Computing the polyadic decomposition of nonnegative third order tensors

Jean-Philip Royer[a,b], Nadège Thirion-Moreau[a,*], Pierre Comon[b]

[a] *LSEET, UMR CNRS 6017, Université du Sud Toulon Var, F-83957, La Garde Cédex, France, Tel: +33-4-9414 2456/2671*

[b] *I3S, Algorithmes/Euclide-B, 2000 route des Lucioles, BP 121, F-06903, Sophia Antipolis Cedex, Tel/Fax: +33-4-9294 2717/2896*

Abstract

Computing the minimal polyadic decomposition (also often referred to as canonical decomposition, or sometimes Parafac) amounts to finding the global minimum of a coercive polynomial in many variables. In the case of arrays with nonnegative entries, the low-rank approximation problem is well posed. In addition, due to the large dimension of the problem, the decomposition can be rather efficiently calculated with the help of preconditioned nonlinear conjugate gradient algorithms, as subsequently shown, if equipped with an algebraic calculation of the globally optimal stepsize in low dimension. Other algorithms are also studied (gradient and quasi-Newton approaches) for comparisons. Two versions of each algorithm are considered: the Enhanced Line Search version (ELS), and the backtracking version alternating with ELS. Computer simulations are provided and demonstrate the good behavior of these algorithms dedicated to nonnegative arrays, compared to others put forward in the literature. Finally, applications in the context of data analysis illustrate various algorithms. The main advantage of the suggested approach is to explicitly take into account the nonnegative nature of the loading matrices in the problem parameterization, instead of enforcing positive entries by projection. According to the experiments we have run, such an approach also happens to be more robust with respect to possible modeling errors.

Keywords: Data analysis; Nonlinear conjugate gradient; preconditioning; Quasi-Newton; BFGS; DFP; nonnegative; 3-way; array; tensor; factorization; multi-way; Canonical Polyadic decomposition; CP decomposition; Canonical decomposition; CanDecomp; Parafac; Chemometrics; Data mining; Fluorescence

[☆] This work has been supported by a "PRES euro-méditerranéen" grant.

[*] Corresponding author.

Email addresses: jph.royer@gmail.com (Jean-Philip Royer), thirion@univ-tln.fr (Nadège Thirion-Moreau), pcomon@unice.fr (Pierre Comon)

1. Introduction

The minimal polyadic decomposition of a tensor, sometimes referred to as "Canonical Polyadic" (CP), is also called "CanDecomp", "CanD", or "Parafac". This decomposition, whose definition is recalled in Section 2, turns out to be very useful in a wide panel of applications; see e.g. [6, 7, 13, 27] and references therein. However, several difficulties arise when the CP needs to be computed. First, even if an exact fit exists with a known number of terms, the calculation of the CP consists of finding the zeros of a polynomial of degree six or larger, in a very large number of variables. This problem is numerically very difficult to solve, even if the number of zeros is finite. Second, if the model is subject to errors, an approximate fit is wished to be computed. However, it is now well known that a best approximate may not always exist [23, 15, 7]. Third, in several applications such as hyperspectral imaging or chemometrics, the loading matrices need to be constrained to be real and nonnegative [5, 27]. We shall subsequently concentrate on this framework. Fortunately, one advantage of the latter constraint is that the approximation problem becomes well posed [15]. Lastly, a recent book has been even dedicated to this particular problem [5].

Numerical algorithms are provided in the present paper, and are based on preconditioned non-linear conjugate gradient, well matched to large dimensions, combined with a global search in a one-dimensional subspace. The latter combination permits to escape from local minima. Other algorithms are also studied (gradient and quasi-Newton approaches, for the purpose of comparison). Note that a nonlinear conjugate gradient optimization technique has already been suggested in [22] but with a simple version of preconditioning (by a diagonal matrix).

The article is organized as follows. After a brief introduction, Section 2 starts with some definitions and properties of third order tensors. The problem of the polyadic decomposition of 3-way arrays is then stated and existing standard algorithms are pointed out. Section 3 is dedicated to nonnegative 3-way array factorization. The cost function we suggest to use is introduced, and basic quantities such as gradient matrices are then calculated. In Section 4, the preconditioned non-linear conjugate gradient approach is presented as well as three other approaches: gradient, quasi-Newton, and non-linear conjugate gradient approaches without preconditioning. With regard to the choice of the step size, two different strategies are studied: a global search via Enhanced Line Search (ELS) and backtracking alternating with ELS. Computer simulations are provided to illustrate the effectiveness of the proposed algorithms, and to compare them with other algorithms, which are more standard in the literature. In Section 5, we show the usefulness of these algorithms, and explain how they can be applied in Data Analysis. Finally, a conclusion is drawn in Section 6.

2. Problem statement

2.1. Notation

The outer (tensor) product between two tensors $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ and $\mathbf{Y} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_M}$ is denoted by $\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N \times J_1 \times J_2 \times \ldots \times J_M}$ and defined by $z_{i_1 i_2 \ldots i_N j_1 j_2 \ldots j_M} = x_{i_1 i_2 \ldots i_N} y_{j_1 j_2 \ldots j_M}$.

Denote by $(\cdot)^T$ matrix transposition. As special cases, the outer product between two vectors $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$ yields a rank-one matrix $\mathbf{C} = \mathbf{a} \circledast \mathbf{b} = \mathbf{a}\mathbf{b}^T \in \mathbb{R}^{I \times J}$. The outer product of three vectors $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$ and $\mathbf{c} \in \mathbb{R}^K$ yields a third order rank-one tensor $\mathbf{Z} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c} \in \mathbb{R}^{I \times J \times K}$ where $z_{ijk} = a_i b_j c_k$.
The Kronecker product between two matrices $\mathbf{A} = (a_{ij}) = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_F] \in \mathbb{R}^{I \times F}$ and $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_G] \in \mathbb{R}^{J \times G}$ is defined as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \ldots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \ldots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \ldots & a_{IJ}\mathbf{B} \end{pmatrix}$$

The Khatri-Rao product between two matrices with the same number of columns, $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_F] \in \mathbb{R}^{I \times F}$ and $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_F] \in \mathbb{R}^{J \times F}$, is defined as the column-wise Kronecker product: $\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1, \quad \mathbf{a}_2 \otimes \mathbf{b}_2, \ \ldots \ \mathbf{a}_F \otimes \mathbf{b}_F] \in \mathbb{R}^{IJ \times F}$.

