DAMPED GAUSS-NEWTON ALGORITHM FOR NONNEGATIVE TUCKER DECOMPOSITION

Anh Huy Phan[†], *Petr Tichavský*[‡]* *and Andrzej Cichocki*[†]

[†]Brain Science Institute, RIKEN, Wakoshi, Japan
[‡]Institute of Information Theory and Automation, Prague, Czech Republic

ABSTRACT

Algorithms based on alternating optimization for nonnegative Tucker decompositions (NTD) such as ALS, multiplicative least squares, HALS have been confirmed effective and efficient. However, those algorithms often converge very slowly. To this end, we propose a novel algorithm for NTD using the Levenberg-Marquardt technique with fast computation method to construct the approximate Hessian and gradient without building up the large-scale Jacobian. The proposed algorithm has been verified to overwhelmingly outperform "state-of-the-art" NTD algorithms for difficult benchmarks, and application of face clustering.

***Index Terms*—** nonnegative Tucker decomposition, Gauss-Newton, Levenberg-Marquardt, low rank approximation, face clustering

1. INTRODUCTION

Tucker decomposition with nonnegative constraints has been found in many important applications such as pattern recognition, scenes classification, EEG analysis, BCI EEG motor imaginery [1, 2, 3, 4, 5], and can be formulated as follows, "Decompose a given nonnegative data tensor $\underline{\mathbf{Y}} \in \mathbb{R}_{+}^{I_1 \times I_2 \cdots \times I_N}$ into a set of N nonnegative factors $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_{R_n}^{(n)}] \in \mathbb{R}_{+}^{I_n \times R_n}$, $(n = 1, 2, \ldots, N)$ and a nonnegative core tensor $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$", that is,

$$\underline{\mathbf{Y}} \approx \underline{\mathbf{G}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} = \underline{\mathbf{G}} \times \{\mathbf{A}\} = \hat{\underline{\mathbf{Y}}}, \quad (1)$$

where "$\times_n$" denotes product of a tensor and a matrix along mode-n. Most NTD algorithms minimize the cost function

$$D(\underline{\mathbf{Y}}, \{\mathbf{A}^{(n)}\}, \underline{\mathbf{G}}) = \|\underline{\mathbf{Y}} - \underline{\mathbf{G}} \times \{\mathbf{A}\}\|_F^2, \quad (2)$$

via alternating optimization which often accompanies update rules with low computational cost, but face problems of slow convergence. The multiplicative algorithms [1, 2, 3] are based on minimization of the squared Euclidean distance (Frobenius norm) and the Kullback-Leibler divergence.

*The work of P. Tichavsky was supported by Ministry of Education, Youth and Sports of the Czech Republic through the project 1M0572 and by Grant Agency of the Czech Republic through the project 102/09/1278.

All-at-once algorithms which simultaneously update all the factors cope with such problems. For canonical polyadic decomposition (CP), and nonnegative tensor factorization (NTF), the damped Gauss-Newton (dGN) algorithm was first proposed by Paatero [6]. The Gauss-Newton algorithm can be derived from Newton's method, and has an at most quadratic rate of convergence. However, these methods also face similar problem with large-scale Jacobians and large-scale inverses of the Hessians. Recently, Tichavský and Koldovský [7] have proposed a novel method to compute inverse of approximate Hessian based on $3R^2 \times 3R^2$ dimensional matrices for 3-D tensor factorizations. The algorithm has been generalized and extended to arbitrary dimensional and complex-valued or nonnegative tensors in [8, 9]. For NTD, due to high computational cost of Kronecker products and consumption of extremely large temporary extra-storage, the dGN method has not yet been considered.

In this paper, we propose an all-at-once algorithm with low complexity for NTD based on the dGN iteration. A logarithmic barrier penalty term has been imposed on the cost function (2) to enforce nonnegativity constraints. The proposed algorithm is verified to overwhelmingly outperform "state-of-the-art" NTD algorithms for difficult benchmarks.

