# WEIGHTED KRYLOV-LEVENBERG-MARQUARDT METHOD FOR CANONICAL POLYADIC TENSOR DECOMPOSITION

*Petr Tichavský*[1], *Anh-Huy Phan*[2,3], *and Andrzej Cichocki*[2]

[1]The Czech Academy of Sciences, Institute of Information Theory and Automation, P.O.Box 18, 182 08 Prague 8, Czech Republic
[2]Skolkovo Institute of Science and Technology (Skoltech), Moscow 143026, Russia
[3]Tokyo University of Agriculture and Technology, Tokyo, 183-8538, Japan

## ABSTRACT

Weighted canonical polyadic (CP) tensor decomposition appears in a wide range of applications. A typical situation where the weighted decomposition is needed is when some tensor elements are unknown, and the task is to fill in the missing elements under the assumption that the tensor admits a low-rank model. The traditional methods for large-scale decomposition tasks are based on alternating least-squares methods or gradient methods. Second-order methods might have significantly better convergence, but so far they were used only on small tensors. The proposed Krylov-Levenberg-Marquardt method enables to do second-order-based iterations even in large-scale decomposition problems, with or without weights. We show in simulations that the proposed technique can outperform existing state-of-the-art algorithms in some scenarios.

***Index Terms***— Tensor decomposition, tensor completion, PARAFAC, CANDECOMP

## 1. INTRODUCTION

The workhorse algorithm for the Canonical Polyadic (CP) tensor decomposition is the Alternating Least Squares (ALS), see, e.g., [1, 2, 3, 4], often exhibits a slow convergence. A significantly faster convergence is obtained by second-order methods like Gauss-Newton or damped Gauss-Newton (Levenberg-Marquardt, LM) [5, 6], or nonlinear least squares (NLS) method [7, 8, 9].

If the tensors are huge in terms of their dimension, it is possible to decompose them through sampling, either random [11] or regular [10] or compression [12], and the problem is converted to the CP decomposition of a lower size.

A more difficult problem is the one of tensor completion: to fill missing entries of a partially known tensor under a low-rank constraint. This is a special case of a situation of finding a low-rank approximation of a given tensor. It might be that not all elements of the tensor are known or the tensor elements have assigned weights. See, e.g., [13, 14, 15, 16] and the references therein.

In some applications, some factor matrices in the CP decomposition might be constrained as nonnegative or to lie in some range. In that case, the Alternating Direction of Multipliers Method (ADMM) can be advised [17].

In all these classes of the problem, it might be convenient to replace the ALS-type or gradient-based algorithms by second-order-type methods to improve convergence and reduce the number of iterations needed to achieve convergence. Until recently, the second-order-type methods were considered as impractical due to large demands on computer memory and number of operations per iteration. These problems are alleviated in the Krylov-based methods [18], in particular, the Krylov-Levenberg-Marquardt (KLM) method. The method is based on the ability to quickly compute the product of the type $\mathbf{Hx}$, where $\mathbf{H}$ is the approximate Hessian of the CP decomposition problem and $\mathbf{x}$ is an arbitrary vector of appropriate dimension, without the need the Hessian to be computed explicitly. The size of the Hessian is $N \times N$ where $N$ is the number of parameters in the model, i.e., the number of elements of all factor matrices together. It can be several thousand or even million. While the number of elements in $\mathbf{H}$ is $N^2$, the elements need not be accessed, and the number of operations needed to compute the product $\mathbf{Hx}$ can be much smaller, usually of the order $O(N)$.

In [18], it was shown that the KLM algorithm allows achieving the CP decomposition with a lower *sensitivity* than traditional methods. The sensitivity is a quantity that characterizes how small Gaussian-distributed perturbation in elements of all the factor matrices translate in deviation of the tensor built from these matrices. The paper does not contain enough details about how the products $\mathbf{Hx}$ can be computed, in particular for the weighted CP decomposition. This paper aims to fill this gap in literature and provides the missing piece of information. Note that the KLM algorithm can be combined with the error preserving correction method of [21].