2.2. Preliminaries

A tensor is an object defined on a product between linear spaces. Once the bases of these spaces are fixed, a third order tensor can be represented by a three-way array (a hypermatrix). The order of a tensor hence corresponds to the number of indices of the associated array. One also talks about the number of *ways* or *modes* [27]. In this paper, due to the considered applications, including fluorescence spectroscopy [2][27] or hyperspectral imaging [30], we focus on real positive 3-way arrays denoted by $\mathbf{T} = (t_{ijk}) \in \mathbb{R}^{I \times J \times K}$, admitting the following trilinear decomposition, also known as a triadic decomposition [11] of $\mathbf{T}$

$$\mathbf{T} = \sum_{f=1}^{F} \mathbf{a}_f \circledast \mathbf{b}_f \circledast \mathbf{c}_f, \tag{1}$$

where the three involved matrices $\mathbf{A} = (a_{if}) = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_F] \in \mathbb{R}^{I \times F}$, $\mathbf{B} = (b_{jf}) = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_F] \in \mathbb{R}^{J \times F}$, $\mathbf{C} = (c_{kf}) = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_F] \in \mathbb{R}^{K \times F}$ are the so-called *loading matrices*, whose columns are the *loading factors*, F is an integer. Equivalently, we have the relation between array entries:

$$t_{ijk} = \sum_{f=1}^{F} a_{if} b_{jf} c_{kf} \quad \forall i = 1, \ldots, I \quad \forall j = 1, \ldots, J \quad \forall k = 1, \ldots, K. \tag{2}$$

The smallest integer F that can be found such that the equality above holds exactly is called the *tensor rank* [14]. For this value of F, the above decomposition is called the Canonical Polyadic decomposition (CP) of tensor $\mathbf{T}$. Note that this acronym may also stand for CanDecomp/Parafac, if some readers prefer. Finally, it is sometimes convenient to assume that all vectors have unit length, so that the modified model below is then used, instead of (1):

$$\mathbf{T} = \sum_{f=1}^{F} \lambda_f \, \mathbf{a}_f \circledast \mathbf{b}_f \circledast \mathbf{c}_f \tag{3}$$

where λ_j are real positive scaling factors and $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_F]^T$. The model (3) can be written in a compact form using the Khatri-Rao product $\odot$, as

$$\mathbf{T}_{(1)}^{I,JK} = \mathbf{A}\boldsymbol{\Lambda}(\mathbf{C} \odot \mathbf{B})^T, \tag{4}$$

$$\mathbf{T}_{(2)}^{J,KI} = \mathbf{B}\boldsymbol{\Lambda}(\mathbf{C} \odot \mathbf{A})^T, \tag{5}$$

$$\mathbf{T}_{(3)}^{K,JI} = \mathbf{C}\boldsymbol{\Lambda}(\mathbf{B} \odot \mathbf{A})^T, \tag{6}$$

where $\mathbf{T}_{(1)}^{I,JK}$ (resp. $\mathbf{T}_{(2)}^{J,KI}$ and $\mathbf{T}_{(3)}^{K,JI}$) is the matrix of size $I \times JK$ (resp. $J \times KI$ and $K \times JI$) obtained by unfolding the array $\mathbf{T}$ of size $I \times J \times K$ in the first mode (resp. the second mode and the third mode); $\boldsymbol{\Lambda}$ is the $F \times F$ diagonal matrix defined as $\boldsymbol{\Lambda} = \mathsf{diag}\{\boldsymbol{\lambda}\}$ where operator $\mathsf{diag}\{\cdot\}$ returns a square diagonal matrix which contains in its diagonal the elements of the vector given in argument.

2.3. CP decomposition of 3-way tensors

Assuming that F is known (or overestimated), the problem of the polyadic decomposition of a 3-way tensor $\mathbf{T} \in \mathbb{R}^{I \times J \times K}$ is to estimate the three loading matrices $\mathbf{A} \in \mathbb{R}^{I \times F}$, $\mathbf{B} \in \mathbb{R}^{J \times F}$ and $\mathbf{C} \in \mathbb{R}^{K \times F}$ (and eventually $\boldsymbol{\Lambda} \in \mathbb{R}^{F \times F}$ if the model described in (3) is considered). A rather classical way to solve such a problem consists of minimizing a suitably designed cost function. Typically, we minimize (with respect to the three loading matrices), the cost function:

$$\mathcal{F}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \|\mathbf{T}_{(1)}^{I,JK} - \mathbf{A}\boldsymbol{\Lambda}(\mathbf{C} \odot \mathbf{B})^T\|_F^2 \tag{7}$$

$$= \|\mathbf{T}_{(2)}^{J,KI} - \mathbf{B}\boldsymbol{\Lambda}(\mathbf{C} \odot \mathbf{A})^T\|_F^2 \tag{8}$$

$$= \|\mathbf{T}_{(3)}^{K,JI} - \mathbf{C}\boldsymbol{\Lambda}(\mathbf{B} \odot \mathbf{A})^T\|_F^2, \tag{9}$$

where $\|\cdot\|_F$ stands for the Frobenius norm. In the problem of polyadic canonical decomposition of tensor $\mathbf{T}$, its rank F has to be estimated too.

2.4. Standard approaches

In the tensor literature, there exist several standard ways to solve this optimization problem (see for example [29] for a survey and a comparison of some existing standard methods). The most popular approach is to apply the ALS technique [3, 4, 10, 12], its line search version [2] or more recently its enhanced line search version [25]. In such an approach, the cost function is alternatively optimized with respect to one given loading matrix, the two others being assumed fixed and independent, which is clearly suboptimal. The differential $\mathsf{d}\mathcal{F}$ of $\mathcal{F}$ has to be derived and finally the gradient components (the $I \times F$ matrix $\nabla_{\mathbf{A}}\mathcal{F}$, the $J \times F$ matrix $\nabla_{\mathbf{B}}\mathcal{F}$ and the $K \times F$ matrix $\nabla_{\mathbf{C}}\mathcal{F}$) can be calculated.