Hereafter, we shall denote the mode-n matricized version of a tensor $\underline{\mathbf{Y}}$ by $\mathbf{Y}_{(n)}$. $\mathbf{P}_n$ is a permutation matrix: $\text{vec}(\underline{\mathbf{Y}}) = \mathbf{P}_n \text{ vec}\big(\mathbf{Y}_{(n)}\big), n = 1, 2, \ldots, N$. Symbol "$\otimes$" denotes the Kronecker product, $\mathbf{A}^{\otimes} = \mathbf{A}^{(N)} \otimes \mathbf{A}^{(N-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}$ and $\mathbf{A}^{\otimes -n} = \bigotimes_{k \neq n} \mathbf{A}^{(k)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}$. Multiplication of a tensor with all but one mode is defined as $\underline{\mathbf{G}} \times_{-n} \{\mathbf{A}\} = \underline{\mathbf{G}} \times_1 \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}$. Mode-$n$ multiplication of a tensor $\underline{\mathbf{Y}}$ by a vector $\boldsymbol{a} \in \mathbb{R}^{I_n}$ along mode-n is denoted by $\underline{\mathbf{Z}} = \underline{\mathbf{Y}} \,\bar{\times}_n\, \boldsymbol{a} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$. The contracted product of two tensors is defined as in [10, 1].

2. DAMPED GAUSS-NEWTON ALGORITHM

The proposed algorithm minimizes the cost function (2) with a logarithmic penalty function to prevent factors $\mathbf{A}^{(n)}$ and the core tensor $\underline{\mathbf{G}}$ reaching zeros.

$$D_+ = D(\underline{\mathbf{Y}}, \{\mathbf{A}^{(n)}\}, \underline{\mathbf{G}}) - \alpha P_l(\{\mathbf{A}^{(n)}\}, \underline{\mathbf{G}}), \quad (3)$$

$$P_l = \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \sum_{r=1}^{R_n} \log(a_{i_n r}^{(n)}) + \sum_{\boldsymbol{r}=[r_1,r_2,\ldots,r_N]} \log(g_{\boldsymbol{r}}), \quad (4)$$

where $\alpha > 0$, $a_{i_n r}^{(n)}$ and $g_{\boldsymbol{r}} = g_{r_1 r_2 \cdots r_N}$ are elements of factors $\mathbf{A}^{(n)}$ and core tensor $\underline{\mathbf{G}}$, respectively. The update rule derived from (3) simultaneously updates all the factors $\mathbf{A}^{(n)}$ and the core tensor $\underline{\mathbf{G}}$ based on the damped GN iteration [6] given by

$$\boldsymbol{v} \leftarrow \boldsymbol{v} - (\mathbf{H} + \mu \mathbf{I})^{-1} \; \boldsymbol{g}, \quad (5)$$

where $\boldsymbol{v}^T = \left[\text{vec}\left(\mathbf{A}^{(1)}\right)^T, \cdots, \text{vec}\left(\mathbf{A}^{(N)}\right)^T, \text{vec}(\underline{\mathbf{G}})^T\right], \mu > 0$ is the damping parameter, and $\mathbf{I}$ is the identity matrix. The gradient $\boldsymbol{g}$ and the approximate Hessian $\mathbf{H}$ are given by

$$\boldsymbol{g} = \mathbf{J}^T(\hat{\boldsymbol{y}} - \boldsymbol{y}) - \alpha \, \frac{\partial P_l}{\partial \boldsymbol{v}} = \mathbf{J}^T(\hat{\boldsymbol{y}} - \boldsymbol{y}) - \alpha \, \boldsymbol{v}^{\bullet[-1]}, \quad (6)$$

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} - \alpha \, \frac{\partial^2 P_l}{\partial \boldsymbol{v}^2} = \mathbf{J}^T \mathbf{J} + \alpha \; \text{diag}\{\boldsymbol{v}^{\bullet[-2]}\}, \quad (7)$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & \mathbf{J}_2 & \cdots & \mathbf{J}_N & \mathbf{J}_{N+1} \end{bmatrix}, \quad (8)$$

$$\mathbf{J}_n = \begin{cases} \mathbf{P}_n^T \left(\mathbf{A}^{\otimes -n} \mathbf{G}_{(n)}{}^T \otimes \mathbf{I}_{I_n}\right), & n = 1, 2, \ldots, N, \\ \mathbf{A}^{\otimes}, & n = N+1, \end{cases} \quad (9)$$