This work was supported by the Czech Science Foundation through Project No. 17-00902S. The work of A.-H. Phan and A. Cichocki was supported by the Ministry of Education and Science of the Russian Federation under Grant 14.756.31.0001.

The rest of the paper is organized as follows. Section 2 introduces the problem, Section 3 explains the construction of the Krylov subspace, and Section 4 presents the fast computation of the product $\mathbf{Hx}$ for an arbitrary vector $\mathbf{x}$. Section 5 contains numerical examples, and Section 6 concludes the paper.

## 2. WEIGHTED CP DECOMPOSITION

For simplicity, we formulate the problem for order-3 tensor only, adopting the $[[\cdot]]$ Kruskal tensor notation of Kolda and Bader [3].

Assume that a given tensor $\hat{\mathcal{T}}$ is fitted by a rank$-R$ tensor

$$\mathcal{T} = \mathcal{T}(\boldsymbol{\theta}) = [[\mathbf{A}, \mathbf{B}, \mathbf{C}]] \tag{1}$$

where the factor matrices, $\mathbf{A}, \mathbf{B}, \mathbf{C}$, have $R$ columns, and $\boldsymbol{\theta}$ is a vector which stacks their all elements as

$$\boldsymbol{\theta} = [\mathrm{vec}(\mathbf{A}); \mathrm{vec}(\mathbf{B}); \mathrm{vec}(\mathbf{C})] \, . \tag{2}$$

The separation by semicolons means stacking the vectors one above the other (like in Matlab).

Assume that we are given a weight tensor $\mathcal{W}$ of the same size as $\mathcal{T}$, having nonnegative elements. We wish to minimize the cost function

$$\varepsilon_{\mathcal{W}}(\boldsymbol{\theta}) = \|\mathcal{W}^{1/2} \star (\hat{\mathcal{T}} - \mathcal{T}(\boldsymbol{\theta}))\|_F^2 \, . \tag{3}$$

where $\| \cdot \|$ is the Frobenius norm, $\mathcal{W}^{1/2}$ is the elementwise square root of $\mathcal{W}$, and "$\star$" is the Hadamard (elementwise) product.

In the following, we denote a Jacobi matrix $\mathbf{J}$, an error gradient $\mathbf{g}$, and an approximate Hessian $\mathbf{H}$,

$$\mathbf{J} = \frac{\partial \mathrm{vec}(\mathcal{T})}{\partial \boldsymbol{\theta}} \tag{4}$$

$$\mathbf{g} = \mathbf{J}^T \mathbf{W} \mathrm{vec}(\mathcal{T} - \hat{\mathcal{T}}) \tag{5}$$

$$\mathbf{H} = \mathbf{J}^T \mathbf{W} \mathbf{J} \tag{6}$$

where $\mathbf{W} = \mathrm{diag}(\mathrm{vec}(\mathcal{W}))$. Note that some other authors call $\mathbf{H}$ *Gramian* [7]. In the Levenberg-Marquardt algorithm, one needs to compute products of the type $(\mathbf{H} + \mu\mathbf{I})^{-1}\mathbf{g}$ where $\mu$ is a parameter of regularization.

## 3. KRYLOV SUBSPACE TECHNIQUE

Assume that we can quickly compute the product $\mathbf{y} = \mathbf{Hx}$ for an arbitrary vector $\mathbf{x}$ of appropriate dimension, without the need of having the matrix $\mathbf{H}$ available directly.

The main idea of this work is to consider a low-rank approximation of the Hessian $\mathbf{H}$ of size $N \times N$,

$$\mathbf{H} \approx \mathbf{U}\mathbf{Z}\mathbf{U}^T \tag{7}$$

where $\mathbf{U}$ is a tall matrix of the size $N \times M$, $M \ll N$, and $\mathbf{Z}$ would be a symmetric $M \times M$ matrix. Then, the integer $M$ can be called rank of the approximation. Once we know the matrices $\mathbf{U}$ and $\mathbf{Z}$ in the approximation, we can compute the product $(\mathbf{H} + \mu\mathbf{I})^{-1}\mathbf{g}$ through the matrix inversion lemma,

$$(\mathbf{H} + \mu\mathbf{I})^{-1}\mathbf{g} \approx \frac{1}{\mu}\mathbf{g} - \frac{1}{\mu}\mathbf{U}(\mu\mathbf{Z}^{-1} + \mathbf{U}^T\mathbf{U})^{-1}(\mathbf{U}^T\mathbf{g}) \tag{8}$$

This computation requires inversion of two matrices of the size $M \times M$, which is not computationally demanding, if $M$ is not too big. The complexity is $O(NM+M^3) = O(IRM+M^3)$ operations. The approximation in (8) comes from (7), not from the matrix inversion lemma.