We have:

$$\nabla_{\mathbf{A}}\mathcal{F}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) = \left[-\mathbf{T}_{(1)}^{I,JK} + \mathbf{A}\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{B})^T\right](\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda}$$
$$= -\mathbf{T}_{(1)}^{I,JK}(\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda} + \mathbf{A}\mathbf{\Lambda}(\mathbf{C}^T\mathbf{C})\boxdot(\mathbf{B}^T\mathbf{B})\mathbf{\Lambda}, \quad (10)$$
$$\nabla_{\mathbf{B}}\mathcal{F}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) = \left[-\mathbf{T}_{(2)}^{J,KI} + \mathbf{B}\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{A})^T\right](\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda},$$
$$= -\mathbf{T}_{(2)}^{J,KI}(\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda} + \mathbf{B}\mathbf{\Lambda}(\mathbf{C}^T\mathbf{C})\boxdot(\mathbf{A}^T\mathbf{A})\mathbf{\Lambda} \quad (11)$$
$$\nabla_{\mathbf{C}}\mathcal{F}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) = \left[-\mathbf{T}_{(3)}^{K,JI} + \mathbf{C}\mathbf{\Lambda}(\mathbf{B}\odot\mathbf{A})^T\right](\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda}$$
$$= -\mathbf{T}_{(3)}^{K,JI}(\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda} + \mathbf{C}\mathbf{\Lambda}(\mathbf{B}^T\mathbf{B})\boxdot(\mathbf{A}^T\mathbf{A})\mathbf{\Lambda}, \quad (12)$$

where $\boxdot$ stands for the Hadamard (entry-wise) matrix product. Results in the case $\mathbf{\Lambda} = \mathbf{I}_F$, where $\mathbf{I}_F$ is the identity matrix of size $F \times F$, can be found in [5, 8].
By equating the gradient components to zero, a simple solution is obtained:

$$\widehat{\mathbf{A}} = \mathbf{T}_{(1)}^{I,JK}(\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{B})^T)^{\dagger} \quad (13)$$
$$\widehat{\mathbf{B}} = \mathbf{T}_{(2)}^{J,KI}(\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{A})^T)^{\dagger} \quad (14)$$
$$\widehat{\mathbf{C}} = \mathbf{T}_{(3)}^{K,JI}(\mathbf{\Lambda}(\mathbf{B}\odot\mathbf{A})^T)^{\dagger}, \quad (15)$$

where $(\cdot)^{\dagger}$ stands for the pseudo-inverse (Moore-Penrose generalized inverse).
In [7, 8], gradient approaches were mentioned. It was suggested in [21, 28] to use Gauss-Newton approaches, and the Levenberg-Marquardt method was implemented in [7]. Lastly in [13], quasi-Newton approaches have been reported.
However, the polyadic decomposition of $n-$way arrays may be an ill-posed problem and may lead to unstable estimation of its components; one can mention the two-factor degeneracy (2FDs) [23], which corresponds to the presence in the solution of two almost collinear factors with opposite signs, almost cancelling each other.
Moreover, as argued earlier, we concentrate on real nonnegative tensors and their decomposition with real nonnegative loading matrices [27, 30]. Hence in the next sections, we focus on a well-posed problem [15], *i.e.* Nonnegative Tensor Factorization (NTF).

3. Nonnegative 3-way array factorization

In this section, we discuss approaches in which the three loading matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are constrained to be nonnegative.

3.1. Existing approaches

A first approach developed in [21, 28, 29] has been to use some of the existing well-known NonNegative Least Squares (NNLS) methods to solve the following "vectorized" system:

$$\mathsf{vec}\{\mathbf{T}_{(1)}^{I,JK} - \mathbf{A}\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{B})^T\} = \mathbf{0}_{IJK,1} \quad (16)$$

where the vec$\{\cdot\}$ operator applied to a given matrix stacks its columns into a column vector and $\mathbf{0}_{IJK,1}$ is a vector of size $IJK \times 1$ which contains only null elements.
A second approach consists of modifying the previous cost function $\mathcal{F}$, by adding penalty terms whose aim is to impose boundedness on the solution and/or to enforce other specific properties on the solution such as smoothness, sparsity or uncorrelatedness. In [5], it is suggested among other things to use one of the two following cost functions:

$$\mathcal{G}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) = \mathcal{F}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}), +\alpha_A\|\mathbf{A}\|_F^2 + \alpha_B\|\mathbf{B}\|_F^2 + \alpha_C\|\mathbf{C}\|_F^2 \tag{17a}$$

$$\mathcal{G}_1(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) = \mathcal{F}(\mathbf{A},\mathbf{B},\mathbf{C};\mathbf{\Lambda}) + \alpha_A\|\mathbf{A}\|_1 + \alpha_B\|\mathbf{B}\|_1 + \alpha_C\|\mathbf{C}\|_1, \tag{17b}$$

subject to nonnegativity constraints, where α_A, α_B and α_C are nonnegative regularization parameters. In (17a) the standard Tikhonov (l_2-norm) regularization is meant to enforce smoothness of the solution and in (17b) the l_1-norm regularization ($\|\mathbf{A}\|_1 = \sum_{i,j}|a_{ij}|$) is meant to enforce sparsity of the solution. The different various algorithms already evoked in the previous section can be applied to solve that optimization problem. The gradient components given in (10), (11) and (12) are simply replaced by:

$$\nabla_{\mathbf{A}}\mathcal{G}(\cdot) = \nabla_{\mathbf{A}}\mathcal{F}(\cdot) + 2\alpha_A\mathbf{A} \quad \text{or} \quad \nabla_{\mathbf{A}}\mathcal{G}_1(\cdot) = \nabla_{\mathbf{A}}\mathcal{F}(\cdot) + \alpha_A\mathbf{1}_{I,F}, \tag{18}$$

$$\nabla_{\mathbf{B}}\mathcal{G}(\cdot) = \nabla_{\mathbf{B}}\mathcal{F}(\cdot) + 2\alpha_B\mathbf{B} \quad \text{or} \quad \nabla_{\mathbf{B}}\mathcal{G}_1(\cdot) = \nabla_{\mathbf{B}}\mathcal{F}(\cdot) + \alpha_B\mathbf{1}_{J,F}, \tag{19}$$

$$\nabla_{\mathbf{C}}\mathcal{G}(\cdot) = \nabla_{\mathbf{C}}\mathcal{F}(\cdot) + 2\alpha_C\mathbf{C} \quad \text{or} \quad \nabla_{\mathbf{C}}\mathcal{G}_1(\cdot) = \nabla_{\mathbf{C}}\mathcal{F}(\cdot) + \alpha_C\mathbf{1}_{K,F}, \tag{20}$$

