Chapter 16
Fast Nonnegative Tensor Factorization with an Active-Set-Like Method

Jingu Kim and Haesun Park

Abstract We introduce an efficient algorithm for computing a low-rank nonnegative CANDECOMP/PARAFAC (NNCP) decomposition. In text mining, signal processing, and computer vision among other areas, imposing nonnegativity constraints to the low-rank factors of matrices and tensors has been shown an effective technique providing physically meaningful interpretation. A principled methodology for computing NNCP is alternating nonnegative least squares, in which the nonnegativity-constrained least squares (NNLS) problems are solved in each iteration. In this chapter, we propose to solve the NNLS problems using the block principal pivoting method. The block principal pivoting method overcomes some difficulties of the classical active method for the NNLS problems with a large number of variables. We introduce techniques to accelerate the block principal pivoting method for multiple right-hand sides, which is typical in NNCP computation. Computational experiments show the state-of-the-art performance of the proposed method.

16.1 Introduction

Tensors are mathematical objects for representing multidimensional arrays. Tensors include vectors and matrices as first-order and second-order special cases, respectively, and more generally, tensors of Nth-order can represent an outer product of N vector spaces. Recently, decompositions and low-rank approximations of tensors have been actively studied and applied in numerous areas including signal processing, image processing, data mining, and neuroscience. Several different decomposition models, their algorithms, and applications are summarized in recent reviews by Kolda and Bader [19] and Acar and Yener [1].

J. Kim (✉) · H. Park
School of Computational Science and Engineering, College of Computing, Georgia Institute of Technology, Atlanta, USA
e-mail: jingu@cc.gatech.edu

H. Park
e-mail: hpark@cc.gatech.edu

M.W. Berry et al. (eds.), *High-Performance Scientific Computing*,
DOI 10.1007/978-1-4471-2437-5_16,

In this chapter, we discuss tensors with nonnegative elements and their low-rank approximations. In particular, we are interested in computing a CANDECOMP/PARAFAC decomposition [5, 11] with nonnegativity constraints on factors. In the context of matrices, when data or signals are inherently represented by nonnegative numbers, imposing nonnegativity constraints to low-rank factors was shown to provide physically meaningful interpretation [21, 26]. Widely known as nonnegative matrix factorization (NMF), it has been extensively investigated and utilized in areas of computer vision, text mining, and bioinformatics. In higher-order tensors with nonnegative elements, tensor factorizations with nonnegativity constraints on factors have been developed in several papers [4, 6, 24, 29]. Interestingly, some method for finding nonnegative factors of higher-order tensors, such as [6], were introduced even before NMF. Recent work dealt with properties such as degeneracy [23] and applications such as sound source separation [9], text mining [2], and computer vision [27].

Suppose a tensor of order three, $\mathcal{X} \in \mathbb{R}^{M_1 \times M_2 \times M_3}$, is given. We will introduce main concepts using this third-order tensor for the sake of simplicity, and will deal with a tensor with a general order later. A canonical decomposition (CANDECOMP) [5], or equivalently the parallel factor analysis (PARAFAC) [11], of $\mathcal{X}$ can be written as

$$\mathcal{X} = \sum_{k=1}^{K} \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k, \tag{16.1}$$

where $\mathbf{a}_k \in \mathbb{R}^{M_1}$, $\mathbf{b}_k \in \mathbb{R}^{M_2}$, $\mathbf{c}_k \in \mathbb{R}^{M_3}$, and "$\circ$" represents an outer product of vectors. Following [19], we will call a decomposition in the form of Eq. (16.1) the CP (CANDECOMP/PARAFAC) decomposition. A tensor in a form of $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ is called a *rank-one* tensor: In the CP decomposition, tensor $\mathcal{X}$ is represented as a sum of K rank-one tensors. A smallest integer K for which Eq. (16.1) holds with some vectors $\mathbf{a}_k$, $\mathbf{b}_k$, and $\mathbf{c}_k$ for $k \in \{1, \ldots, K\}$ is called the *rank* of tensor $\mathcal{X}$. The CP decomposition can be more compactly represented with *factor matrices* (or *loading matrices)*, $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_K]$, $\mathbf{B} = [\mathbf{b}_1 \cdots \mathbf{b}_K]$, and $\mathbf{C} = [\mathbf{c}_1 \cdots \mathbf{c}_K]$, as follows:

$$\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!],$$

where $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = \sum_{k=1}^{K} \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k$ (see [19]). With a tensor $\mathcal{X}$ of rank R, given an integer $K \leq R$, the computational problem of the CP decomposition is finding factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ that best approximates $\mathcal{X}$.

Now, for a tensor $\mathcal{X}$ with only nonnegative elements, we are interested in recovering factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ that also contain only nonnegative components. Using the Frobenius norm as a criterion for approximation, the factor matrices can be found by solving an optimization problem:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \left\| \mathcal{X} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \right\|_F^2 \quad \text{s.t.} \quad \mathbf{A}, \mathbf{B}, \mathbf{C} \geq 0. \tag{16.2}$$

Inequalities $\mathbf{A}, \mathbf{B}, \mathbf{C} \geq 0$ denote that all the elements of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are nonnegative. The factorization problem in Eq. (16.2) is known as nonnegative CP (NNCP). The

computation of NNCP is demanding not only because many variables are involved in optimization but also because nonnegativity constraints are imposed on the factors. A number of algorithms have been developed for NNCP [4, 10, 15, 29], and we will review them in Sect. 16.2.

In this chapter, extending our prior work on NMF [17], we present a new and efficient algorithm for computing NNCP. Our algorithm is based on alternating nonnegativity-constrained least squares (ANLS) framework, where in each iteration the nonnegativity-constrained least squares (NNLS) subproblems are solved. We propose to solve the NNLS problems based on the block principal pivoting method [12]. The block principal pivoting method accelerates the traditional active-set method [20] by allowing exchanges of multiple variables between index groups per iteration. We adopt ideas that improve the block principal pivoting method in multiple right-hand sides [17].

The remaining of this chapter is organized as follows. In Sect. 16.2, related work is reviewed. In Sect. 16.3, the ANLS framework is described, and in Sect. 16.4, the block principal pivoting method is introduced as well as ideas for improvements for multiple right-hand sides. In Sect. 16.5, we describe how the proposed method can be used to solve regularized and sparse formulations. In Sect. 16.6, experimentation settings and results are shown. We conclude this chapter in Sect. 16.7.