The Krylov method [19, 20] takes $\mathbf{U}$ as the orthogonal basis of the linear hull of

$$[\mathbf{g}, \mathbf{Hg}, \mathbf{H}^2\mathbf{g}, \ldots, \mathbf{H}^{M-1}\mathbf{g}] \, .$$

The basis $\mathbf{U}$ can proceed through the Gram-Schmidt orthogonalization process, see [18] for details. Then, $\mathbf{Z}$ is estimated as

$$\mathbf{Z} = \mathbf{U}^T\mathbf{H}\mathbf{U} \, . \tag{9}$$

Note that $\mathbf{Z}$ can be computed as a side product of the Gram-Schmidt orthogonalization.

In the next section, we show that the product $\mathbf{y} = \mathbf{Hx}$ can be computed with $O(IR^2)$ and $O(I_AI_BI_CR)$ operations in the case without and with weighting, respectively, where $I = I_A + I_B + I_C$. One iteration of KLM executes this computation $M$ times. The Gram-Schmidt orthogonalization requires additional $O(IMR^2)$ operations. Computing the error gradient $\mathbf{g}$ requires $O(I_AI_BI_CR)$ operations. The total complexity per iteration of KLM is then $O(I_AI_BI_CR + IM^2R + M^3)$ and $O(I_AI_BI_CMR + IM^2R + M^3)$ operations for the unweighted and weighted algorithms, respectively.

## 4. FAST COMPUTING OF $\mathbf{y} = \mathbf{Hx}$

We start with the *unweighted* CP decomposition, where $\mathcal{W}$ is a tensor of ones and $\mathbf{W}$ is the identity matrix. This part is well explained in [7, 8], we include it here for completeness.

Let $\mathbf{X}_A$ be an arbitrary matrix of the same size as $\mathbf{A}$, and similarly $\mathbf{X}_B$ and $\mathbf{X}_C$ be arbitrary matrices of the same size as $\mathbf{B}$ and $\mathbf{C}$, respectively, and let

$$\mathbf{x} = [\mathrm{vec}(\mathbf{X}_A); \mathrm{vec}(\mathbf{X}_B); \mathrm{vec}(\mathbf{X}_C)] \, .$$

Then, it can be shown that product of the approximate Hessian with a vector $\mathbf{x}$, i.e., $\mathbf{y} = \mathbf{J}^T\mathbf{J}\mathbf{x}$, can be expressed as

$$\mathbf{y} = \mathbf{J}^T\mathbf{J}\mathbf{x} = [\mathrm{vec}(\mathbf{Y}_A); \mathrm{vec}(\mathbf{Y}_B); \mathrm{vec}(\mathbf{Y}_C)] \tag{10}$$

where

$$\begin{aligned}
\mathbf{Y}_A &= \mathbf{X}_A((\mathbf{C}^T\mathbf{C}) \star (\mathbf{B}^T\mathbf{B})) \\
&\quad + \mathbf{A}((\mathbf{X}_B^T\mathbf{B}) \star (\mathbf{C}^T\mathbf{C}) + (\mathbf{X}_C^T\mathbf{C}) \star (\mathbf{B}^T\mathbf{B})) \\
\mathbf{Y}_B &= \mathbf{X}_B((\mathbf{A}^T\mathbf{A}) \star (\mathbf{C}^T\mathbf{C})) \\
&\quad + \mathbf{B}((\mathbf{X}_C^T\mathbf{C}) \star (\mathbf{A}^T\mathbf{A}) + (\mathbf{X}_A^T\mathbf{A}) \star (\mathbf{C}^T\mathbf{C})) \\
\mathbf{Y}_C &= \mathbf{X}_C((\mathbf{A}^T\mathbf{A}) \star (\mathbf{B}^T\mathbf{B})) \\
&\quad + \mathbf{C}((\mathbf{X}_B^T\mathbf{B}) \star (\mathbf{A}^T\mathbf{A}) + (\mathbf{X}_A^T\mathbf{A}) \star (\mathbf{B}^T\mathbf{B})) \, .
\end{aligned}$$