where $\mathbf{1}_{K,F}$ stands for the $K \times F$ matrix with ones everywhere. In [5], it was suggested to use the ALS technique again. By equating the gradient components to zero, the solutions in the case of the l_2-norm penalization are found to be equal to:

$$\widehat{\mathbf{A}} = \mathbf{T}_{(1)}^{I,JK}(\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda}\left[\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{B})^T)(\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda} + 2\alpha_A\mathbf{I}_F\right]^\dagger, \tag{21}$$

$$\widehat{\mathbf{B}} = \mathbf{T}_{(2)}^{J,KI}(\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda}\left[\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{A})^T)(\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda} + 2\alpha_B\mathbf{I}_F\right]^\dagger, \tag{22}$$

$$\widehat{\mathbf{C}} = \mathbf{T}_{(3)}^{K,JI}(\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda}\left[\mathbf{\Lambda}(\mathbf{B}\odot\mathbf{A})^T)(\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda} + 2\alpha_C\mathbf{I}_F\right]^\dagger. \tag{23}$$

whereas, in the case of the l_1-norm penalization, they are:

$$\widehat{\mathbf{A}} = \left[\mathbf{T}_{(1)}^{I,JK}(\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda} - \alpha_A\mathbf{1}_{I,F}\right]\left[\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{B})^T)(\mathbf{C}\odot\mathbf{B})\mathbf{\Lambda}\right]^\dagger, \tag{24}$$

$$\widehat{\mathbf{B}} = \left[\mathbf{T}_{(2)}^{J,KI}(\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda} - \alpha_B\mathbf{1}_{J,F}\right]\left[\mathbf{\Lambda}(\mathbf{C}\odot\mathbf{A})^T)(\mathbf{C}\odot\mathbf{A})\mathbf{\Lambda}\right]^\dagger, \tag{25}$$

$$\widehat{\mathbf{C}} = \left[\mathbf{T}_{(3)}^{K,JI}(\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda} - \alpha_C\mathbf{1}_{K,F}\right]\left[\mathbf{\Lambda}(\mathbf{B}\odot\mathbf{A})^T)(\mathbf{B}\odot\mathbf{A})\mathbf{\Lambda}\right]^\dagger. \tag{26}$$

Finally according to [5], a "projection operator" $[\cdot]_+$ is applied, whose aim is to enforce positive entries (since this property is obviously not guaranteed by the penalty terms that have been added).

$$\widehat{\mathbf{A}} \leftarrow \left[\widehat{\mathbf{A}}\right]_+, \quad \widehat{\mathbf{B}} \leftarrow \left[\widehat{\mathbf{B}}\right]_+, \quad \widehat{\mathbf{C}} \leftarrow \left[\widehat{\mathbf{C}}\right]_+. \tag{27}$$

where $[\mathbf{M} = (m_{ij})]_+$ returns a matrix of the same size as $\mathbf{M}$, whose (i,j) entry is $\max\{\epsilon, m_{ij}\}$ if ϵ is a small constant (typically 10^{-16}).

3.2. Suggested approach

3.2.1. Loading matrices parameterization

One obvious way to constraint the loading matrices to have nonnegative entries is to resort to a proper parameterization without modifying the cost function. This kind of parameterization has been recently used in nonnegative matrix factorization [5] problems. To consider that a matrix, say $\mathbf{A}'$, possesses only nonnegative terms, we can simply assume that all its entries are defined as $a'_{ij} = a_{ij}^2$. Using the Hadamard entry-wise product, it implies that $\mathbf{A}' = \mathbf{A} \boxdot \mathbf{A}$, for some (non unique) matrix $\mathbf{A}$. This suggests the following cost function:

$$\begin{aligned}
\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \mathcal{F}(\mathbf{A}\boxdot\mathbf{A},\mathbf{B}\boxdot\mathbf{B},\mathbf{C}\boxdot\mathbf{C}) && (28)\\
&= \|\mathbf{T}_{(1)}^{I,JK} - (\mathbf{A}\boxdot\mathbf{A})\mathbf{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]^T\|_F^2 = \|\boldsymbol{\delta}_{(1)}\|_F^2 && (29)\\
&= \|\mathbf{T}_{(2)}^{J,KI} - (\mathbf{B}\boxdot\mathbf{B})\mathbf{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T\|_F^2 = \|\boldsymbol{\delta}_{(2)}\|_F^2 && (30)\\
&= \|\mathbf{T}_{(3)}^{K,JI} - (\mathbf{C}\boxdot\mathbf{C})\mathbf{\Lambda}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T\|_F^2 = \|\boldsymbol{\delta}_{(3)}\|_F^2, && (31)
\end{aligned}$$

The differential $\mathsf{d}\mathcal{H}$ of $\mathcal{H}$ has to be derived, and then we will be able to calculate the gradient components (the $I \times F$ matrix $\nabla_{\mathbf{A}}\mathcal{H}$, the $J \times F$ matrix $\nabla_{\mathbf{B}}\mathcal{H}$ and the $K \times F$ matrix $\nabla_{\mathbf{C}}\mathcal{H}$) and eventually the Hessian matrices.

With this goal, define the Frobenius scalar product $\langle \mathbf{A},\mathbf{B}\rangle = \mathsf{trace}\{\mathbf{A}^T\mathbf{B}\}$. We also have: $\langle \mathbf{A},\mathbf{A}\rangle = \|\mathbf{A}\|_F^2 = \mathsf{trace}\{\mathbf{A}^T\mathbf{A}\}$. As a consequence, the cost function $\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})$ can be rewritten – in the first mode for example – as:

$$\begin{aligned}
\langle \boldsymbol{\delta}_{(1)},\boldsymbol{\delta}_{(1)}\rangle &= \mathsf{trace}\left\{\boldsymbol{\delta}_{(1)}^T\boldsymbol{\delta}_{(1)}\right\}\\
&= \mathsf{trace}\left\{\left(\mathbf{T}_{(1)}^{I,JK} - (\mathbf{A}\boxdot\mathbf{A})\mathbf{\Lambda}[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})]^T\right)^T\cdot\right.\\
&\qquad\left.\left(\mathbf{T}_{(1)}^{I,JK} - (\mathbf{A}\boxdot\mathbf{A})\mathbf{\Lambda}[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})]^T\right)\right\}.
\end{aligned}$$