where $\hat{\boldsymbol{y}} = \text{vec}\left(\hat{\underline{\mathbf{Y}}}\right)$, $\boldsymbol{y} = \text{vec}(\underline{\mathbf{Y}})$, and $[\boldsymbol{x}]^{\bullet[p]}$ denotes element-wise power, $\mathbf{I}_{I_n}$ is an $I_n \times I_n$ identity matrix. The Jacobian matrix $\mathbf{J} \in \mathbb{R}^{(\prod I_n) \times \left(\sum_{n=1}^{N} R_n I_n\right)}$ can be directly utilized in the learning rule (5). However, this demands high computational cost for construction of the approximate Hessian $(\mathbf{H} + \mu \mathbf{I})$ due to high computational cost of Kronecker products. In the sequence, we present more efficient computation methods for the learning rule (5).

2.1. Fast computation of the gradient $\boldsymbol{g}$

From (9), for $n = 1, 2, \ldots, N$, we have the following result

$$\begin{aligned} \mathbf{J}_n^T (\boldsymbol{y} - \hat{\boldsymbol{y}}) &= \left(\mathbf{A}^{\otimes -n\,T} \mathbf{G}_{(n)} \otimes \mathbf{I}_n\right) \mathbf{P}_n^T \, \text{vec}\left(\underline{\mathbf{Y}} - \hat{\underline{\mathbf{Y}}}\right) \\ &= \text{vec}\left(\left(\mathbf{Y}_{(n)} - \hat{\mathbf{Y}}_{(n)}\right) \; \mathbf{A}^{\otimes -n\,T} \, \mathbf{G}_{(n)}\right) \\ &= \text{vec}\left(\langle \underline{\mathbf{Y}} \times_{-n} \{\mathbf{A}^T\} - \underline{\mathbf{G}} \times_{-n} \{\mathbf{A}^T \mathbf{A}\}, \underline{\mathbf{G}} \rangle_{-n}\right), \end{aligned} \quad (10)$$

where $\langle \underline{\mathbf{Y}}, \hat{\underline{\mathbf{Y}}} \rangle_{-n}$ denotes contracted product between $\underline{\mathbf{Y}}, \hat{\underline{\mathbf{Y}}}$ along all their modes except mode n. Tensor products $\underline{\mathbf{Y}} \times_{-n} \{\mathbf{A}^T\} = \underline{\mathbf{Y}} \times_1 \mathbf{A}^{(1)\,T} \cdots \times_{n-1} \mathbf{A}^{(n-1)\,T} \times_{n+1} \mathbf{A}^{(n+1)\,T} \cdots \times_N \mathbf{A}^{(N)\,T}$, and $\underline{\mathbf{G}} \times_{-n} \{\mathbf{A}^T \mathbf{A}\} = \underline{\mathbf{G}} \times_1 \mathbf{A}^{(1)\,T} \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)\,T} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)\,T} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)\,T} \mathbf{A}^{(N)}$ can be calculated over a hierarchical stage of tensor-matrix multiplications. This avoids Kronecker products which are often computationally demanding, and consume significant temporary extra-storage. For example, Kronecker products $\mathbf{A}^{\otimes -n}$ produce large-scale matrices of size $\prod_{k \neq n} I_k \times \prod_{k \neq n} R_k$. Therefore, tensor products $\underline{\mathbf{Y}} \times_{-n} \{\mathbf{A}^T\}$ and $\underline{\mathbf{G}} \times_{-n} \{\mathbf{A}^T \mathbf{A}\}$ are much less computationally expensive than matrix products $\mathbf{Y}_{(n)} \; \mathbf{A}^{\otimes -n\,T}$, $\hat{\mathbf{Y}}_{(n)} \; \mathbf{A}^{\otimes -n\,T}$. Moreover, we don't need to build up the approximation tensor $\hat{\underline{\mathbf{Y}}}$ in (10).

Similarly, we have

$$\mathbf{J}_{N+1}^T (\boldsymbol{y} - \hat{\boldsymbol{y}}) = \text{vec}\left(\underline{\mathbf{Y}} \times \{\mathbf{A}^T\} - \underline{\mathbf{G}} \times \{\mathbf{A}^T \mathbf{A}\}\right). \quad (11)$$

From (10) and (11), we established a fast computation for the gradient $\boldsymbol{g}$ without computing the Jacobian $\mathbf{J}$.