Notations Let us summarize some notations used in this chapter. A lowercase or an uppercase letter, such as x or X, is used to denote a scalar; a boldface lowercase letter, such as $\mathbf{x}$, is used to denote a vector; a boldface uppercase letter, such as $\mathbf{X}$, is used to denote a matrix; and a boldface Euler script letter, such as $\boldsymbol{\mathcal{X}}$, is used to denote a tensor of order three or higher. Indices typically grow from 1 to its uppercase letter: For example, $n \in \{1, \ldots, N\}$. Elements of a sequence of vectors, matrices, or tensors are denoted by superscripts within parentheses: $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$. For a matrix $\mathbf{X}$, $\mathbf{x}_i$ denotes its ith column, and x_{ij} denotes its (i, j) component.

16.2 Related Work

Several computational methods have been developed for solving NNCP. Within the ANLS framework, different methods for solving the NNLS subproblems have been proposed. A classical method for solving the NNLS problem is the active set method of Lawson and Hanson [20]; however, applying Lawson and Hanson's method directly to NNCP is extremely slow. Bro and De Jong [4] suggested an improved active-set method to solve the NNLS problems, and Ven Benthem and Keenan [28] further accelerated the active-set method, which was later utilized in NMF [14] and NNCP [15]. In Friedlander and Hatz [10], the NNCP subproblems are solved by a two-metric projected gradient descent method.

In our work of this chapter, we solve the NNLS subproblems using the block principal pivoting method [12, 17]. The block principal pivoting method is similar to the active set method in that (1) the groups of zero and nonzero variables are

explicitly kept track of, and (2) a system of linear equations is solved at each iteration. However, unlike the active set method, the objective function value in the block principal pivoting method does not monotonically decrease. Instead, by exchanging multiple variables between variable groups after each iteration, the block principal pivoting method is much faster than the active set method. Due the relationship with the active set method, we note the block principal pivoting method as an *active-set-like* method.

Numerous other algorithms that are not based on the ANLS framework were suggested. Paatero discussed a Gauss-Newton method [24] and a conjugate gradient method [25], but nonnegativity constraints were not rigorously handled in those work. Extending the multiplicative updating rule of Lee and Seung [22], Welling and Weber [29] proposed a multiplicative updating method for NNCP. Earlier in [6], Carroll et al. proposed a simple procedure that focuses on a rank-one approximation conditioned that other variables are fixed. Recently, Cichocki et al. proposed a similar algorithm, called hierarchical alternating least squares (HALS), which updates each column of factor matrices at a time [8].

16.3 ANLS Framework

We describe the ANLS framework for solving NNCP. Let us consider the a Nth-order tensor $\mathcal{X} \in \mathbb{R}^{M_1 \times \cdots \times M_N}$ and a corresponding factorization problem

$$\begin{aligned} \min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} \quad & f\left(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\right) = \left\| \mathcal{X} - \left[\!\left[\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)} \right]\!\right] \right\|_F^2 \\ \text{s.t.} \quad & \mathbf{A}^{(n)} \geq 0 \quad \text{for } n = 1,\ldots,N, \end{aligned} \tag{16.3}$$

where $\mathbf{A}^{(n)} \in \mathbb{R}^{M_n \times K}$ for $n = 1,\ldots,N$, and

$$\left[\!\left[\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)} \right]\!\right] = \sum_{k=1}^{K} \mathbf{a}_k^{(1)} \circ \cdots \circ \mathbf{a}_k^{(N)}.$$

In order to introduce the ANLS framework, we need definitions of some tensor operations. See Kolda and Bader [19] and references therein for more details of these operations.

Mode-n matricization The mode-n matricization of a tensor $\mathcal{X}$, denoted by $\mathbf{X}_{(n)}$, is a matrix obtained by linearizing all indices except n. More formally, $\mathbf{X}_{(n)}$ is a matrix of size $M_n \times \prod_{k=1,k\neq n}^{N} M_k$, and the $(m_1,\ldots,m_N)$th element of $\mathcal{X}$ is mapped to the (m_n, I)th element of $\mathbf{X}_{(n)}$ where

$$I = 1 + \sum_{k=1}^{N} (m_k - 1) I_k, \quad \text{and} \quad I_k = \prod_{j=1,j\neq n}^{k-1} M_j.$$

Khatri–Rao product The Khatri–Rao product of two matrices $\mathbf{A} \in \mathbb{R}^{J_1 \times L}$ and $\mathbf{B} \in \mathbb{R}^{J_2 \times L}$, denoted by $\mathbf{A} \odot \mathbf{B}$, is defined as

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{b}_1 & a_{12}\mathbf{b}_2 & \cdots & a_{1L}\mathbf{b}_L \\ a_{21}\mathbf{b}_1 & a_{22}\mathbf{b}_2 & \cdots & a_{2L}\mathbf{b}_L \\ \vdots & \vdots & \ddots & \vdots \\ a_{J_1 1}\mathbf{b}_1 & a_{J_1 2}\mathbf{b}_2 & \cdots & a_{J_1 L}\mathbf{b}_L \end{bmatrix}.$$

Using above notations, the approximation model

$$\mathcal{X} \approx [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$$

can be written as, for any $n \in \{1, \ldots, N\}$,

$$\mathbf{X}^{(n)} \approx \mathbf{A}^{(n)} \times \left(\mathbf{B}^{(n)}\right)^T, \tag{16.4}$$

where

$$\mathbf{B}^{(n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \in \mathbb{R}^{(\prod_{k=1, k\neq n}^{N} M_k) \times K}. \tag{16.5}$$

Equation (16.4) is a key relationship that is utilized in the ANLS framework. The ANLS framework is a block-coordinate-descent method applied to Eq. (16.3). First, $\mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$ are initialized with nonnegative components. Then, for $n = 1, \ldots, N$, the following subproblem is solved iteratively:

$$\begin{aligned} \min_{\mathbf{A}^{(n)}} \quad & \left\| \mathbf{B}^{(n)} \times \left(\mathbf{A}^{(n)}\right)^T - \left(\mathbf{X}^{(n)}\right)^T \right\|_F^2 \\ \text{s.t.} \quad & \mathbf{A}^{(n)} \geq 0. \end{aligned} \tag{16.6}$$

The convergence property of a block-coordinate-descent method [3] states that if each subproblem in the form of Eq. (16.6) has a unique solution, then every limit point produced by the ANLS framework is a stationary point. In particular, if matrices $\mathbf{B}^{(n)}$ are of full column rank, each subproblem has a unique solution.