The calculation of $\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})$ is performed in Appendix A, and is equal to:

$$\begin{aligned}
\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \langle 4\left[\mathbf{A}\boxdot\left((-\boldsymbol{\delta}_{(1)})\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]\mathbf{\Lambda}\right)\right],\mathsf{d}\mathbf{A}\rangle\\
&+ \langle 4\left[\mathbf{B}\boxdot\left((-\boldsymbol{\delta}_{(2)})\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\mathbf{\Lambda}\right)\right],\mathsf{d}\mathbf{B}\rangle\\
&+ \langle 4\left[\mathbf{C}\boxdot\left((-\boldsymbol{\delta}_{(3)})\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\mathbf{\Lambda}\right)\right],\mathsf{d}\mathbf{C}\rangle \qquad (32)
\end{aligned}$$

3.2.2. Gradient matrices

Using (32), the three gradient components $\nabla_{\mathbf{A}}\mathcal{H}$, $\nabla_{\mathbf{B}}\mathcal{H}$ and $\nabla_{\mathbf{C}}\mathcal{H}$ can be derived:

$$\nabla_{\mathbf{A}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial\mathbf{A}} = 4\mathbf{A}\boxdot\left((-\boldsymbol{\delta}_{(1)})\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]\mathbf{\Lambda}\right), \qquad (33)$$

$$\nabla_{\mathbf{B}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial\mathbf{B}} = 4\mathbf{B}\boxdot\left((-\boldsymbol{\delta}_{(2)})[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})]\mathbf{\Lambda}\right), \qquad (34)$$

$$\nabla_{\mathbf{C}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial\mathbf{C}} = 4\mathbf{C}\boxdot\left((-\boldsymbol{\delta}_{(3)})[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})]\mathbf{\Lambda}\right). \qquad (35)$$

We can then build either the following $(I+J+K)\times F$ matrices $\mathbf{G}^{(k)}$ and $\mathbf{X}^{(k)}$:

$$\mathbf{G}^{(k)} = \begin{pmatrix} \nabla_{\mathbf{A}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)}) \\ \nabla_{\mathbf{B}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)}) \\ \nabla_{\mathbf{C}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)}) \end{pmatrix}, \quad \mathbf{X}^{(k)} = \begin{pmatrix} \mathbf{A}^{(k)} \\ \mathbf{B}^{(k)} \\ \mathbf{C}^{(k)} \end{pmatrix} \tag{36}$$

or the following $(I+J+K)F\times 1$ vectors:

$$\mathbf{g}^{(k)} = \begin{pmatrix} \mathsf{vec}\{\nabla_{\mathbf{A}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)})\} \\ \mathsf{vec}\{\nabla_{\mathbf{B}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)})\} \\ \mathsf{vec}\{\nabla_{\mathbf{C}}\mathcal{H}(\mathbf{A}^{(k)},\mathbf{B}^{(k)},\mathbf{C}^{(k)})\} \end{pmatrix}, \quad \mathbf{x}^{(k)} = \begin{pmatrix} \mathsf{vec}\{\mathbf{A}^{(k)}\} \\ \mathsf{vec}\{\mathbf{B}^{(k)}\} \\ \mathsf{vec}\{\mathbf{C}^{(k)}\} \end{pmatrix} \tag{37}$$

4. Preconditioned nonlinear conjugate gradient algorithms

To estimate the three loading matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, the cost function $\mathcal{H}$ given in (29), (30) or (31) has to be minimized. To that aim, we suggest to optimize the cost function $\mathcal{H}$ simultaneously with respect to all variables using a preconditioned nonlinear conjugate gradient method [26].
In the classical gradient approach, variable $\mathbf{X}$ given in (36) is updated at each iteration k ($k = 1, 2, \ldots$) according to the following adaptation rule:

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} - \mu^{(k)}\mathbf{G}^{(k)} \quad \text{or} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mu^{(k)}\mathbf{g}^{(k)}, \tag{38}$$

where $\mathbf{G}^{(k)}$ is the gradient matrix given in (36) using (33), (34) and (35) and $\mu^{(k)}$ the step size (the problem of the choice of the stepsize is treated in Section 4.3). We notice that when the nonnegativity constraint no more holds, (33), (34) and (35) are simply respectively replaced by (10), (11) and (12).
In the preconditioned conjugate gradient approach, the descent direction is initialized using $\mathbf{d}^{(1)} = -\mathbf{g}^{(1)}$ and updated at each iteration k according to the following adaptation rule:

$$\left\{ \begin{array}{ll} \mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} + \mu^{(k)}\mathbf{d}^{(k)} \\ \mathbf{d}^{(k+1)} & = -(\mathbf{M}^{(k+1)})^{-1}\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)} \end{array} \right. \tag{39}$$

The $(I+J+K)F\times 1$ vector $\mathbf{d}^{(k)}$ contains the search directions and the square $(I+J+K)F\times(I+J+K)F$ matrix $\mathbf{M}$ stands for the preconditioner. As noticed in [24][26], the nonlinear conjugate gradient method can be preconditioned by choosing a preconditioner $\mathbf{M}$ that approximates the Hessian matrix or at least its diagonal. In the nonlinear conjugate gradient, two expressions for the value of β are classically used: the Fletcher-Reeves (β_{FR}) and the Polak-Ribière (β_{PR}) formulas [24]:

$$\beta_{\mathsf{FR}}^{(k+1)} = \frac{\mathbf{g}^{(k+1)^T}\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)^T},\mathbf{g}^{(k)}} \tag{40}$$

$$\beta_{\mathsf{PR}}^{(k+1)} = \frac{\mathbf{g}^{(k+1)^T}(\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)})}{\mathbf{g}^{(k)^T}\mathbf{g}^{(k)}}. \tag{41}$$

Finally, as noticed in [24] (p. 102), if we reinitialize a conjugate gradient method by setting $\mathbf{d}^{(i)} = -\mathbf{g}^{(i)}$, from time to time, we might get better performance than by constructing $\mathbf{d}^{(i)}$ by one of the standard formulas (*i.e.* combining (39) and (40) or (39) and (41)) at each iteration. In our case, we have chosen to perform this "restart" every $(I + J + K)F$ iterations.