2.2. Construction of approximate Hessian H

This section will present a low computational cost to build up the approximate Hessian given in (7). For simplicity, we construct the approximate Hessian $\mathbf{H} = \mathbf{J}^T \mathbf{J}$ without the regularization term which can be expressed as concatenation of $(N+1)^2$ block matrices $\mathbf{H}^{(n,m)} = \mathbf{H}^{(m,n)\,T}, m, n = 1, 2, \ldots, N+1$ as

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^{(1,1)} & \cdots & \mathbf{H}^{(1,N+1)} \\ \vdots & \ddots & \vdots \\ \mathbf{H}^{(N+1,1)} & \cdots & \mathbf{H}^{(N+1,N+1)} \end{bmatrix}. \quad (12)$$

2.2.1. *Off-Diagonal Block Matrices* $\mathbf{H}^{(n,m)} (n \neq m)$

Without loss of generality and because of symmetry $\mathbf{H}^{(n,m)} = \mathbf{H}^{(m,n)}, n \neq m$, we consider submatrices $\mathbf{H}^{(n,m)}$, for $1 \leqslant n < m \leqslant N$. From (9), for $n = 1, 2, \ldots, N$, we have

$$\begin{aligned} \mathbf{J}_n^T &= \mathbf{P}_{R_n, I_n} \left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)} \, \mathbf{A}^{\otimes -n T}\right) \mathbf{P}_{I_n, K_n} \mathbf{P}_n \\ &= \mathbf{P}_{R_n, I_n} \left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right) \left(\mathbf{I}_{I_n} \otimes \mathbf{A}^{\otimes -n T}\right) \mathbf{P}_{I_n, K_n} \mathbf{P}_n \\ &= \mathbf{P}_{R_n, I_n} \left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right) \left(\mathbf{P}_{I_n, R_{n+1:N}} \otimes \mathbf{I}_{R_{1:n-1}}\right) \\ &\quad \left(\bigotimes_{k=n+1}^{N} \mathbf{A}^{(k)\,T} \otimes \mathbf{I}_{I_n} \otimes \bigotimes_{k=1}^{n-1} \mathbf{A}^{(k)\,T}\right), \end{aligned} \quad (13)$$

where $K_n = \prod_{k \neq n} I_k$, $R_{n+1:N} = \prod_{k=n+1}^{N} R_k$, $R_{1:n-1} = \prod_{k=1}^{n-1} R_k$, $\mathbf{I}_J$ is an $J \times J$ identity matrix. Permutation matrices $\mathbf{P}_{R_n, I_n}$, $\mathbf{P}_{I_n, K_n}$ and $\mathbf{P}_{I_n, R_{n+1:N}}$ are defined as: $\text{vec}(\mathbf{X}_{I \times J}) = \mathbf{P}_{I,J} \; \text{vec}\left(\mathbf{X}_{I \times J}^T\right)$. Let $\mathbf{Q}_{(n)}$ denote the matrix product in (13)

$$\mathbf{Q}^{(n)} = \left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right) \left(\mathbf{P}_{I_n, R_{n+1:N}} \otimes \mathbf{I}_{R_{1:n-1}}\right) = \begin{bmatrix} \tilde{\mathbf{G}}_{(n)}^{(n_1)} \\ \vdots \\ \tilde{\mathbf{G}}_{(n)}^{(n_{I_n})} \end{bmatrix}, \quad (14)$$

where $\tilde{\mathbf{G}}_{(n)}^{(n_{i_n})}$ $(i_n = 1, 2, \ldots, I_n, n = 1, 2, \ldots, N)$ are the mode-n matricized versions of $(N+1)$-dimensional tensors $\tilde{\underline{\mathbf{G}}}^{(n_{i_n})}$ of size $R_1 \times R_2 \times \cdots \times R_n \times I_n \times R_{n+1} \times \cdots \times R_N$ whose subtensor obtained by fixing the $(n+1)$-th index to i_n is $\underline{\mathbf{G}}$, and other entries are zeros. That is the mode-$(n+1)$ matricization of $\tilde{\underline{\mathbf{G}}}^{(n_{i_n})}$ is a matrix whose i_n-th row is $\text{vec}(\underline{\mathbf{G}})^T$ and others are zeros

$$\tilde{\mathbf{G}}_{(n+1)}^{(n_{i_n})} = \begin{bmatrix} \mathbf{0}_{(i_n - 1) \times \prod_{k=1}^{N} R_k} \\ \text{vec}(\underline{\mathbf{G}})^T \\ \mathbf{0}_{(I_n - i_n) \times \prod_{k=1}^{N} R_k} \end{bmatrix}. \quad (15)$$