The problem in Eq. (16.6) is in the form of the nonnegativity-constrained least squares (NNLS) problems, and an efficient algorithm to solve the problem will be the subject of next section. For now, typical characteristics of the subproblem in Eq. (16.6) deserves to be noted. Due to the flattening by the Khatri–Rao product, matrix $\mathbf{B}^{(n)}$ in Eq. (16.6) is typically long and thin. Also, as NNCP is often used for low-rank approximation, matrix $(\mathbf{A}^{(n)})^T$ in Eq. (16.6) is typically flat and wide. These properties will be important in designing efficient algorithms for solving Eq. (16.6), which we now describe.

16.4 Block Principal Pivoting Method

The block principal pivoting method, which we adopt in this work to solve Eq. (16.6), was earlier proposed by Judice and Pires [12] for a single right-hand

side case. We will first explain this method and then explain efficient ways to accelerate the multiple right-hand side case as proposed in [17].

The motivation of the block principal pivoting method comes from the difficulty of conventional active set algorithms which occur when the number of variables increases. In the active set method, because typically only one variable is exchanged per iteration between working sets, the number of iterations until termination heavily depends on the number of variables. To accelerate computation, an algorithm whose iteration count does not depend on the number of variables is desirable. The block principal pivoting method manages to do so by exchanging multiple variables at a time.

For the moment, consider an NNLS problem with a single right-hand side vector:

$$\min_{\mathbf{x}\geq 0} \|\mathbf{V}\mathbf{x}-\mathbf{w}\|_2^2, \tag{16.7}$$

where $\mathbf{V}\in\mathbb{R}^{P\times Q}$, $\mathbf{x}\in\mathbb{R}^{Q\times 1}$, and $\mathbf{w}\in\mathbb{R}^{P\times 1}$. The subproblems in Eq. (16.6) are decomposed to independent instances of Eq. (16.7) with respect to each column vector of $(\mathbf{A}^{(n)})^T$. Hence, an algorithm for Eq. (16.7) is a basic building block of an algorithm for Eq. (16.6).

The Karush–Kuhn–Tucker (KKT) optimality conditions for Eq. (16.7) are given as

$$\mathbf{y}=\mathbf{V}^T\mathbf{V}\mathbf{x}-\mathbf{V}^T\mathbf{w}, \tag{16.8a}$$

$$\mathbf{y}\geq 0,\quad \mathbf{x}\geq 0, \tag{16.8b}$$

$$x_q y_q = 0,\quad q=1,\ldots,Q. \tag{16.8c}$$

We assume that the matrix $\mathbf{V}$ has full column rank. In this case, a solution $\mathbf{x}$ that satisfies the conditions in Eqs. (16.8a)–(16.8c) is the optimal solution of Eq. (16.7).

We divide the index set $\{1,\ldots,Q\}$ into two subgroups $\mathcal{F}$ and $\mathcal{G}$ where $\mathcal{F}\cup\mathcal{G}=\{1,\ldots,Q\}$ and $\mathcal{F}\cap\mathcal{G}=\emptyset$. Let $\mathbf{x}_\mathcal{F}$, $\mathbf{x}_\mathcal{G}$, $\mathbf{y}_\mathcal{F}$, and $\mathbf{y}_\mathcal{G}$ denote the subsets of variables with corresponding indices, and let $\mathbf{V}_\mathcal{F}$ and $\mathbf{V}_\mathcal{G}$ denote the submatrices of $\mathbf{V}$ with corresponding column indices. Initially, we assign zeros to $\mathbf{x}_\mathcal{G}$ and $\mathbf{y}_\mathcal{F}$. Then, by construction, $\mathbf{x}=(\mathbf{x}_\mathcal{F},\mathbf{x}_\mathcal{G})$ and $\mathbf{y}=(\mathbf{y}_\mathcal{F},\mathbf{y}_\mathcal{G})$ always satisfy Eq. (16.8c) for any $\mathbf{x}_\mathcal{F}$ and $\mathbf{y}_\mathcal{G}$. Now, we compute $\mathbf{x}_\mathcal{F}$ and $\mathbf{y}_\mathcal{G}$ using Eq. (16.8a) and check whether the computed values of $\mathbf{x}_\mathcal{F}$ and $\mathbf{y}_\mathcal{G}$ satisfy Eq. (16.8b). Computation of $\mathbf{x}_\mathcal{F}$ and $\mathbf{y}_\mathcal{G}$ is done as follows:

$$\mathbf{V}_\mathcal{F}^T\mathbf{V}_\mathcal{F}\mathbf{x}_\mathcal{F}=\mathbf{V}_\mathcal{F}^T\mathbf{w}, \tag{16.9a}$$

$$\mathbf{y}_\mathcal{G}=\mathbf{V}_\mathcal{G}^T(\mathbf{V}_\mathcal{F}\mathbf{x}_\mathcal{F}-\mathbf{w}). \tag{16.9b}$$

One can first solve for $\mathbf{x}_\mathcal{F}$ in Eq. (16.9a) and use it to compute $\mathbf{y}_\mathcal{G}$ in Eq. (16.9b). We call the computed pair $(\mathbf{x}_\mathcal{F},\mathbf{y}_\mathcal{G})$ a complementary basic solution.