4.1. Particular cases

4.1.1. Nonlinear conjugate gradient algorithm

Considering that $\mathbf{M} = \mathbf{I}_{(I+J+K)F}$ in (39), we simply obtain the nonlinear conjugate gradient method:

$$\left\{ \begin{array}{ll} \mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} + \mu^{(k)}\mathbf{d}^{(k)} \\ \mathbf{d}^{(k+1)} & = -\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)} \end{array} \right. \tag{42}$$

which can be equivalently written in the ensuing matrix form:

$$\left\{ \begin{array}{ll} \mathbf{X}^{(k+1)} & = \mathbf{X}^{(k)} + \mu^{(k)}\mathbf{D}^{(k)} \\ \mathbf{D}^{(k+1)} & = -\mathbf{G}^{(k+1)} + \beta^{(k)}\mathbf{D}^{(k)} \end{array} \right. \tag{43}$$

with

$$\mathbf{D}^{(k)} = \begin{pmatrix} \mathbf{D}_{\mathbf{A}}^{(k)} \\ \mathbf{D}_{\mathbf{B}}^{(k)} \\ \mathbf{D}_{\mathbf{C}}^{(k)} \end{pmatrix}, \qquad \mathbf{d}^{(k)} = \begin{pmatrix} \mathsf{vec}\{\mathbf{D}_{\mathbf{A}}^{(k)}\} \\ \mathsf{vec}\{\mathbf{D}_{\mathbf{B}}^{(k)}\} \\ \mathsf{vec}\{\mathbf{D}_{\mathbf{C}}^{(k)}\} \end{pmatrix} = \begin{pmatrix} \mathbf{d}_{\mathbf{A}}^{(k)} \\ \mathbf{d}_{\mathbf{B}}^{(k)} \\ \mathbf{d}_{\mathbf{C}}^{(k)} \end{pmatrix} \tag{44}$$

The two expressions for the value of β that are classically used remain the Fletcher-Reeves (β_{FR}) and the Polak-Ribière (β_{PR}) formulas (now written using matrices instead of vectors) [24]:

$$\beta_{\mathsf{FR}}^{(k+1)} = \frac{\langle \mathbf{G}^{(k+1)}, \mathbf{G}^{(k+1)} \rangle}{\langle \mathbf{G}^{(k)}, \mathbf{G}^{(k)} \rangle} = \frac{\|\mathbf{G}^{(k+1)}\|_F^2}{\|\mathbf{G}^{(k)}\|_F^2}, \tag{45}$$

$$\beta_{\mathsf{PR}}^{(k+1)} = \frac{\langle \mathbf{G}^{(k+1)}, \mathbf{G}^{(k+1)} - \mathbf{G}^{(k)} \rangle}{\langle \mathbf{G}^{(k)}, \mathbf{G}^{(k)} \rangle} = \frac{\langle \mathbf{G}^{(k+1)}, \mathbf{G}^{(k+1)} - \mathbf{G}^{(k)} \rangle}{\|\mathbf{G}^{(k)}\|_F^2}. \tag{46}$$

And again, this algorithm is initialized by using $\mathbf{D}^{(1)} = -\mathbf{G}^{(1)}$ and restarted after a given number, say $(I + J + K)F$ of iterations, with $\mathbf{D}^{(i)} = -\mathbf{G}^{(i)}$ as initial guess, to speed up the convergence.

4.1.2. Quasi Newton approaches (BFGS and DFP algorithms)

In (39), by setting $\beta = 0$ and considering that the preconditioner $\mathbf{M}$ is a $(I + J + K)F \times (I + J + K)F$ approximation of the Hessian matrix given by (47):

$$\mathbf{M}^{(k+1)} = \mathbf{M}^{(k)} + \frac{\Delta\mathbf{g}^{(k)}(\Delta\mathbf{g}^{(k)})^T}{\langle \Delta\mathbf{g}^{(k)}, \Delta\mathbf{x}^{(k)} \rangle} - \frac{(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})^T}{\langle \mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)}, \Delta\mathbf{x}^{(k)} \rangle}, \tag{47}$$

we obtain the following adaptation rule as in the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm:

$$\left\{\begin{array}{ll}\mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} - \mu^{(k)}(\mathbf{M}^{(k)})^{-1}\mathbf{g}^{(k)} \\ \Delta\mathbf{x}^{(k)} & = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \Delta\mathbf{g}^{(k)} & = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} \\ \mathbf{M}^{(k+1)} & = \mathbf{M}^{(k)} + \frac{\Delta\mathbf{g}^{(k)}(\Delta\mathbf{g}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\Delta\mathbf{x}^{(k)}\rangle} - \frac{(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})^T}{\langle\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)},\Delta\mathbf{x}^{(k)}\rangle}\end{array}\right. \tag{48}$$

Using the inversion lemma and denoting by $\rho = \frac{1}{(\Delta\mathbf{g}^{(k)})^T\Delta\mathbf{x}^{(k)}}$, the inverse of the approximate Hessian matrix $\mathbf{M}^{(k)}$ can be estimated. The algorithm in (48) can be rewritten:

$$\left\{\begin{array}{ll}\mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} - \mu^{(k)}(\mathbf{M}^{(k)})^{-1}\mathbf{g}^{(k)} \\ \Delta\mathbf{x}^{(k)} & = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \Delta\mathbf{g}^{(k)} & = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} \\ (\mathbf{M}^{(k+1)})^{-1} & = (\mathbf{M}^{(k)})^{-1} + \rho\left[1+\rho(\Delta\mathbf{g}^{(k)})^T(\mathbf{M}^{(k)})^{-1}\Delta\mathbf{g}^{(k)}\right]\Delta\mathbf{x}^{(k)}(\Delta\mathbf{x}^{(k)})^T \\ & \quad -\rho\Delta\mathbf{x}^{(k)}(\Delta\mathbf{g}^{(k)})^T(\mathbf{M}^{(k)})^{-1} - \rho(\mathbf{M}^{(k)})^{-1}\Delta\mathbf{g}^{(k)}(\Delta\mathbf{x}^{(k)})^T\end{array}\right. \tag{49}$$