We can see that this computation of $\mathbf{y}$ involves only "small" matrices like $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{X}_A, \mathbf{X}_B$ and $\mathbf{X}_C$ and requires neither the Jacobian $\mathbf{J}$ nor the approximate Hessian $\mathbf{J}^T\mathbf{J}$.

Similarly, for order–4 tensors, where $\mathcal{T} = [[\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]]$, $\mathbf{x} = [\text{vec}(\mathbf{X}_A); \text{vec}(\mathbf{X}_B); \text{vec}(\mathbf{X}_C); \text{vec}(\mathbf{X}_D)]$, and $\mathbf{y} = [\text{vec}(\mathbf{Y}_A); \text{vec}(\mathbf{Y}_B); \text{vec}(\mathbf{Y}_C); \text{vec}(\mathbf{Y}_D)]$, we have

$$\begin{aligned}
\mathbf{Y}_A &= \mathbf{X}_A(\mathbf{B}_B \star \mathbf{C}_C \star \mathbf{D}_D) + \mathbf{A}((\mathbf{X}_B^T\mathbf{B}) \star \mathbf{C}_C \star \mathbf{D}_D \\
&\quad +(\mathbf{X}_C^T\mathbf{C}) \star \mathbf{B}_B \star \mathbf{D}_D + (\mathbf{X}_D^T\mathbf{D}) \star \mathbf{B}_B \star \mathbf{C}_C) \\
\mathbf{Y}_B &= \mathbf{X}_B(\mathbf{A}_A \star \mathbf{C}_C \star \mathbf{D}_D) + \mathbf{B}((\mathbf{X}_C^T\mathbf{C}) \star \mathbf{A}_A \star \mathbf{D}_D \\
&\quad +(\mathbf{X}_A^T\mathbf{A}) \star \mathbf{C}_C \star \mathbf{D}_D + (\mathbf{X}_D^T\mathbf{D}) \star \mathbf{A}_A \star \mathbf{C}_C) \\
\mathbf{Y}_C &= \mathbf{X}_C(\mathbf{A}_A \star \mathbf{B}_B \star \mathbf{D}_D) + \mathbf{C}((\mathbf{X}_B^T\mathbf{B}) \star \mathbf{A}_A \star \mathbf{D}_D \\
&\quad +(\mathbf{X}_A^T\mathbf{A}) \star \mathbf{B}_B \star \mathbf{D}_D + (\mathbf{X}_D^T\mathbf{D}) \star \mathbf{A}_A \star \mathbf{B}_B) \\
\mathbf{Y}_D &= \mathbf{X}_D(\mathbf{A}_A \star \mathbf{B}_B \star \mathbf{C}_C) + \mathbf{D}((\mathbf{X}_B^T\mathbf{B}) \star \mathbf{A}_A \star \mathbf{C}_C \\
&\quad (\mathbf{X}_A^T\mathbf{A}) \star \mathbf{B}_B \star \mathbf{C}_C + (\mathbf{X}_C^T\mathbf{C}) \star \mathbf{A}_A \star \mathbf{B}_B)
\end{aligned}$$

where $\mathbf{A}_A = \mathbf{A}^T\mathbf{A}$, $\mathbf{B}_B = \mathbf{B}^T\mathbf{B}$, $\mathbf{C}_C = \mathbf{C}^T\mathbf{C}$, and $\mathbf{D}_D = \mathbf{D}^T\mathbf{D}$.

In the case of weighted CP decomposition, we need to compute the product $\mathbf{y} = \mathbf{H}\mathbf{x}$ where $\mathbf{H} = \mathbf{J}^T\mathbf{W}\mathbf{J}$. The computation is more complex, but still, we do not need larger data structures than $\mathcal{T}$ and $\mathcal{W}$.