If a complementary basic solution $(\mathbf{x}_\mathcal{F},\mathbf{y}_\mathcal{G})$ satisfies $\mathbf{x}_\mathcal{F}\geq 0$ and $\mathbf{y}_\mathcal{G}\geq 0$, then it is called *feasible*. In this case, $\mathbf{x}=(\mathbf{x}_\mathcal{F},0)$ is the optimal solution of Eq. (16.7), and the algorithm terminates. Otherwise, a complementary basic solution $(\mathbf{x}_\mathcal{F},\mathbf{y}_\mathcal{G})$

is *infeasible*, and we need to update $\mathcal{F}$ and $\mathcal{G}$ by exchanging variables for which Eq. (16.8b) does not hold. Formally, we define the following index set:

$$\mathcal{H} = \{q \in \mathcal{F} : \mathbf{x}_q < 0\} \cup \{q \in \mathcal{G} : \mathbf{y}_q < 0\} \tag{16.10}$$

and choose a nonempty subset $\hat{\mathcal{H}} \subset \mathcal{H}$. Then, $\mathcal{F}$ and $\mathcal{G}$ are updated by the following rules:

$$\mathcal{F} = \left(\mathcal{F} - \hat{\mathcal{H}}\right) \cup \left(\hat{\mathcal{H}} \cap \mathcal{G}\right), \tag{16.11a}$$

$$\mathcal{G} = \left(\mathcal{G} - \hat{\mathcal{H}}\right) \cup \left(\hat{\mathcal{H}} \cap \mathcal{F}\right). \tag{16.11b}$$

The number of elements in set $\hat{\mathcal{H}}$, which we denote by $|\hat{\mathcal{H}}|$, represents how many variables are exchanged per iteration between $\mathcal{F}$ and $\mathcal{G}$. If $|\hat{\mathcal{H}}| > 1$, then an algorithm is called a block principal pivoting algorithm; if $|\hat{\mathcal{H}}| = 1$, then an algorithm is called a single principal pivoting algorithm. The active set algorithm can be understood as an instance of single principal pivoting algorithms. An algorithm repeats this procedure until the number of infeasible variables (i.e., $|\hat{\mathcal{H}}|$) becomes zero.

In order to speed up the search procedure, one usually uses $\hat{\mathcal{H}} = \mathcal{H}$, which we call the *full exchange rule*. The full exchange rule means that we exchange all variables of $\mathcal{F}$ and $\mathcal{G}$ that do not satisfy Eqs. (16.8a)–(16.8b), and the rule accelerates computation by reducing the number of iterations. However, contrary to the active set algorithm in which the variable to exchange is carefully selected to reduce the residual, the full exchange rule may lead to a cycle and fail to find an optimal solution although it occurs rarely. To ensure finite termination, we need to employ a backup rule, which uses the following exchange set for Eqs. (16.11a) and (16.11b):

$$\hat{\mathcal{H}} = \left\{q : q = \max\{q \in \mathcal{H}\}\right\}. \tag{16.12}$$

The backup rule, where only the infeasible variable with the largest index is exchanged, is a single principal pivoting rule. This simple exchange rule guarantees a finite termination: Assuming that matrix $\mathbf{V}$ has full column rank, the exchange rule in Eq. (16.12) returns the solution of Eqs. (16.8a)–(16.8c) in a finite number of iterations [12]. Combining the full exchange rule and the backup rule, the block principal pivoting method for Eq. (16.7) that terminates within a finite number of iterations is summarized in [12].

Now, let us move on to the multiple right-hand side case:

$$\min_{\mathbf{X} \geq 0} \|\mathbf{V}\mathbf{X} - \mathbf{W}\|_F^2, \tag{16.13}$$

where $\mathbf{V} \in \mathbb{R}^{P \times Q}$, $\mathbf{X} \in \mathbb{R}^{Q \times L}$ and $\mathbf{W} \in \mathbb{R}^{P \times L}$. One can solve Eq. (16.13) by separately solving NNLS problems for each right-hand side vector. Although this approach is possible, we will see that there exist efficient ways to accelerate the multiple right-hand side case employing two important improvements suggested in [17].

Observe that the sets $\mathcal{F}$ and $\mathcal{G}$ change over iterations, and Eqs. (16.9a) and (16.9b) has to be solved for varying $\mathcal{F}$ and $\mathcal{G}$ every time. The first improvement is based on the observation that matrix $\mathbf{V}$, which corresponds to $\mathbf{B}^{(n)}$ of Eq. (16.6),

Algorithm 16.1: Block principal pivoting algorithm for the NNLS with multiple right-hand side vectors. $\mathbf{x}_{\mathcal{F}_l}$ and $\mathbf{y}_{\mathcal{G}_l}$ represents the subsets of lth column of $\mathbf{X}$ and $\mathbf{Y}$ indexed by $\mathcal{F}_l$ and $\mathcal{G}_l$, respectively

Input: $\mathbf{V} \in \mathbb{R}^{P \times Q}$,$\mathbf{W} \in \mathbb{R}^{Q \times L}$

Output: $\mathbf{X}(\in \mathbb{R}^{Q \times L}) = \arg\min_{\mathbf{X} \geq 0} \|\mathbf{VX} - \mathbf{W}\|_F^2$

Compute $\mathbf{V}^T\mathbf{V}$ and $\mathbf{V}^T\mathbf{W}$.
Initialize $\mathcal{F}_l = \emptyset$ and $\mathcal{G}_l = \{1, \ldots, q\}$ for all $l \in \{1, \ldots, L\}$. Set $\mathbf{X} = 0$, $Y = -\mathbf{V}^T\mathbf{W}$, $\boldsymbol{\alpha}(\in \mathbb{R}^r) = 3$, and $\boldsymbol{\beta}(\in \mathbb{R}^r) = q + 1$.
Compute $\mathbf{x}_{\mathcal{F}_l}$ and $\mathbf{y}_{\mathcal{G}_l}$ for all $l \in \{1, \ldots, L\}$ by Eqs. (16.9a) and (16.9b) using column grouping.
while any $(\mathbf{x}_{\mathcal{F}_l}, \mathbf{y}_{\mathcal{G}_l})$ is infeasible **do**
    Find the indices of columns in which the solution is infeasible: $I = \{j : (\mathbf{x}_{\mathcal{F}_j}, \mathbf{y}_{\mathcal{G}_j}) \text{ is infeasible}\}$.
    Compute $\mathcal{H}_l$ for all $l \in I$ by Eq. (16.10).
    For all $l \in I$ with $|\mathcal{H}_l| < \beta_l$, set $\beta_l = |\mathcal{H}_l|$, $\alpha_l = 3$ and $\hat{\mathcal{H}}_l = \mathcal{H}_l$.
    For all $l \in I$ with $|\mathcal{H}_l| \geq \beta_l$ and $\alpha_l \geq 1$, set $\alpha_l = \alpha_l - 1$ and $\hat{\mathcal{H}}_l = \mathcal{H}_l$.
    For all $l \in I$ with $|\mathcal{H}_l| \geq \beta_l$ and $\alpha_l = 0$, set $\hat{\mathcal{H}}_l$ by Eq. (16.12).
    Update $\mathcal{F}_l$ and $\mathcal{G}_l$ for all $l \in I$ by Eqs. (16.11a)–(16.11b).
    Update $\mathbf{x}_{\mathcal{F}_l}$ and $\mathbf{y}_{\mathcal{G}_l}$ for all $l \in I$ by Eqs. (16.9a) and (16.9b) using column grouping.
end while