For each pair of indices (n,m), we define a set of $(N+1)$ matrices $\mathbf{B}^{(k)}$ given by

$$\mathbf{B}^{(k)} = \begin{cases} \mathbf{A}^{(k)\,T}\mathbf{A}^{(k)}, & 1 \leqslant k \leqslant n, \\ \mathbf{A}^{(n)\,T}, & k = n+1, \\ \mathbf{A}^{(k-1)\,T}\mathbf{A}^{(k-1)}, & n+2 \leqslant k \leqslant m, \\ \mathbf{A}^{(m)}, & k = m+1, \\ \mathbf{A}^{(k-1)\,T}\mathbf{A}^{(k-1)}, & m+2 \leqslant k \leqslant N+1. \end{cases}$$

Note that for simplicity, indices n, m are omitted in matrices $\mathbf{B}^{(k)}$. We have the following expression for $1 \leqslant n < m \leqslant N$

$$\begin{aligned} &\tilde{\mathbf{G}}^{(n_{i_n})}_{(n)} \left(\left(\bigotimes_{k=m+1}^{N} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \otimes \mathbf{A}^{(m)T} \otimes \left(\bigotimes_{k=n+1}^{m-1} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \otimes \right. \\ &\left. \mathbf{A}^{(n)} \otimes \left(\bigotimes_{k=1}^{n-1} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \right) \tilde{\mathbf{G}}^{(m_{i_m})\,T}_{(m)} = \tilde{\mathbf{G}}^{(n_{i_n})}_{(n)} \left(\{\mathbf{B}\}^{\otimes -n} \right)^T \tilde{\mathbf{G}}^{(m_{i_m})\,T}_{(m)} \\ &= \left(\underline{\tilde{\mathbf{G}}} \times_{-\binom{n}{n+1}_{m+1}} \{\mathbf{B}\} \times_{m+1} \mathbf{A}^{(m)} \times_{n+1} \mathbf{A}^{(n)\,T} \right)_{(n)} \tilde{\mathbf{G}}^{(m_{i_m})\,T}_{(m)} \\ &= \left(\underline{\mathbf{W}}^{(n,m)} \bar{\times}_{m+1} \boldsymbol{a}^{(m)T}_{i_m:} \times_{n+1} \boldsymbol{a}^{(n)\,T}_{i_n:} \right)_{(n)} \mathbf{G}^T_{(m)}, \end{aligned} \tag{16}$$

where $\underline{\mathbf{W}}^{(n,m)} = \underline{\hat{\mathbf{G}}} \times_{-\binom{n}{n+1}_{m+1}} \{\mathbf{B}\}$, and $\underline{\hat{\mathbf{G}}}$ is an augmented version of the tensor $\underline{\mathbf{G}}$ of size $R_1 \times \cdots \times R_n \times 1 \times R_{n+1} \times \cdots \times R_N$. From (9), (13), submatrices $\mathbf{H}^{(n,m)} = \mathbf{J}_n^T \mathbf{J}_m, n < m$ can be expressed as follows

$$\begin{aligned} &\mathbf{P}^T_{R_n,I_n} \mathbf{H}^{(n,m)} \mathbf{P}_{R_m,I_m} = \mathbf{Q}^{(n)} \{\mathbf{B}^{\otimes -n\,T}\} \mathbf{Q}^{(m)\,T} \\ &= \left[\tilde{\mathbf{G}}^{(n_{i_n})}_{(n)} \{\mathbf{B}^{\otimes -n\,T}\} \tilde{\mathbf{G}}^{(m_{i_m})\,T}_{(m)} \right]_{\substack{i_n=1,2,\ldots,I_n \\ i_m=1,2,\ldots,I_m}} \\ &= \left[\langle \underline{\mathbf{W}}^{(n,m)} \bar{\times}_{m+1} \boldsymbol{a}^{(m)T}_{i_m:} \times_{n+1} \boldsymbol{a}^{(n)\,T}_{i_n:}, \underline{\mathbf{G}} \rangle_{-n,-m} \right]_{\substack{i_n=1,2,\ldots,I_n \\ i_m=1,2,\ldots,I_m}}, \end{aligned} \tag{17}$$