In the case of order–3 tensors, the product can be written in terms of a tensor $\mathcal{F}$ defined as

$$\mathcal{F} = \mathcal{W} \star ([[\mathbf{X}_A, \mathbf{B}, \mathbf{C}]] + [[\mathbf{A}, \mathbf{X}_B, \mathbf{C}]] + [[\mathbf{A}, \mathbf{B}, \mathbf{X}_C]])$$

as

$$\begin{aligned}
\mathbf{Y}_A &= \mathbf{F}_{(1)}(\mathbf{C} \odot \mathbf{B}) \\
\mathbf{Y}_B &= \mathbf{F}_{(2)}(\mathbf{C} \odot \mathbf{A}) \\
\mathbf{Y}_C &= \mathbf{F}_{(3)}(\mathbf{B} \odot \mathbf{A}) \,.
\end{aligned} \tag{11}$$

where $\mathbf{F}_{(1)}$, $\mathbf{F}_{(2)}$, $\mathbf{F}_{(3)}$ are the matricizations of $\mathcal{F}$ along the first, second, and third modes, respectively. Indeed, the definitions of $\mathbf{Y}_A$, $\mathbf{Y}_B$ and $\mathbf{Y}_C$ in (10) and (11) are equivalent, if $\mathcal{W}$ is composed of $1'$s only.

Similarly, for order–4 tensors we define

$$\begin{aligned}
\mathcal{F} &= \mathcal{W} \star ([[\mathbf{X}_A, \mathbf{B}, \mathbf{C}, \mathbf{D}]] + [[\mathbf{A}, \mathbf{X}_B, \mathbf{C}, \mathbf{D}]] \\
&\quad +[[\mathbf{A}, \mathbf{B}, \mathbf{X}_C, \mathbf{D}]] + [[\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{X}_D]])
\end{aligned}$$

and the matrices $\mathbf{Y}_A$, $\mathbf{Y}_B$, $\mathbf{Y}_C$, and $\mathbf{Y}_D$ are expressed as

$$\begin{aligned}
\mathbf{Y}_A &= \mathbf{F}_{(1)}(\mathbf{D} \odot \mathbf{C} \odot \mathbf{B}) \\
\mathbf{Y}_B &= \mathbf{F}_{(2)}(\mathbf{D} \odot \mathbf{C} \odot \mathbf{A}) \\
\mathbf{Y}_C &= \mathbf{F}_{(3)}(\mathbf{D} \odot \mathbf{B} \odot \mathbf{A}) \\
\mathbf{Y}_D &= \mathbf{F}_{(4)}(\mathbf{C} \odot \mathbf{B} \odot \mathbf{A}) \,.
\end{aligned} \tag{12}$$

## 5. SIMULATIONS

**Example 1.** We consider an artificial tensor $\mathcal{T} = [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]$ of size $50 \times 80 \times 90$ and rank $R = 10$. The factor matrices of the tensor are defined in Matlab notation as

$$\begin{aligned}
\mathbf{A} &= 0.9 * \mathbf{A}_0(:, 1) * ones(1, R) + 0.1 * \mathbf{A}_0 \\
\mathbf{B} &= 0.9 * \mathbf{B}_0(:, 1) * ones(1, R) + 0.1 * \mathbf{B}_0 \\
\mathbf{C} &= 0.9 * \mathbf{C}_0(:, 1) * ones(1, R) + 0.1 * \mathbf{C}_0
\end{aligned}$$

where $\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0$ are random matrices with orthogonal columns generated as $\mathbf{A}_0 = orth(randn(50, R))$, $\mathbf{B}_0 = orth(randn(80, R))$, and $\mathbf{C}_0 = orth(randn(90, R)$, respectively. Thanks to this construction, all the three factor matrices have moderately collinear columns, which makes the decomposition harder.