is typically very long and thin. In this case, constructing matrices $\mathbf{V}_{\mathcal{F}}^T\mathbf{V}_{\mathcal{F}}$, $\mathbf{V}_{\mathcal{F}}^T\mathbf{w}$, $\mathbf{V}_{\mathcal{G}}^T\mathbf{V}_{\mathcal{F}}$, and $\mathbf{V}_{\mathcal{G}}^T\mathbf{w}$ before solving Eqs. (16.9a) and (16.9b) is computationally very expensive. To ease this difficulty, $\mathbf{V}^T\mathbf{V}$ and $\mathbf{V}^T\mathbf{W}$ can be computed in the beginning and reused in later iterations. One can easily see that $\mathbf{V}_{\mathcal{F}}^T\mathbf{V}_{\mathcal{F}}$, $\mathbf{V}_{\mathcal{F}}^T\mathbf{w}_l$, $\mathbf{V}_{\mathcal{G}}^T\mathbf{V}_{\mathcal{F}}$, and $\mathbf{V}_{\mathcal{G}}^T\mathbf{w}_l$, $l \in \{1, \ldots, L\}$, can be directly retrieved as a submatrix of $\mathbf{V}^T\mathbf{V}$ or $\mathbf{V}^T\mathbf{W}$. Because the column size of $\mathbf{V}$ is small, storage needed for $\mathbf{V}^T\mathbf{V}$ and $\mathbf{V}^T\mathbf{W}$ is also small.

The second improvement involves exploiting common computations in solving Eq. (16.9a). Here we simultaneously run the block principal pivoting algorithm for multiple right-hand side vectors. At each iteration, we have index sets $\mathcal{F}_l$ and $\mathcal{G}_l$ for each column $l \in \{1, \ldots, L\}$, and we must compute $\mathbf{x}_{\mathcal{F}_l}$ and $\mathbf{y}_{\mathcal{G}_l}$ using Eqs. (16.9a) and (16.9b). The idea is to find groups of columns that share the same index sets $\mathcal{F}_l$ and $\mathcal{G}_l$. We reorder the columns with respect to these groups and solve Eqs. (16.9a) and (16.9b) for the columns in the same group. By doing so, we avoid repeated Cholesky factorization computations required for solving Eq. (16.9a). When matrix $\mathbf{X}$ is flat and wide, which is typically the case for $(\mathbf{A}^{(n)})^T$ in Eq. (16.6), more columns are likely to share their index sets $\mathcal{F}_l$ and $\mathcal{G}_l$, allowing bigger speed-up.

Incorporating these improvements, a full description of the block principal pivoting method for Eq. (16.13) is shown in Algorithm 16.1. Finite termination of Al-

gorithm 16.1 is achieved by controlling the number of infeasible variables using $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. For more details of how it is controlled, see [17, 18].

16.5 Regularized and Sparse NNCP

The ANLS framework described in Sect. 16.3 can be easily extended to formulations with regularization. In a general form, a regularized formulation appears as

$$\begin{aligned} \min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} \quad & \left\| \mathcal{X} - [\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}]\!] \right\|_F^2 + \sum_{n=1}^{N} \lambda_n \phi_n\left(\mathbf{A}^{(n)}\right), \\ \text{s.t.} \quad & \mathbf{A}^{(n)} \geq 0 \text{ for } n = 1, \cdots, N, \end{aligned} \tag{16.14}$$

where $\phi_n(\mathbf{A}^{(n)})$ represents a regularization term and $\lambda_n \geq 0$ is a parameter to be chosen. A commonly used regularization term is the Frobenius norm:

$$\phi_n\left(\mathbf{A}^{(n)}\right) = \left\| \mathbf{A}^{(n)} \right\|_F^2.$$

In this case, the subproblem for finding $\mathbf{A}^{(n)}$ is modified as

$$\begin{aligned} \min_{\mathbf{A}^{(n)}} \quad & \left\| \begin{pmatrix} \mathbf{B}^{(n)} \\ \sqrt{\lambda_n}\mathbf{I}_{K\times K} \end{pmatrix} \times \left(\mathbf{A}^{(n)}\right)^T - \left(\mathbf{X}^{(n)}\right)^T \right\|_F^2 \\ \text{s.t.} \quad & \mathbf{A}^{(n)} \geq 0, \end{aligned} \tag{16.15}$$

where $\mathbf{I}_{K\times K}$ is a $K \times K$ identity matrix. Observe that matrix $\begin{pmatrix} \mathbf{B}^{(n)} \\ \sqrt{\lambda_n}\mathbf{I}_{K\times K} \end{pmatrix}$ is always of full column rank; hence, when $\mathbf{B}^{(n)}$ is not necessarily of full column rank, the Frobenius norm regularization can be adopted to ensure that the NNLS subproblem is of full column rank, satisfying the requirement of the convergence property of a block-coordinate-descent method, mentioned in Sect. 16.3. In addition, the block principal pivoting method assumes that the matrix $\mathbf{V}$ in Eq. (16.13) is of full column rank, and the Frobenius norm regularization automatically satisfies this condition.