On the other hand, setting $\beta = 0$ in (39), and considering that the preconditioner $\mathbf{M}$ is a $(I+J+K)F \times (I+J+K)F$ approximation of the inverse of the Hessian matrix as:

$$\mathbf{M}^{(k+1)} = \mathbf{M}^{(k)} + \frac{\Delta\mathbf{x}^{(k)}(\Delta\mathbf{x}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\Delta\mathbf{x}^{(k)}\rangle} - \frac{(\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)})(\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)}\rangle} \tag{50}$$

we obtain the following adaption rule as in the Davidon-Fletcher-Powell algorithm (DFP):

$$\left\{\begin{array}{ll}\mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} - \mu^{(k)}\mathbf{M}^{(k)}\mathbf{g}^{(k)} \\ \Delta\mathbf{x}^{(k)} & = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \Delta\mathbf{g}^{(k)} & = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} \\ \mathbf{M}^{(k+1)} & = \mathbf{M}^{(k)} + \frac{\Delta\mathbf{x}^{(k)}(\Delta\mathbf{x}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\Delta\mathbf{x}^{(k)}\rangle} - \frac{(\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)})(\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\mathbf{M}^{(k)}\Delta\mathbf{g}^{(k)}\rangle}\end{array}\right. \tag{51}$$

In the two cases, the algorithm is initialized using for $\mathbf{M}^{(1)}$ (or $(\mathbf{M}^{(1)})^{-1}$), a symmetric, $(I+J+K)F \times (I+J+K)F$ positive-definite matrix.

4.1.3. Levenberg-Marquardt algorithm

When the preconditioner $\mathbf{M}$ tends to loose its "hereditary positive-definiteness" property through the iterations, and hence may fail to construct descent directions, it is better to stabilize it using trust region techniques that modify $\mathbf{M}$ by adding a multiple of the identity matrix as in the Levenberg-Marquardt approach [17]:

$$\left\{\begin{array}{ll}\mathbf{x}^{(k+1)} & = \mathbf{x}^{(k)} - \mu^{(k)}(\mathbf{M}^{(k)} + \alpha\mathbf{I}_{(I+J+K)F})^{-1}\mathbf{g}^{(k)} \\ \Delta\mathbf{x}^{(k)} & = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \Delta\mathbf{g}^{(k)} & = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} \\ \mathbf{M}^{(k+1)} & = \mathbf{M}^{(k)} + \frac{\Delta\mathbf{g}^{(k)}(\Delta\mathbf{g}^{(k)})^T}{\langle\Delta\mathbf{g}^{(k)},\Delta\mathbf{x}^{(k)}\rangle} - \frac{(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})(\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)})^T}{\langle\mathbf{M}^{(k)}\Delta\mathbf{x}^{(k)},\Delta\mathbf{x}^{(k)}\rangle}\end{array}\right. \tag{52}$$

where α is a relaxation coefficient. We notice that by setting $\alpha = 0$ in (52), the quasi-Newton algorithm (48) is recovered. On the other hand, by setting $\mathbf{M} = \mathbf{I}_{(I+J+K)F}$ in (52), or by taking α large enough, the gradient algorithm in (38) is obtained.

4.2. Algorithmic complexity

Regarding the algorithmic complexity of the ALS algorithm, the calculation has been done in [7]. It amounts to $O(7F^2(JK+KI+IJ)+3FIJK)$. For the gradient algorithm, the computational cost per iteration k approximately amounts to $O(6IFJK)$, since for each of the three gradient components, we have four operations: 2 matrices products[1] + 1 Khatri-Rao product[2] + 1 addition. This computational cost is thus governed by the calculation of the matrix $\mathbf{G}$. The total number of arithmetic operations is $O(6IFJKN_{it})$ if N_{it} stands for the total number of iterations to reach convergence. For the gradient algorithm with nonnegativity constraint, the algorithmic complexity is nearly the same *i.e.* $O(6IFJK)$ and the total number of arithmetic operations is $O(6IFJKN_{it})$ too.
For the nonlinear conjugate gradient method (in both cases *i.e.* with or without the nonnegativity constraint), the algorithmic complexity approximatively amounts to $O(6FIJK + 2(I + J + K)F^2)$, since the calculation of β adds two matrices multiplications.
For the BFGS method (Newton-Raphson approach, in both cases *i.e.* with or without the nonnegativity constraint) the algorithmic complexity per iteration amounts to $O(6IFJK + 4(I + J + K)^2F^2 + (I + J + K)^3F^3)$ since 4 matrices multiplications and one linear system solving[3] have been added. Finally the computational cost per iteration $\approx O((I + J + K)^3F^3)$ which implies that it is mainly governed by the linear system solving.
If the PC latter is avoided, using (49) instead of (48), the computational cost is reduced to $\approx O(4(I + J + K)^2F^2)$.
For the DFP method, (in both cases *i.e.* with or without the nonnegativity constraint) the algorithmic complexity per iteration amounts to: $O(4(I + J + K)^2F^2)$. Finally, for the preconditioned linear conjugate gradient method (in both cases *i.e.* with or without the nonnegativity constraint), the algorithmic complexity per iteration amounts to $\approx O((I+J+K)^3F^3)$ too, since the overhead due to the calculation of β is negligible. These results are summarized in Table 1.

4.3. How to choose $\mu^{(k)}$?

4.3.1. Enhanced line search (ELS)

The ELS enhancement is applicable to any iterative algorithm, provided the optimization criterion is a polynomial or a rational function. It searches for the best stepsize μ_{opt} that corresponds to the *global minimum* of (28), (29), (30) or (31). It requires the algebraic minimization of the

[1] The cost for multiplying the $N \times M$ matrix $\mathbf{B}$ by the $M \times P$ matrix $\mathbf{A}$ is assumed $O(NMP)$.

[2] The cost for calculating the Khatri-Rao product between the $N \times M$ matrix $\mathbf{B}$ by the $P \times M$ matrix $\mathbf{A}$ is assumed $O(NMP)$.