Finally, we add a small noise to the tensor by putting $\hat{\mathcal{T}} = \mathcal{T} + 0.01\mathcal{N}$ where $\mathcal{N}$ is generated as $\mathcal{N} = randn(50, 80, 90)$. Among elements of $\hat{\mathcal{T}}$ we select $10\%$ which will be available to the estimators, and the remaining $90\%$ will be hidden but used for evaluation purposes. In Matlab notation, the weight tensor is given as $\mathcal{W} = double(rand(50, 80, 90) < 0.1)$.

We test five algorithms in our simulation study: the weighted KLM with parameters $M = 10, 20$ and 50, NLS algorithm of Tensorlab (in the input tensor, the unavailable elements are changed to NaN), and the PARAFAC-ALS algorithm of Rasmus Bro [2]. We admit that Tensorlab and PARAFAC-ALS are not fully relevant for this test, because they do not use any constraint to prevent overfitting. We wished to compare with the recent algorithm TREL-1 [14] as well, but its code was not available. In Figure 1, we plot the typical learning curves of KLM and NLS. PARAFAC-ALS was not shown, because it does not have the learning curve as standard output. We can see how the sum of squared fitting errors converges to the final value, as a function of time. The number of iterations of PARAFAC-ALS is set to 1000 and its computational time is 75s. Usually, in less difficult scenarios, NLS is the fastest algorithm of all.

Table 2 presents the normalized fitting errors of the algorithms (cost function) $E_i$ and the normalized error of the unobserved entries,

$$E_i = \|\hat{\mathcal{T}}(\mathcal{W}) - \mathcal{T}_i(\mathcal{W})\|_F^2 / \|\hat{\mathcal{T}}(\mathcal{W})\|_F^2 \tag{13}$$

$$\bar{E}_i = \|\mathcal{T}(\bar{\mathcal{W}}) - \mathcal{T}_i(\bar{\mathcal{W}})\|_F^2 / \|\mathcal{T}(\bar{\mathcal{W}})\|_F^2 \tag{14}$$

where $\hat{\mathcal{T}}$ and $\mathcal{T}$ is the noisy and noiseless tensors, $\mathcal{T}_i$ stands for the estimators, $\mathcal{W}$ denotes the tensor elements available to the estimators, and $\bar{\mathcal{W}}$ is its complement. The last row shows sensitivity of the estimates provided by the algorithms (after balancing norms of the factors),

$$S_i = \sum_{r=1}^{R} \|\mathbf{a}_r^{(i)}\|^2(\|\mathbf{b}_r^{(i)}\|^2 + \|\mathbf{c}_r^{(i)}\|^2) + \|\mathbf{b}_r^{(i)}\|^2\|\mathbf{c}_r^{(i)}\|^2 \tag{15}$$

where $\{\mathbf{a}_r^{(i)}\}$, $\{\mathbf{b}_r^{(i)}\}$,$\{\mathbf{c}_r^{(i)}\}$ are columns of the estimated factor matrices. If the task is to estimate the missing tensor elements through the low-rank assumption, the most successful algorithm was KLM-20. The worst estimate is provided by

| alg. | KLM-10 | KLM-20 | KLM-50 | NLS | Bro |
|---|---|---|---|---|---|
| $E_i$ | 0.8675 | 0.3620 | 0.3448 | 0.3614 | **0.3237** |
| $\bar{E}_i$ | 1.2661 | **0.0421** | 0.0683 | 196.2157 | 0.1175 |
| $S_i$ | 48.89 | 48.19 | 52.17 | **2931.0** | 65.48 |

**Table 1**: Fitting errors of the algorithms and sensitivity of the decomposition.

| alg. | KLM-10 | KLM-20 | KLM-50 | NLS | Bro |
|---|---|---|---|---|---|
| $E_i$ | 0.0356 | 0.0355 | 0.0355 | 5.1487 | 0.0314 |
| $\bar{E}_i$ | **0.0002** | **0.0002** | **0.0002** | 0.8889 | 0.0009 |
| $S_i$ | 37.4 | 38.9 | 38.6 | **1689.0** | 47.41 |

**Table 2**: Fitting errors of the algorithms and sums of sensitivity of the decomposition for the example with a lower noise level.

