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ be an incomplete tensor, and $\Omega \subseteq \{1 \dots I\} \times \{1 \dots J\} \times \{1 \dots K\}$ be the set of indices of its known entries. That is $\boldsymbol{\mathcal{X}}(i,j,k)$ is known if $(i,j,k) \in \Omega$. Also, let $\boldsymbol{\mathcal{M}}$ be a tensor with the same size as $\boldsymbol{\mathcal{X}}$, with elements $\boldsymbol{\mathcal{M}}(i,j,k)$ equal to one or zero based on the availability of the corresponding element of $\boldsymbol{\mathcal{X}}$. That is $\boldsymbol{\mathcal{M}}(i,j,k) = 1$ if $(i,j,k) \in \Omega$, and $\boldsymbol{\mathcal{M}}_{i,j,k} = 0$ otherwise. The nonnegative tensor completion problem can be expressed as

$$\min_{\mathbf{A}\geq\mathbf{0},\mathbf{B}\geq\mathbf{0},\mathbf{C}\geq\mathbf{0}} \quad f_\Omega(\mathbf{A},\mathbf{B},\mathbf{C}) + \tfrac{\lambda}{2}\|\mathbf{A}\|_F^2 + \tfrac{\lambda}{2}\|\mathbf{B}\|_F^2 + \tfrac{\lambda}{2}\|\mathbf{C}\|_F^2\,, \tag{4.1}$$

where

$$f_\Omega(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{1}{2}\,\|\boldsymbol{\mathcal{M}} \circledast (\boldsymbol{\mathcal{X}} - [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!])\|_F^2\,.$$

We can derive matrix-based equivalent expressions as before. More specifically,

$$\begin{aligned} f_\Omega(\mathbf{A},\mathbf{B},\mathbf{C}) &= \frac{1}{2}\,\|\mathbf{M}_\mathbf{A} \circledast \left(\mathbf{X}_\mathbf{A} - \mathbf{A}(\mathbf{C}\odot\mathbf{B})^T\right)\|_F^2 \\ &= \frac{1}{2}\,\|\mathbf{M}_\mathbf{B} \circledast \left(\mathbf{X}_\mathbf{B} - \mathbf{B}(\mathbf{C}\odot\mathbf{A})^T\right)\|_F^2 \\ &= \frac{1}{2}\,\|\mathbf{M}_\mathbf{C} \circledast \left(\mathbf{X}_\mathbf{C} - \mathbf{C}(\mathbf{B}\odot\mathbf{A})^T\right)\|_F^2, \end{aligned}$$

where $\mathbf{M}_\mathbf{A}$, $\mathbf{M}_\mathbf{B}$, and $\mathbf{M}_\mathbf{C}$ are the matrix unfoldings of $\boldsymbol{\mathcal{M}}$ with respect to the first, second, and third mode, respectively.

4.2 Nonnegative Matrix Completion

First, we consider the Nonnegative Matrix Completion problem, whose solution will be the building block for the solution of the AO NT Completion problem. Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{A} \in \mathbb{R}^{m\times r}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$. Also, let $\Omega \subseteq \{1 \dots m\} \times \{1 \dots n\}$ be the set of indices of the known entries of $\mathbf{X}$, and $\mathbf{M}$ be a matrix with the same size as $\mathbf{X}$, with elements $\mathbf{M}(i, j)$ equal to one or zero based on the availability of the corresponding element of $\mathbf{X}$. We consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f_\Omega(\mathbf{A}) := \frac{1}{2} \|\mathbf{M} \circledast \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right) \|_F^2. \tag{4.2}$$

The gradient of f_Ω, at point $\mathbf{A}$, is

$$\nabla f_\Omega(\mathbf{A}) = -\left(\mathbf{M} \circledast \mathbf{X} - \mathbf{M} \circledast \mathbf{A}\mathbf{B}^T\right) \mathbf{B}. \tag{4.3}$$

A crucial part of the Nesterov MNLS algorithm is the computation of values μ and L. The optimal values, μ_{opt} and L_{opt}, are computed from the Hessian of the problem.