where $\langle \underline{\mathbf{Y}}, \underline{\mathbf{G}} \rangle_{-n,-m}$ denotes the contracted product along all modes except mode-n for tensor $\underline{\mathbf{Y}}$, and except mode-m for the tensor $\underline{\mathbf{G}}$. Similarly, we straighforwardly express submatrices $\mathbf{H}^{(n,N+1)} = \mathbf{J}_n^T \mathbf{J}_{N+1}$ as follows

$$\mathbf{H}^{(n,N+1)} = \mathbf{P}_{R_n,I_n} \begin{bmatrix} \left(\underline{\mathbf{Z}}^{(n)} \times_{n+1} \boldsymbol{a}^{(n)\,T}_{1:} \right)_{(n)} \\ \vdots \\ \left(\underline{\mathbf{Z}}^{(n)} \times_{n+1} \boldsymbol{a}^{(n)\,T}_{i_n:} \right)_{(n)} \\ \vdots \\ \left(\underline{\mathbf{Z}}^{(n)} \times_{n+1} \boldsymbol{a}^{(n)\,T}_{I_n:} \right)_{(n)} \end{bmatrix}, \tag{18}$$

where $\underline{\mathbf{Z}}^{(n)} = \underline{\hat{\mathbf{G}}} \times_{-\binom{n,}{n+1}} \{\mathbf{A}^T\mathbf{A}\}$.

2.2.2. Diagonal Block Matrices $\mathbf{H}^{(n,n)}$

A diagonal block matrix $\mathbf{H}^{(n,n)} = \mathbf{J}_n^T \mathbf{J}_n$ can be expressed by

$$\mathbf{H}^{(n,n)} = \begin{cases} \langle \underline{\mathbf{G}} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \underline{\mathbf{G}} \rangle_{-n} \otimes \mathbf{I}_{I_n}, & n \neq N+1, \\ \{\mathbf{A}^T\,\mathbf{A}\}^{\otimes}, & n = N+1. \end{cases} \tag{19}$$

From (17)-(19), the approximate Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J}$ is fully expressed by products of tensors and contracted products. We note that matrices $\mathbf{A}^{(n)T}\mathbf{A}^{(n)}$ of size $R_n \times R_n$ are much smaller than matrices $\mathbf{A}^{(n)}$ due to $R_n \ll I_n$. This construction avoids computing large-scale Jacobian $\mathbf{J}$, and Kronecker products of large-scale matrices such as $\mathbf{A}^{\otimes -n}$ or $\mathbf{A}^{\otimes}$.

Finally, from Sections (2.1) and (2.2) we completely bypass computation of the Jacobian and establish a much faster computation for the approximate Hessian and the gradient than their conventional approach given in (7) and (6). Moreover, the proposed method does not demand significant temporary extra-storage. Selection of the regularization parameter α and the damping parameter μ can employ methods presented in [8, 9].

3. SIMULATIONS

3.1. Synthetic data

We compared performance of the algorithm LM_+ with the multiplicative LS (mLS) [2], HALS [4, 11]. Synthetic tensors $\underline{\mathbf{Y}}$ with $I_n = 100, N = 3, 4$ were composed from uniformly distributed random factors comprising $R = 5$ or $R = 3$ components. In some experiments, factors were forced to be sparse with density of 30%. Algorithms were initialized using the HOSVD algorithm [12], and stopped when difference between consecutive relative errors $\varepsilon = \frac{\|\underline{\mathbf{Y}} - \hat{\underline{\mathbf{Y}}}\|_F}{\|\underline{\mathbf{Y}}\|_F} \leqslant 10^{-8}$, or the maximum number of iterations was exceeded. Comparison of performances averaged over 100 runs is given in Table 1. Although the HALS algorithm achieved better performance than the mLS algorithm especially for sparse tensors, both theses algorithms demanded much more iterations than that of the LM_+ to explain the tensor with an acceptable fitness. Their relative errors for dense tensor decompositions were greater than 10^{-3}, and slightly better for sparse tensors. Whereas, the proposed algorithm achieved almost perfect performances with $\varepsilon \leqslant 10^{-5}$ after few iterations, even for large-scale tensor ($I_n = 100, N = 4$). In Fig. 1, we compared the relative errors as functions of iterations for one run of decomposition of a $100 \times 100 \times 100$ dimensional tensor. After few iterations to seek the damping parameter μ and the regularization parameter α, the LM_+ quickly explained the data tensor in ≈ 69 iterations. The mLS and HALS algorithms could not explain the benchmarks even if they were run for 5000 iterations.