If it is desired to promote sparsity on factor matrix $\mathbf{A}^{(n)}$, l_1-norm regularization can be used:

$$\phi_n(\mathbf{A}^{(n)}) = \sum_{j=1}^{M_n} \left\| \left(\mathbf{A}^{(n)}\right)^T (:, j) \right\|_1^2,$$

where $(\mathbf{A}^{(n)})^T(:, j)$ represents the jth column of $(\mathbf{A}^{(n)})^T$. See [13, 16] for applications of this l_1-norm regularization in microarray data analysis and clustering. In this case, the subproblem for finding $\mathbf{A}^{(n)}$ is modified as

$$\begin{aligned} \min_{\mathbf{A}^{(n)}} \quad & \left\| \begin{pmatrix} \mathbf{B}^{(n)} \\ \sqrt{\lambda_n}\mathbf{1}_{1\times K} \end{pmatrix} \times (\mathbf{A}^{(n)})^T - (\mathbf{X}^{(n)})^T \right\|_F^2 \\ \text{s.t.} \quad & \mathbf{A}^{(n)} \geq 0, \end{aligned} \tag{16.16}$$

where $\mathbf{1}_{1\times K}$ is a row vector of ones. Regularization term $\phi_n(\cdot)$ can be separately chosen for each factor $\mathbf{A}^{(n)}$, and if necessary, both of the Frobenius norm and the l_1-norm may be used.

16.6 Implementation and Results

In this section, we describe the details of our implementation, data sets used, and comparison results. All experiments were executed in MATLAB on a Linux machine with a 2.66 GHz Intel Quad-core processor and 6 GB memory. The multi-threading option of MATLAB was disabled. In all the executions, all the algorithms were provided with the same initial values.

16.6.1 Algorithms for NNCP Used for Comparisons

The following algorithms for NNCP were included in our comparison.

1. (ANLS-BPP) ANLS with the block principal pivoting method proposed in this chapter
2. (ANLS-AS) ANLS with H. Kim and Park's active set method [15]
3. (HALS) Cichocki and Phan's hierarchical alternating least squares algorithm [7, 8]
4. (MU) Welling and Weber's multiplicative updating algorithm [29].

We implemented all algorithms in MATLAB. Besides above methods, we also have tested Friedlander and Hatz's two-metric projected gradient method [10] using their MATLAB code;[1] however, not only it was much slower than methods listed above, but it also required so much memory that we could not execute all comparison cases. We hence do not include the results of Friedlander and Hatz's method here. In all the algorithms, once we obtain factors $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\}$, they are used as initial values of the next iteration.

16.6.2 Data Sets

We have used three data sets for comparisons. The first data set include dense tensors using synthetically generated factors. For each of $K = 10$, 20, 60, and 120, we constructed $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ of size $300 \times K$ using random numbers from the uniform distribution over [0, 1]. Then, we randomly selected 50 percent of elements

[1] http://www.cs.ubc.ca/~mpf/2008-computing-nntf.html.

in $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ to make them zero. Finally, a three way tensor of size $300 \times 300 \times 300$ is constructed by $[\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!]$. Different tensors were created for different K values.

The second data set is a dense tensor obtained from Extended Yale Face Database B[2]. We used aligned and cropped images of size 168×192. From total 2424 images, we obtained a three-way tensor of size $168 \times 192 \times 2424$.

The third data set is a sparse tensor from NIPS conference papers.[3] This data set contains NIPS papers volume 0 to 12, and a tensor is constructed as a four-way tensor representing author×documents×term×year. By counting the occurrence of each entry, a sparse tensor of size $2037 \times 1740 \times 13649 \times 13$ was created.

16.6.3 Experimental Results

To observe the performance of several algorithms, at the end of each iteration we have recorded the relative objective value, $\|\mathcal{X} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\|_F / \|\mathcal{X}\|_F$. Time spent to compute the objective value is excluded from the execution time. One execution result involves relative objective values measured at discrete time points and appears as a piecewise-linear function. We averaged piecewise-linear functions from different random initializations to plot figures.

Results on the synthetic data set are shown in Fig. 16.1. This data set was synthetically created, and the value of global optimum is zero. From Fig. 16.1, it can be seen that ANLS-AS and ANLS-BPP performed the best among the algorithms we tested. The HALS method showed convergence within the time window we have observed, but the MU method was too slow to show convergence. ANLS-AS and ANLS-BPP showed almost the same performance although ANLS-BPP was slightly faster when $k = 120$. The difference between these two methods are better shown in next results.

Results on YaleB and NIPS data sets are shown in Fig. 16.2. Similarly to the results in Fig. 16.1, ANLS-AS and ANLS-BPP showed the best performance. In Fig. 16.2, it can be clearly observed that ANLS-BPP outperforms ANLS-AS for $k = 60$ and $k = 120$ cases. Such a difference demonstrates a difficulty of the active-set method: Since typically only one variable is exchanged between working sets, the active-set method is slow for a problem with a large number of variables. On the other hand, the block principal pivoting method quickly solves large problems by allowing exchanges of multiple variables between $\mathcal{F}$ and $\mathcal{G}$. The convergence of HALS and MU was slower than ANLS-AS and ANLS-BPP. Although the convergence of HALS was faster than MU in the YaleB data set, the initial convergence of MU was faster than HALS in the NIPS data set.

[2] http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html.

[3] http://www.cs.nyu.edu/~roweis/data.html.