It can be shown that the Hessian of f_Ω, at point $\mathbf{A}$, is

$$\nabla^2 f_\Omega(\mathbf{A}) = \left(\mathbf{B}^T \otimes \mathbf{I}\right) \texttt{diag}\left(\texttt{vec}\left(\mathbf{M}\right)\right) \texttt{diag}\left(\texttt{vec}\left(\mathbf{M}\right)\right) \left(\mathbf{B} \otimes \mathbf{I}\right). \tag{4.4}$$

As the size of the problem grows, the computation of (4.4) becomes very demanding. A good approximation is to set $\mu := 0$ and $L := \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, where $\mathbf{B}^T\mathbf{B}$ is the Hessian of the problem with no missing entries. We have observed that, in practice, our choice for μ is very accurate for very sparse problems, while our choice for L is an efficiently computed good upper bound of L_{opt}. Thus, we have

$$\mathbf{O} \approx \mu_{\text{opt}}\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L_{\text{opt}}\mathbf{I} \preceq L\mathbf{I}, \quad \forall\, \mathbf{x} \in \mathbb{R}^n.$$

The modified algorithm for the case of missing elements is given in Algorithm 4.

4.3 Nesterov Based AO NTC

In Algorithm 5, we present the Nesterov-based AO NTC. We start from point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, MNLS Completion problems, based on the previous estimates.

The most demanding computations during the update of matrix $\mathbf{A}_k$ via the Nesterov-type MNLS Completion algorithm are

$$\mathbf{W}_\mathbf{A} = (\mathbf{M}_\mathbf{A} \circledast \mathbf{X}_\mathbf{A})(\mathbf{C} \odot \mathbf{B}) \tag{4.5}$$

and

$$\mathbf{Z}_\mathbf{A} = (\mathbf{M}_\mathbf{A} \circledast \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T)(\mathbf{C} \odot \mathbf{B}), \tag{4.6}$$

and should be studied separately. Analogous quantities are computed for the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$.

Algorithm 4: Nesterov-type algorithm for MNLS Completion

Input: $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{M} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times r}$, $\mathbf{A}_* \in \mathbb{R}^{m \times r}$, λ

$L = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, $\mathbf{W} = -(\mathbf{M} \circledast \mathbf{X})\mathbf{B}$

$q = \frac{\lambda}{L+\lambda}$

$\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$

$\alpha_0 = 1$, $k = 0$

while (1) **do**

  $\nabla f_\Omega(\mathbf{Y}_k) = \mathbf{W} + (\mathbf{M} \circledast \mathbf{Y}_k\mathbf{B}^T)\mathbf{B} + \lambda\mathbf{Y}_k$

  if (terminating_condition is TRUE) **then**

    break

  else

    $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda}\nabla f(\mathbf{Y}_k)\right)_+$

    $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$

    $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$

    $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta_{k+1}(\mathbf{A}_{k+1} - \mathbf{A}_k)$

    $k = k + 1$

return $\mathbf{A}_k$.

Algorithm 5: Nesterov-based AO NTC

Input: $\boldsymbol{\mathcal{X}}$, Ω, $\mathbf{A}_0 > \mathbf{0}$, $\mathbf{B}_0 > \mathbf{0}$, $\mathbf{C}_0 > \mathbf{0}$.

while (1) **do**

  $\mathbf{A}_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{X}_\mathbf{A}, (\mathbf{C}_k \odot \mathbf{B}_k), \mathbf{A}_k)$

  $\mathbf{B}_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{X}_\mathbf{B}, (\mathbf{C}_k \odot \mathbf{A}_{k+1}), \mathbf{B}_k)$

  $\mathbf{C}_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{X}_\mathbf{C}, (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}), \mathbf{C}_k)$

  if (terminating_condition is TRUE) **then** break; **endif**

return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

The computation of the i-th row of $\mathbf{W}_\mathbf{A}$, for $i = 1 \dots I$, is given by

$$\begin{aligned}\mathbf{W}_\mathbf{A}(i,:) &= \Big(\mathbf{M}_\mathbf{A}(i,:) \circledast \mathbf{X}_\mathbf{A}(i,:)\Big)(\mathbf{C} \odot \mathbf{B}) \\ &= \underbrace{\Big(\mathbf{M}_\mathbf{A}(i,:) \circledast \mathbf{X}_\mathbf{A}(i,:)\Big)}_{JK} \left.\begin{bmatrix} \mathbf{C}(1,:) \odot \mathbf{B} \\ \vdots \\ \mathbf{C}(K,:) \odot \mathbf{B} \end{bmatrix}\right\} JK.\end{aligned}$$

The computation involves the multiplication of a $(1 \times JK)$ row vector and a $(JK \times R)$ matrix. A direct multiplication would be prohibitive, since, in most applications, I, J, and K could reach the order of millions or even billions. In order to reduce the computational complexity, we must exploit the sparsity of $\boldsymbol{\mathcal{X}}$.