3.2. Clustering of the ORL face database

This example considers the ORL face database [13] for clustering. The dataset consists of 400 faces for 40 subjects but we selected only 100 faces from the first 10 subjects. We constructed Gabor feature tensors of 8 orientations at 4 scales which were then down-sampled to form $16 \times 16 \times 8 \times 4$ dimensional tensor $\underline{\mathbf{Y}}^{(k)}, k = 1, 2, \ldots, 100$ for each face. That means we have a 5-D tensor $\underline{\mathbf{Y}}$. Nonnegative features were extracted for faces from Gabor tensors using NTD [5]. The data

Table 1. Performance comparison for various algorithms for decomposition of synthetic tensors.

	LS	HALS	LM_+
	$I_n = 50, N = 3, R = 5$		
Error	$(1.62 \pm 0.18)\ 10^{-2}$	$(1.15 \pm 0.12)\ 10^{-2}$	$(\mathbf{1.52 \pm 6.39})\ \mathbf{10^{-7}}$
No. iters	500	500	**47**
	$I_n = 100, N = 3, R = 5$		
Error	$(1.76 \pm 0.15)\ 10^{-2}$	$(1.28 \pm 0.12)\ 10^{-2}$	$(\mathbf{7.27 \pm 10.26})\ \mathbf{10^{-9}}$
No. iters	500	500	**69**
	$I_n = 100, N = 3, R = 5$, sparse factors		
Error	$(1.14 \pm 0.64)\ 10^{-2}$	$(1.28 \pm 10.99)\ 10^{-4}$	$(\mathbf{8.80 \pm 20.09})\ \mathbf{10^{-6}}$
No. iters	5000	2571	**66**
	$I_n = 100, N = 4, R = 3$		
Error	$(2.04 \pm 0.23)\ 10^{-2}$	$(2.01 \pm 0.83)\ 10^{-3}$	$(\mathbf{1.10 \pm 2.31})\ \mathbf{10^{-8}}$
No. iters	5000	5000	**55**
	$I_n = 100, N = 4, R = 3$, sparse factors		
Error	$(3.29 \pm 2.72)\ 10^{-3}$	$(8.41 \pm 45.26)\ 10^{-6}$	$(\mathbf{3.34 \pm 14.62})\ \mathbf{10^{-6}}$
No. iters	5000	2185	**55**

Table 2. Comparison of accuracies (Acc) and normalized mutual information (NMI) for various algorithms for Example 3.2.

Algorithm	36 features				72 features			
	Acc (%)	NMI	Error	No. iters	Acc (%)	NMI	Error	No. iters
mLS	91	89.65	0.4768	300	98	97.09	0.4768	300
HALS	**92**	**91.45**	**0.4745**	300	96.00	94.76	0.4369	300
LM_+	**92**	**91.45**	**0.4745**	**68**	**99**	**98.54**	**0.4368**	**86**

tensor $\underline{\mathbf{Y}}$ was decomposed along the first 4 modes to give core tensor's size of $3 \times 3 \times 2 \times 2 \times 100$. Hence, a face had 36 features compressed from 8192 Gabor features. Finally, the data was clustered using the K-means algorithm. The accuracy (%) and normalized mutual information (NMI) for algorithms are given in Table 2. The LM_+ algorithm achieved 92% accuracy. Increasing number of features to $72 = 3 \times 3 \times 4 \times 2$, our algorithm achieved 99% accuracy. For both cases, the obtained accuracies for the mLS algorithm were 91% and 98%, respectively. For the same dataset, the highest accuracy obtained by NTF algorithms were 94% [9]. The presented results also confirm the superiority of features extracted by NTD over features by NTF.