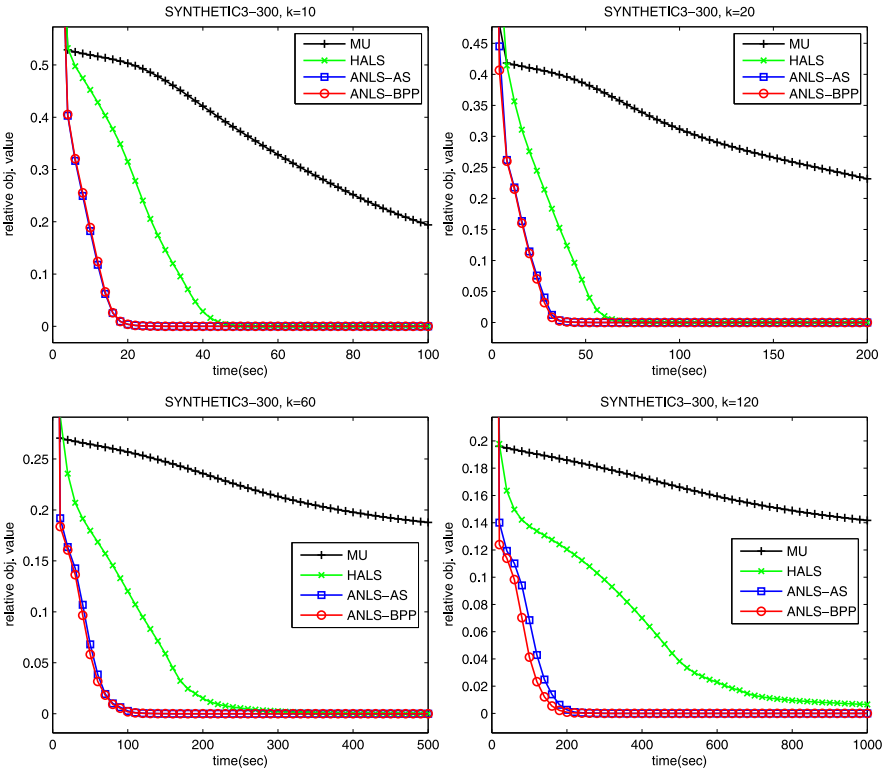

Fig. 16.1 Relative objective value ($\|\mathcal{X} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\|_F / \|\mathcal{X}\|_F$) vs. execution time on the synthetic tensors. Average results of 5 different random initializations are shown. *Top row*: $k = 10$ and $k = 20$, *bottom row*: $k = 60$ and $k = 120$

Lastly, we present more detailed information regarding the executions of ANLS-AS and ANLS-BPP in Fig. 16.3. In Fig. 16.1 and Fig. 16.2, we have observed that ANLS-BPP clearly outperforms ANLS-AS for large k's. Because both of the methods solve each NNLS subproblem exactly, solutions after each iteration from the two methods are the same up to numerical rounding errors. Hence, it suffices to compare the amount of time spent at each iteration. In Fig 16.3, we showed average execution time of each iteration of the two methods. It can be seen that the time required for ANLS-BPP is significantly shorter than the time required for ANLS-AS in early iterations, and their time requirements became gradually closer to each other. The types of NNLS problem in which ANLS-BPP accelerates ANLS-AS is the case that there is much difference in the zero and nonzero pattern between the initial value and the final solution of the NNLS problem. As iteration goes on, factors $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\}$ do not change much from one iteration to the next; hence there are little differences between the computational costs of the two methods.

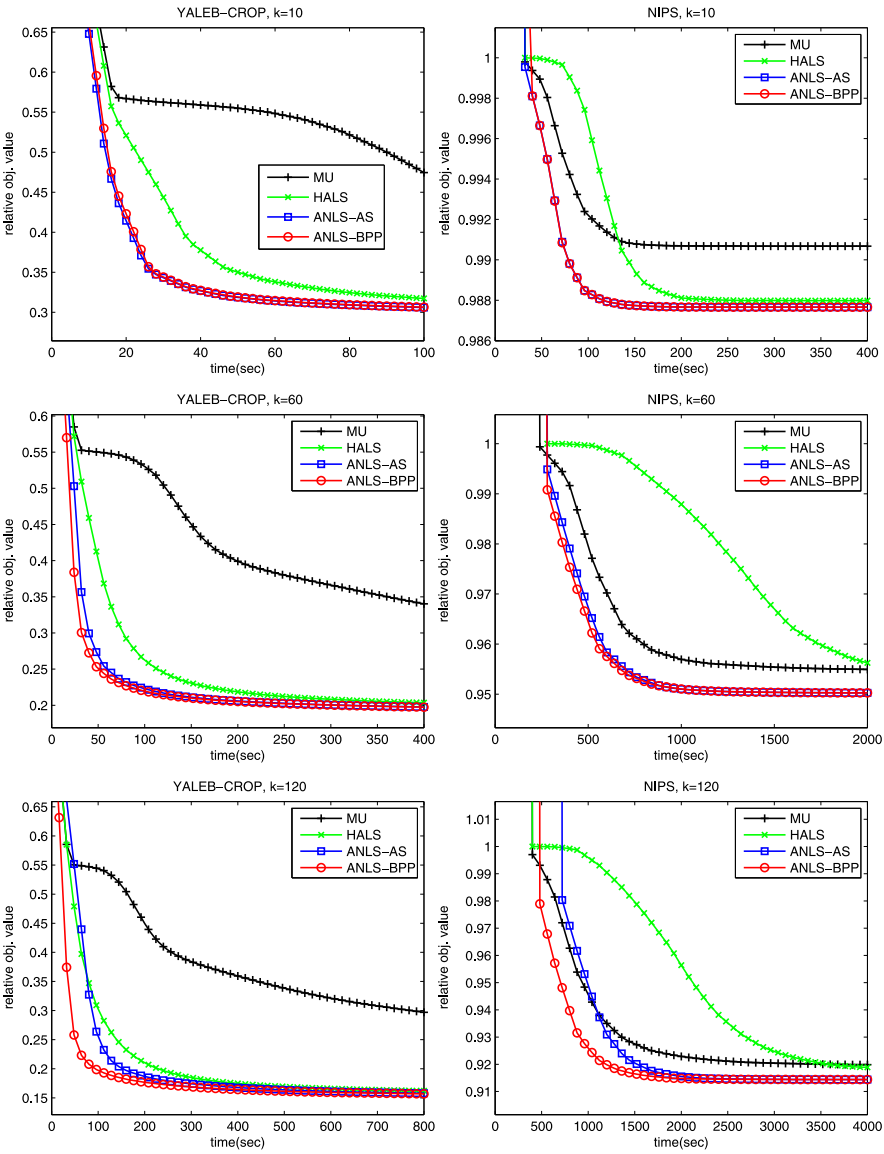

Fig. 16.2 Relative objective value ($\|\mathcal{X} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\|_F / \|\mathcal{X}\|_F$) vs. execution time on the YaleB and NIPS data sets. Average results of 5 different random initializations are shown. *Left*: NIPS data set, *right*: YaleB data set, *top row*: $k = 10$, *middle row*: $k = 60$, and *bottom row*: $k = 120$

16.7 Conclusions and Discussion

We have introduced an efficient algorithm for nonnegative CP (NNCP). The new method is based on the block principal pivoting method for the nonnegativity-