Let nnz_i be the number of known entries in the $i-$th horizontal slice of $\boldsymbol{\mathcal{X}}$. Also, let these known entries have indices $(i, j_q, k_q) \in \Omega$, for $q = 1 \dots nnz_i$. When we matricize with respect to the first mode, every known element $\boldsymbol{\mathcal{X}}(i, j_q, k_q)$ maps to $\mathbf{X}_\mathbf{A}(i, k_q J + j_q)$, for $q = 1 \dots nnz_i$.

Also, the $(k_q J + j_q)$-th row of the Khatri-Rao product corresponds to $\mathbf{C}(k_q, :) \circledast$

$\mathbf{B}(j_q,:)$. Thus, the computation of the i-th row of $\mathbf{W_A}$ reduces to

$$\begin{aligned}\mathbf{W_A}(i,:) &= \underbrace{\left[\mathbf{X_A}(i,k_1J+j_1)\ldots\mathbf{X_A}(i,k_{nnz_i}J+j_{nnz_i})\right]}_{nnz_i}\left.\begin{bmatrix}\mathbf{C}(k_1,:)\circledast\mathbf{B}(j_1,:)\\ \vdots\\ \mathbf{C}(k_{nnz_i},:)\circledast\mathbf{B}(j_{nnz_i},:)\end{bmatrix}\right\}nnz_i\\ &=\sum_{q=1}^{nnz_i}\mathbf{X_A}(i,k_qJ+j_q)\,\mathbf{C}(k_q,:)\circledast\mathbf{B}(j_q,:)\\ &=\sum_{q=1}^{nnz_i}\boldsymbol{\mathcal{X}}(i,j_q,k_q)\,\mathbf{C}(k_q,:)\circledast\mathbf{B}(j_q,:).\end{aligned}$$

Similarly, we obtain an efficient computation of $\mathbf{Z_A}$ as

$$\mathbf{Z_A}(i,:)=\sum_{q=1}^{nnz_i}\Big(\mathbf{A}(i,:)\big(\mathbf{C}(k_q,:)\circledast\mathbf{B}(j_q,:)\big)^T\Big)\big(\mathbf{C}(k_q,:)\circledast\mathbf{B}(j_q,:)\big).$$

4.4 Parallel Implementation

Motivated by the coarse-grained approach of [31], we consider the implementation of the Nesterov-based AO NTC algorithm on a system with p processing elements. In order to describe the parallel implementation, we introduce certain partitionings of the factor matrices and the tensor matricizations. We partition the factor matrices $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$ in block rows as

$$\mathbf{A}_k=\begin{bmatrix}\left(\mathbf{A}_k^1\right)^T & \cdots & \left(\mathbf{A}_k^p\right)^T\end{bmatrix}^T, \tag{4.7}$$

with $\mathbf{A}_k^i\in\mathbb{R}^{\frac{I}{p}\times R}$, for $i=1,\ldots,p$, and analogous partitionings for $\mathbf{B}_k$ and $\mathbf{C}_k$, each of size $\frac{J}{p}\times R$ and $\frac{K}{p}\times R$, respectively. The i-th processing element computes the i-th block row of $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$, for $i=1,\ldots,p$. We partition accordingly the matricization $\mathbf{X_A}$ and get

$$\mathbf{X_A}=\begin{bmatrix}\left(\mathbf{X_A}^1\right)^T & \cdots & \left(\mathbf{X_A}^p\right)^T\end{bmatrix}^T, \tag{4.8}$$

with $\mathbf{X_A}^i\in\mathbb{R}^{\frac{I}{p}\times JK}$. In a similar manner, we partition $\mathbf{X_B}$ and $\mathbf{X_C}$ into p block rows, each of size $\frac{J}{p}\times IK$ and $\frac{K}{p}\times IJ$, respectively. The i-th block row of $\mathbf{X_A}$, $\mathbf{X_B}$, $\mathbf{X_C}$ have been allocated to the i-th processing element, for $i=1,\ldots,p$.