NLS. The last line shows the reason. NLS produces the estimate with the highest sensitivity. Indeed, the sensitivity of the NLS can be reduced by considering an option with L1 or L2 regularization, which is not included in the current software. L1 regularization is used in TREL1 [14].

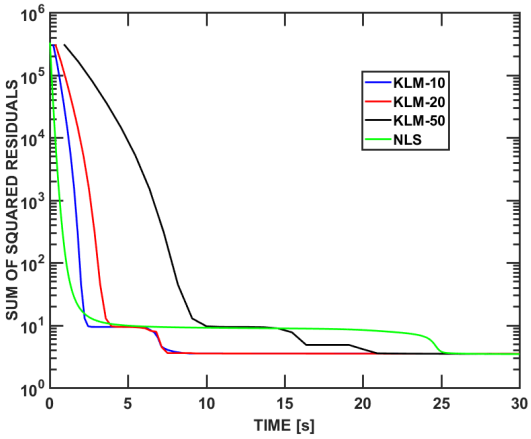


**Fig. 1**: Typical learning curves of KLM with $M = 10, 20, 50$ and NLS (Tensorlab) for Example 1.

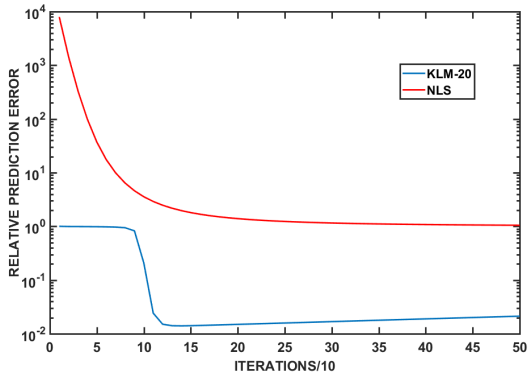


**Fig. 2**: Relative prediction errors $\bar{E}_i$ of KLM-20 and NLS (Tensorlab) versus the iteration number.

A natural explanation of the observed behavior of the algorithm would be a hypothesis of over-fitting. One may think the algorithm NLS was let run too long. To investigate this issue, we repeat the experiment, but stop the algorithms (here NLS and KLM-20) every 10 iterations, and measure the relative squared prediction error of the hidden tensor elements as a function of iteration number. The results are shown in Figure 2. We can see that KLM-20 achieves its minimum prediction error at approximately 150 iterations. Then, the error tends slightly to increase: there is some over-fitting if the iteration number is too high, but it is not very crude. On the other hand, NLS does not exhibit any over-fitting, and the results are constantly bad.

**Example 2.** We repeat the previous experiment but the additive noise has $10\times$ lower amplitude, i.e., $\hat{\mathcal{T}} = \mathcal{T} + 0.001\mathcal{N}$. In this case, we can see that NLS of Tensorlab fails to converge properly, see Figure 3. The three variants of KLM work almost equally well. They outperform PARAFAC-ALS in prediction error, also.

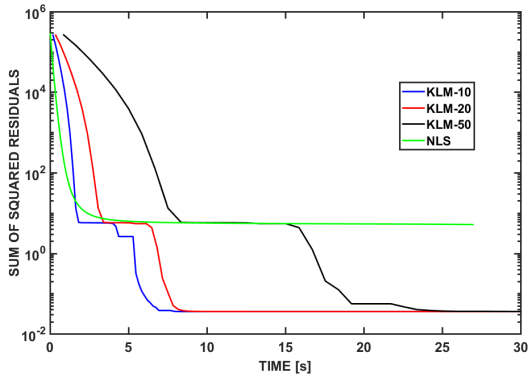


**Fig. 3**: Typical learning curves of KLM with $M = 10, 20, 50$ and NLS (Tensorlab) for Example 2.

## 6. CONCLUSIONS

The recent KLM algorithm was extended to the weighted CP tensor decomposition. The algorithm appears to produce good results even in situations where other methods fail. In general, it appears to produce decomposition with lower sensitivity. The high sensitivity of the decomposition may indicate over-fitting. Matlab code of the respective algorithms, KLM and weighted KLM (WKLM) has been posted at https://github.com/Tichavsky/tensor-decomposition.