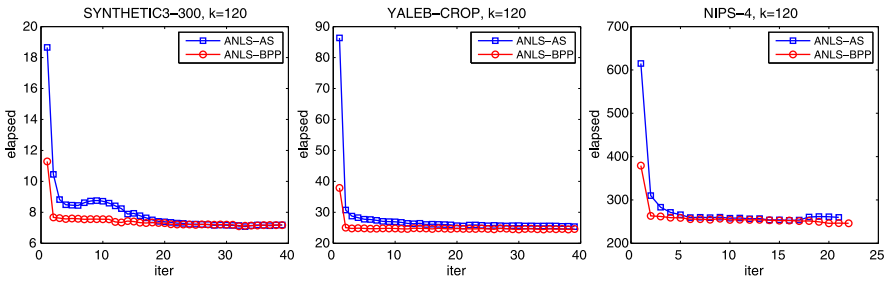

Fig. 16.3 Execution time of each iteration of the active set (ANLS-AS) and the block principal pivoting method (ANLS-BPP) for $k = 120$ cases of each data set. Average results of five different random initializations are shown. *Left*: synthetic data set, center: YaleB data set, *right*: NIPS data set

constrained least squares (NNLS) problems. The block principal pivoting method accelerates the classical active-set method by allowing exchanges of multiple variables per iteration. We have presented ideas for improving the block principal method for the NNLS problems with multiple right-hand sides. Computational comparisons showed the state-of-the-art performance of the proposed method for NNCP.

A drawback of an NNCP algorithm based on the active set or the block principal pivoting method is that the methods assume that the Khatri–Rao product in Eq. (16.5) is of full column rank for all $n \in \{1, \ldots, N\}$ throughout iterations. To alleviate this concern, as noted in Sect. 16.5, Frobenius norm-based regularization can be used to avoid rank-deficient cases. In practice, the algorithms performed well in our experiments without the regularization.

An interesting direction of future work is to investigate the conditions in which HALS performs better than the block principal pivoting method. In nonnegative matrix factorization, which can be considered as a special case of NNCP discussed in this chapter, we have observed that the HALS method converges very quickly [18]. In our results for NNCP in this chapter, however, HALS showed slower convergence than the block principal pivoting method.

Acknowledgements The work in this chapter was supported in part by the National Science Foundation grants CCF-0732318, CCF-0808863, and CCF-0956517. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Acar, E., Yener, B.: Unsupervised multiway data analysis: A literature survey. IEEE Trans. Knowl. Data Eng. **21**(1), 6–20 (2009)
2. Bader, B.W., Berry, M.W., Browne, M.: Discussion tracking in Enron email using PARAFAC. In: Survey of Text Mining II: Clustering, Classification, and Retrieval, pp. 147–163. Springer, Berlin (2008)
3. Bertsekas, D.P.: Nonlinear Programming. Scientific, Athena (1999)

4. Bro, R., De Jong, S.: A fast non-negativity-constrained least squares algorithm. J. Chem. **11**, 393–401 (1997)
5. Carroll, J.D., Chang, J.J.: Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. Psychometrika **35**(3), 283–319 (1970)
6. Carroll, J.D., Soete, G.D., Pruzansky, S.: Fitting of the latent class model via iteratively reweighted least squares CANDECOMP with nonnegativity constraints. In: Multiway data analysis, pp. 463–472. North-Holland, Amsterdam (1989). http://portal.acm.org/citation.cfm?id=120565.120614
7. Cichocki, A., Phan, A.H.: Fast local algorithms for large scale nonnegative matrix and tensor factorizations. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E92-A**(3), 708–721 (2009)
8. Cichocki, A., Zdunek, R., Amari, S.I.: Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization. In: Lecture Notes in Computer Science, vol. 4666, pp. 169–176. Springer, Berlin (2007)
9. FitzGerald, D., Cranitch, M., Coyle, E.: Non-negative tensor factorisation for sound source separation. In: Proceedings of the Irish Signals and Systems Conference (2005)
10. Friedlander, M.P., Hatz, K.: Computing nonnegative tensor factorizations. Comput. Optim. Appl. **23**(4), 631–647 (2008). doi:10.1080/10556780801996244
11. Harshman, R.A.: Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. In: UCLA Working Papers in Phonetics, vol. 16, pp. 1–84 (1970)
12. Júdice, J.J., Pires, F.M.: A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. Comput. Oper. Res. **21**(5), 587–596 (1994)
13. Kim, H., Park, H.: Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. Bioinformatics **23**(12), 1495–1502 (2007)
14. Kim, H., Park, H.: Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. SIAM J. Matrix Anal. Appl. **30**(2), 713–730 (2008). doi:10.1137/07069239X
15. Kim, H., Park, H., Eldén, L.: Non-negative tensor factorization based on alternating large-scale non-negativity-constrained least squares. In: Proceedings of IEEE 7th International Conference on Bioinformatics and Bioengineering (BIBE07), vol. 2, pp. 1147–1151 (2007)
16. Kim, J., Park, H.: Sparse nonnegative matrix factorization for clustering. Tech. rep., Georgia Institute of Technology Technical Report GT-CSE-08-01 (2008)
17. Kim, J., Park, H.: Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM), pp. 353–362 (2008)
18. Kim, J., Park, H.: Fast nonnegative matrix factorization: An active-set-like method and comparisons. SIAM J. Sci. Comput. **33**, 3261 (2011)
19. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
20. Lawson, C.L., Hanson, R.J.: Solving Least Squares Problems. Prentice Hall, New York (1974)
21. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755), 788–791 (1999)
22. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: Advances in Neural Information Processing Systems, vol. 13, pp. 556–562. MIT Press, Cambridge (2001)
23. Lim, L.H., Comon, P.: Nonnegative approximations of nonnegative tensors. J. Chem., **23**(7–8), 432–441 (2009)
24. Paatero, P.: A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis. Chemom. Intell. Lab. Syst. **38**(2), 223–242 (1997)
25. Paatero, P.: The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. J. Comput. Graph. Stat. **8**, 854–888 (1999)

26. Paatero, P., Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. EnvironMetrics **5**(1), 111–126 (1994)
27. Shashua, A., Hazan, T.: Non-negative tensor factorization with applications to statistics and computer vision. In: ICML '05: Proceedings of the 22nd International Conference on Machine Learning, pp. 792–799. ACM, New York (2005). doi: http://doi.acm.org/10.1145/1102351.1102451
28. Van Benthem, M.H., Keenan, M.R.: Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems. J. Chem. **18**, 441–450 (2004). doi:10.1002/cem.889
29. Welling, M., Weber, M.: Positive tensor factorization. Pattern Recognit. Lett. **22**(12), 1255–1261 (2001). doi:10.1016/S0167-8655(01)00070-8