A distributed-memory implementation of the NTC is given in Algorithm 6

In the sequel, we describe the computation of $\mathbf{A}_{k+1}$. The algorithm proceeds as follows. All processing elements work in parallel. The i-th processing element uses its local data $\mathbf{X}_\mathbf{A}^n$, as well as the whole matrices $\mathbf{B}_k$ and $\mathbf{C}_k$, and computes the i-th block row of matrix $\mathbf{A}_{k+1}$, $\mathbf{A}_{k+1}^i$. Then, each processing element broadcasts its output to all other processing elements; this operation can be implemented via the MPI statement `MPI_Allgather`. At the end of this step, all processing elements possess $\mathbf{A}_{k+1}$. Then, we compute $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$ with analogous computations, completing one iteration of the AO NTC algorithm. The algorithm continues until convergence.

The communication requirements of this implementation consist of one `Allgather`

operation per MNLS, implying gathering of terms with IF, JF, and KF elements per outer iteration.

Algorithm 6: Parallel AO NTC

Input: Processing element n, for $n = 1, \ldots, N_p$, knows $\boldsymbol{\mathcal{X}}^n$. All processing elements know $\mathbf{B}_0$, $\mathbf{C}_0$, tol > 0.

Set $k = 0$
while (terminating condition is FALSE) **do**
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{A}}(i,:) = \sum_{q=1}^{nnz_i} \boldsymbol{\mathcal{X}}^n(i, j_q, k_q)\, \mathbf{C}(k_q,:) \circledast \mathbf{B}(j_q,:)$,
    $\mathbf{Z}^n_{\mathbf{A}}(i,:) = \sum_{q=1}^{nnz_i} \Big(\mathbf{A}(i,:)\big(\mathbf{C}(k_q,:) \circledast \mathbf{B}(j_q,:)\big)^T\Big)\big(\mathbf{C}(k_q,:) \circledast \mathbf{B}(j_q,:)\big)$
    $\mathbf{A}^n_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{W}^n_{\mathbf{A}}, \mathbf{Z}^n_{\mathbf{A}}, \mathbf{A}^n_k, \text{tol})$
    $\text{All\_gather}(\mathbf{A}^n_{k+1})$
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{B}}(j,:) = \sum_{q=1}^{nnz_j} \boldsymbol{\mathcal{X}}^n(i_q, j, k_q)\, \mathbf{C}(k_q,:) \circledast \mathbf{A}(i_q,:)$,
    $\mathbf{Z}^n_{\mathbf{B}}(j,:) = \sum_{q=1}^{nnz_j} \Big(\mathbf{B}(j,:)\big(\mathbf{C}(k_q,:) \circledast \mathbf{A}(i_q,:)\big)^T\Big)\big(\mathbf{C}(k_q,:) \circledast \mathbf{A}(i_q,:)\big)$
    $\mathbf{B}^n_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{W}^n_{\mathbf{B}}, \mathbf{Z}^n_{\mathbf{B}}, \mathbf{B}^n_k, \text{tol})$
    $\text{All\_gather}(\mathbf{B}^n_{k+1})$
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{C}}(k,:) = \sum_{q=1}^{nnz_k} \boldsymbol{\mathcal{X}}^n(i_q, j_q, k)\, \mathbf{B}(j_q,:) \circledast \mathbf{A}(i_q,:)$,
    $\mathbf{Z}^n_{\mathbf{C}}(k,:) = \sum_{q=1}^{nnz_k} \Big(\mathbf{C}(k,:)\big(\mathbf{B}(j_q,:) \circledast \mathbf{A}(i_q,:)\big)^T\Big)\big(\mathbf{B}(j_q,:) \circledast \mathbf{A}(i_q,:)\big)$
    $\mathbf{C}^n_{k+1} = \text{Nesterov\_MNLS\_Completion}(\mathbf{W}^n_{\mathbf{C}}, \mathbf{Z}^n_{\mathbf{C}}, \mathbf{C}^n_k, \text{tol})$
    $\text{All\_gather}(\mathbf{C}^n_{k+1})$
  $k = k + 1$
return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

4.5 Numerical Experiments

In this section, we present results obtained from the MPI implementation of the AO NTC. The program is executed on a DELL PowerEdge R820 system with SandyBridge - Intel(R) Xeon(R) CPU E5 − 4650v2 (in total, 16 nodes with 40 cores each at 2.4 Gz) and 512 GB RAM per node. The matrix operations are implemented using routines of the C++ library Eigen [28].