## 7. REFERENCES


[1] 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.

[2] R. Bro, *Multi-way analysis in the food industry: models, algorithms, and applications*, Universiteit van Amsterdam, 1998.

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

[4] A.H. Phan, P. Tichavský and A. Cichocki, "Fast Alternating LS Algorithms for High Order CANDECOMP/-PARAFAC Tensor Factorizations", *IEEE Tr. Signal Processing*, Vol. 61, No. 19, October 2013, pp. 4834–4846.

[5] A.-H Phan, P. Tichavský and A. Cichocki, "Low Complexity Damped Gauss-Newton Algorithms for Parallel Factor Analysis", *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 1, pp. 126–147, January 2013.

[6] P. Tichavský, A.H. Phan, and A. Cichocki, "A further improvement of a fast damped Gauss–Newton algorithm for CANDECOMP-PARAFAC tensor decomposition", *Proc. ICASSP 2013*, Vancouver, Canada, May 2013, pp. 5964–5968.

[7] L. Sorber, M. Van Barel and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: canonical polyadic decomposition, decomposition in rank(Lr,Lr,1) terms, and a new generalization", *SIAM Journal on Optimization* Vol. 23, No. 2, pp. 695-720, 2013.

[8] N. Vervliet and L. De Lathauwer, "Numerical optimization based algorithms for data fusion, in *Data Fusion Methodology and Applications*, M. Cocchi, Ed., Elsevier, 2019.

[9] N. Vervliet, O. Debals, et al., "Tensorlab 3.0", Available online at https://www.tensorlab.net, Mar. 2016

[10] C. I. Kanatsoulis, N. D. Sidiropoulos, "Large-scale Canonical Polyadic Decomposition via Regular Tensor Sampling", *Proc. EUSIPCO 2019*, pp. 1–6, 2019.

[11] N. Vervliet and L. De Lathauwer, A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors, *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 284-295, 2016.

[12] N. D. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition," *IEEE Signal Processing Magazine*, vol.31, no.5, pp. 57-70, 2014.

[13] Y. Liu, F. Shang, L. Jiao, J. Cheng, and H. Cheng, "Trace Norm Regularized CANDECOMP/PARAFAC Decomposition With Missing Data", *IEEE Tran. Cybernetics*, vol. 45, no. 11, pp. 2437–2448, November 2015.

[14] Q. Shi, H. Lu, and Y.-M. Cheung, "Rank-One Matrix Completion with Automatic Rank Estimation via L1-Norm Regularization", *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, Vol. 29, no.10, pp. 4744–4757, October 2018.

[15] Q. Shi, Y.-M Cheung, and H. Lu, "Feature Extraction for Incomplete Data Via Low-Rank Tensor Decomposition With Feature Regularization", *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, Vol. 30, no.6, pp. 1803–1817, June 2019.

[16] Q. Yao, J.T. Kwok, and Bo Han, "Efficient Nonconvex Regularized Tensor Completion with Structure-aware Proximal Iterations", *Proc. of the 36th International Conference on Machine Learning, PMLR* 97, pp. 7035-7044, 2019. arXiv:1807.08725v3 [cs.LG] 23 Jan 2019.

[17] 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, Oct 2015.

[18] P. Tichavský, A.H. Phan, and A. Cichocki, "Sensitivity in Tensor Decomposition" *IEEE Signal Processing Letters*, vol.26, no.11, pp. 1653–1657, November 2019.

[19] A.N. Krylov, "On the Numerical Solution of Equation by Which are Determined in Technical Problems the Frequencies of Small Vibrations of Material Systems". Izvestija AN SSSR (News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk, 1931, VII, Nr.4, 491–539 (in Russian).

[20] J. Liesen and Z. Strakos, *Krylov Subspace Methods: Principles and Analysis*, Oxford Science Publications, 2012.

[21] A.H. Phan, P. Tichavský, and A. Cichocki, "Error Preserving Correction: A Method for CP Decomposition at a Target Error Bound", *IEEE Transactions on Signal Processing*, vol.67, 5 (2019), p. 1175–1190.