4. CONCLUSIONS

A robust dGN algorithm with fast method to construct the approximate Hessian and gradient has been proposed for NTD. Instead of computing the large-scale Jacobian, then building up the gradient and approximate Hessian, we establish direct expressions of gradient and approximate Hessian which avoid Kronecker products and extra-storage. The LM_+ algorithm has been verified and outperforms the multiplicative and (H)ALS algorithms. The proposed algorithm without the logarithmic barrier penalty term in the cost function also works well for general Tucker decomposition. Moreover, fast inverse of the approximate Hessian, similar to those in [7, 8, 9, 14] might be possible. However, due to limited space, analysis of such case has been omitted.

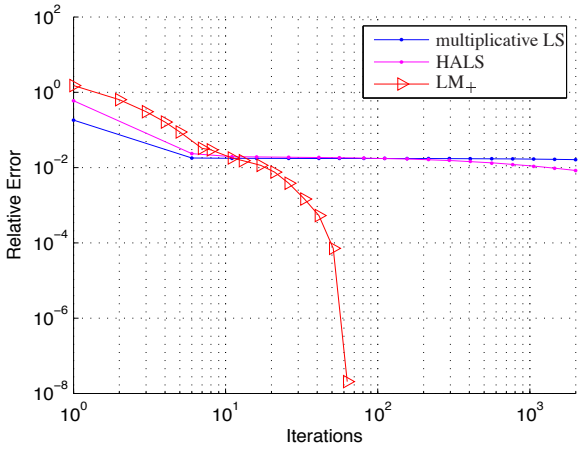

Fig. 1. Convergence of NTD algorithms for decomposition of 3-D synthetic tensor.

5. REFERENCES

[1] A. Cichocki, R Zdunek, A.-H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*, Wiley, Chichester, 2009.

[2] M. Mørup, L.K. Hansen, and S.M. Arnfred, "Algorithms for Sparse Nonnegative Tucker Decompositions," *Neural Computation*, vol. 20, pp. 2112–2131, 2008.

[3] Y.-D. Kim and S. Choi, "Nonnegative Tucker Decomposition," in *Proc. of Conf. Computer Vision and Pattern Recognition (CVPR-2007)*, Minneapolis, Minnesota, June 2007.

[4] A.-H. Phan and A. Cichocki, "Extended HALS algorithm for nonnegative Tucker decomposition and its applications for multiway analysis and classification," *Neurocomputing*, vol. In Press, Corrected Proof, pp. –, 2011.

[5] A.-H. Phan and A. Cichocki, "Tensor decompositions for feature extraction and classification of high dimensional datasets," *Nonlinear Theory and Its Applications, IEICE (invited paper)*, vol. 1, no. 1, pp. 37–68, 2010.

[6] P. Paatero, "A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis," *Chemometrics Intelligent Laboratory Systems*, vol. 38, no. 2, pp. 223–242, 1997.

[7] P. Tichavský and Z. Koldovský, "Simultaneous search for all modes in multilinear models," 2010, Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP10).

[8] A.-H. Phan, P. Tichavský, and A. Cichocki, "Low complexity damped Gauss-Newton algorithms for parallel factor analysis," *SIAM, SIMAX*, 2010, (under review).

[9] A.-H. Phan, P. Tichavský, and A. Cichocki, "Fast damped Gauss-Newton algorithm for sparse and nonnegative tensor factorization," in *ICASSP*, 2011.

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

[11] A.-H. Phan and A. Cichocki, "Local learning rules for nonnegative Tucker decomposition," *LNCS, ICONIP*, vol. 5863, pp. 538–545, 2009.

[12] L. De Lathauwer, B. de Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal of Matrix Analysis and Applications*, vol. 21, pp. 1253–1278, 2001.

[13] F. Samaria and A.C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of the Second IEEE Workshop on Applications of Computer Vision*, 1994.

[14] P. Tichavský and Z. Koldovsky, "Stability of candecomp-parafac tensor decomposition," in *ICASSP*, 2011.