We test the behavior of our implementation using both synthetic and real data. The performance metric we compute is the speedup attained using p processors, and the accuracy of the predictions. For the computation of the speedup, we measure the execution time for 10 outer iterations; for each matrix completion problem we perform 100 (inner) iterations.

The dataset we used is the MovieLens 10M dataset [32], which contains time-stamped ratings of movies. Binning the time into seven-day-wide bins, results in a tensor of size $71567 \times 65133 \times 171$. The number of samples is 8000044 (99.99% sparsity). In order to distribute the known entries as uniformly as possible across the p processors, and resolve load imbalance issues, we first perform a random permutation on our data.

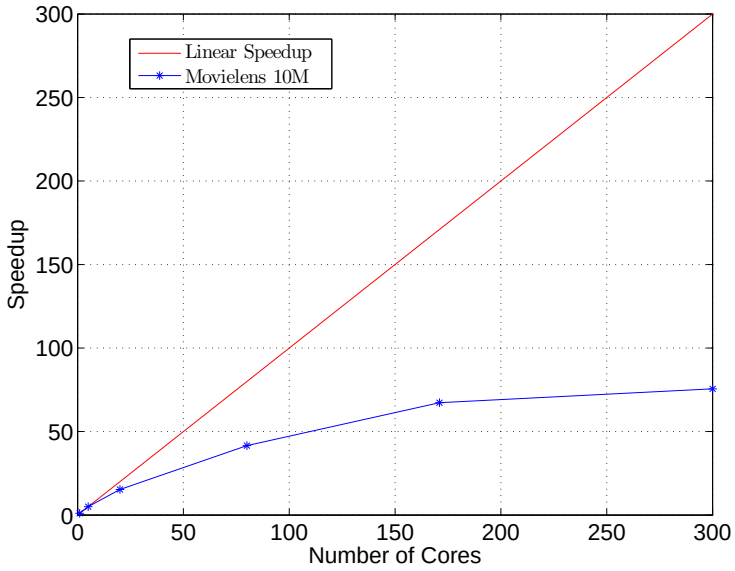

Figure 4.1: Speedup achieved for the MovieLens 10M dataset of size $71567 \times 65133 \times 171$ with p cores, for $p = 1, 5, 20, 171, 300$.

In Figure 4.1, we plot the speedup for the MovieLens 10M dataset, for $p = 1, 5, 20, 171$, and 300. In Figure 4.2, we plot the speedup for a tensor with synthetic data of the same size and sparsity level as the MovieLens 10M dataset, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$, for $p = 1, 5, 20, 171, 300$. In both cases, we use rank $R = 10$.

For the MovieLens 10M dataset, we test the completion accuracy by measuring the mean squared error of 2000000 known ratings with our predictions. The mean squared error we achieved is 0.0033, making our predictions quite accurate.

We observe that, in the case of synthetic data, we attain greater speedup. We attribute this fact to the more uniform distribution of the known entries across the processing elements. More advanced techniques for load imbalance issues should be considered in the future.

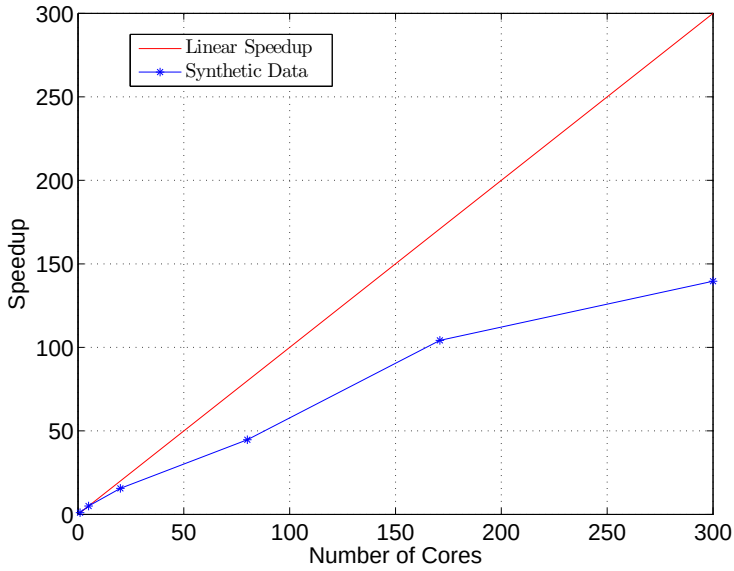

Figure 4.2: Speedup achieved for a $71567 \times 65133 \times 171$ tensor with p cores, for $p = 1, 5, 20, 171, 300$.

Chapter 5

Conclusion and Future Work

We considered the NTF and NTC problems. We adopted the AO framework and solved each MNLS problem via a Nesterov-type algorithm for convex and strongly convex problems. We described in detail parallel implementations of the algorithms. In extensive numerical experiments, the derived algorithms were proven very efficient, compared with state-of-the-art competitors. Our parallel implementations attained significant speedup, rendering our algorithms strong candidates for the solution of very large-scale dense NTF and sparse NTC problems.

Since tensor factorization is a very useful tool, whose popularity has grown significantly, we suggest some ideas for future work. Besides the nonnegativity constraint that we imposed to the factors in our work, it would be interesting to tackle additional constraints, such as sparsity, symmetry, and orthogonality constraints. Also extension to higher-than-three dimensional tensors should be considered.

Furthermore, other tensor factorizations, such as Tucker, PARAFAC2, INDSCAL, should be considered, because they may improve the interpretability of various datasets.

As it was shown in Section 4.5, more advanced load balancing techniques should be examined for the completion problem. Finally, the benefits of incorporating shared memory techniques to our distributed memory implementations should be studied.

Bibliography

[1] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Wiley-Interscience, 2008.

[2] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Wiley, 2009.

[3] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[4] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *Signal Processing Magazine, IEEE*, vol. 32, no. 2, pp. 145–163, 2015.

[6] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[7] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion." *IEEE Journal on Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 586–600, 2015.

[8] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[9] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, accepted for publication, May 2016.

[10] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Computing*, 2015.

[11] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[12] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 77.

[13] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Nenmf: An optimal gradient method for nonnegative matrix factorization," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.

[14] Y. Zhang, G. Zhou, Q. Zhao, A. Cichocki, and X. Wang, "Fast nonnegative tensor factorization based on accelerated proximal gradient and low-rank approximation," *Neurocomputing*, vol. 198, no. Supplement C, pp. 148 – 154, 2016.

[15] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, USA, March 5-9, 2017*. IEEE, 2017.

[16] ——, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementations," *IEEE Transactions on Signal Processing*, to appear.

[17] Y. Nesterov, *Introductory lectures on convex optimization*. Kluwer Academic Publishers, 2004.

[18] B. O' Donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[19] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1126–1153, 2013.

[20] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: A novel method to accelerate parafac," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1128–1147, 2008.

[21] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization," *Computational Optimization and Applications*, vol. 63, no. 1, pp. 121–142, 2016.

[22] C. A. Andersson and R. Bro, "The n-way toolbox for matlab," *Chemometrics Intelligent Laboratory Systems*, vol. 52, pp. 1–4, 2000. [Online]. Available: http://www.models.life.ku.dk/source/nwaytoolbox

[23] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Pearson, 2003.

[24] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab 3.0," Mar. 2016. [Online]. Available: http://www.tensorlab.net/

[25] C. L. Lawson and R. J. Hanson, *Solving Lease Squares Problems*. SIAM, 1995.

[26] J.-P. Royer, N. Thirion-Moreau, and P. Comon, "Computing the polyadic decomposition of nonnegative third order tensors," *Signal Processing*, vol. 91, no. 9, pp. 2159–2171, 2011.

[27] S. M. Nascimento, K. Amano, and D. H. Foster, "Spatial distributions of local illumination color in natural scenes," *Vision Research*, vol. 120, pp. 39–44, 2016.

[28] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[29] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Proceedings of the International Conference on Algorithmic Aspects in Information and Management*, 2008.

[30] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," *Recommender systems handbook*, pp. 217–253, 2011.

[31] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," in *IEEE International Conference on Data Mining, ICDM*, 2014, pp. 989–994.

[32] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, pp. 1–19, Dec. 2015.