[3] The cost for inverting the $N \times N$ matrix $\mathbf{B}$ is assumed $O(N^3)$ (Gauss-Jordan elimination). This cost could be reduced using another algorithm.

following quantity w.r.t. μ:

$$\mathcal{H}(\mathbf{A}^{(k+1)}, \mathbf{B}^{(k+1)}, \mathbf{C}^{(k+1)}) = \mathcal{H}\Big[(\mathbf{A}^{(k)} + \mu {\mathbf{D}_\mathbf{A}}^{(k)}) \boxdot (\mathbf{A}^{(k)} + \mu {\mathbf{D}_\mathbf{A}}^{(k)}),$$
$$(\mathbf{B}^{(k)} + \mu {\mathbf{D}_\mathbf{B}}^{(k)}) \boxdot (\mathbf{B}^{(k)} + \mu {\mathbf{D}_\mathbf{B}}^{(k)}), (\mathbf{C}^{(k)} + \mu {\mathbf{D}_\mathbf{C}}^{(k)}) \boxdot (\mathbf{C}^{(k)} + \mu {\mathbf{D}_\mathbf{C}}^{(k)})\Big]. \quad (53)$$

As shown in Appendix B, this quantity is a 12th-degree polynomial, whose expression is given by (we opt to omit the dependency upon the parameters of $\mathcal{H}$ to simplify the various expressions):

$$\mathcal{H}(.) = \sum_{i=0}^{12} a_i \mu^i, \quad (54a)$$

$$\mathsf{d}\mathcal{H}(.) = \sum_{i=0}^{11} (i+1) a_{i+1} \mu^i, \quad (54b)$$

where the thirteen coefficients a_i, for $i = 0, \ldots, 12$ are given by (see Appendix B to get the definition of $\mathbf{K}_{(i)}$, where i varies from 1 to 6):

$$a_0 = \mathsf{trace}\left[\mathbf{K_0}{\mathbf{K_0}}^T\right] \quad (55a)$$
$$a_1 = \mathsf{trace}\left[2\mathbf{K_1}{\mathbf{K_0}}^T\right] \quad (55b)$$
$$a_2 = \mathsf{trace}\left[2\mathbf{K_2}{\mathbf{K_0}}^T + \mathbf{K_1}{\mathbf{K_1}}^T\right] \quad (55c)$$
$$a_3 = \mathsf{trace}\left[2\left(\mathbf{K_3}{\mathbf{K_0}}^T + \mathbf{K_2}{\mathbf{K_1}}^T\right)\right] \quad (55d)$$
$$a_4 = \mathsf{trace}\left[2\left(\mathbf{K_4}{\mathbf{K_0}}^T + \mathbf{K_3}{\mathbf{K_1}}^T\right) + \mathbf{K_2}{\mathbf{K_2}}^T\right] \quad (55e)$$
$$a_5 = \mathsf{trace}\left[2\left(\mathbf{K_5}{\mathbf{K_0}}^T + \mathbf{K_4}{\mathbf{K_1}}^T + \mathbf{K_3}{\mathbf{K_2}}^T + \mathbf{K_3}{\mathbf{K_2}}^T\right)\right] \quad (55f)$$
$$a_6 = \mathsf{trace}\left[2\left(\mathbf{K_6}{\mathbf{K_0}}^T + \mathbf{K_5}{\mathbf{K_1}}^T + \mathbf{K_4}{\mathbf{K_2}}^T\right) + \mathbf{K_3}{\mathbf{K_3}}^T\right] \quad (55g)$$
$$a_7 = \mathsf{trace}\left[2\left(\mathbf{K_6}{\mathbf{K_1}}^T + \mathbf{K_5}{\mathbf{K_2}}^T + \mathbf{K_4}{\mathbf{K_3}}^T\right)\right] \quad (55h)$$
$$a_8 = \mathsf{trace}\left[2\left(\mathbf{K_6}{\mathbf{K_2}}^T + \mathbf{K_5}{\mathbf{K_3}}^T\right) + \mathbf{K_4}\mathbf{K_4}\right] \quad (55i)$$
$$a_9 = \mathsf{trace}\left[2\left(\mathbf{K_6}{\mathbf{K_3}}^T + \mathbf{K_5}{\mathbf{K_4}}^T\right)\right] \quad (55j)$$
$$a_{10} = \mathsf{trace}\left[2\mathbf{K_6}{\mathbf{K_4}}^T + \mathbf{K_5}{\mathbf{K_5}}^T\right] \quad (55k)$$
$$a_{11} = \mathsf{trace}\left[2\mathbf{K_6}{\mathbf{K_5}}^T\right] \quad (55l)$$
$$a_{12} = \mathsf{trace}\left[\mathbf{K_6}{\mathbf{K_6}}^T\right] \quad (55m)$$

By differentiating the expression of $\mathcal{H}$ with respect to μ, we obtain the polynomial of degree 11 given in (54b). The optimal stepsize μ_{opt} then corresponds to the real and positive root of the 11-order polynomial defined in (54b) leading to the global minimum of criterion (54a).
Concerning the algorithmic complexity, the cost is now ruled by the calculation of the 13 coefficients of the 12th-degree polynomial given in (54a). The obtained results are summarized in Table 2.

4.3.2. Backtracking

The main problem with the enhanced line search is its computational cost. As already noticed, the cost in the ELS version of the algorithms is dominated by the calculation of the 13 coefficients of the polynomial we intend to minimize. An alternative approach, consists of computing the locally optimal step size (called backtracking) and to alternate it with ELS every 10 or 20 iterations for example. The main advantage of such an approach is its low computational cost. Backtracking is a standard technique, which attempts to determine a sufficiently long step size while still producing some amount of decrease in the cost function. As a consequence, the method implies to start with a step μ large enough (for example a unit step size) and to decrease it iteratively by a factor β *i.e.* $\mu = \beta\mu$ (with β commonly chosen between 0.1 and 0.8) until the Armijo condition [1][18] given in (56) is fulfilled. The resulting μ is the stepsize $\mu^{(k)}$ used in the updating rule of the optimization algorithm. We still assume the same cost function $\mathcal{H}$ given by (29). During the updating stage of the considered algorithm, it becomes $\mathcal{H}(\mathbf{A}+\mu\mathbf{D_A}, \mathbf{B}+\mu\mathbf{D_B}, \mathbf{C}+\mu\mathbf{D_C})$ given in (53). Thus, with our notations, the Armijo condition reads:

$$\mathcal{H}(\mathbf{A}+\mu\mathbf{D_A}, \mathbf{B}+\mu\mathbf{D_B}, \mathbf{C}+\mu\mathbf{D_C}) < \mathcal{H}(\mathbf{A}, \mathbf{B}, \mathbf{C}) + \alpha\,\mu\,\mathbf{g}^T\mathbf{d} \tag{56}$$

where α is a constant parameter often chosen within $[10^{-4},\ 10^{-1}]$, $\mathbf{d}$ is the descent direction given in (44) and $\mathbf{g}$ is the gradient given in (37). Since $\mathbf{d}$ is a descent direction, we have $\mathbf{g}^T\mathbf{d} < 0$ (in the specific case of the gradient algorithm, $\mathbf{d} = -\mathbf{g}$, whereas $\mathbf{d} = -\mathbf{M}^{-1}\mathbf{g}$ for quasi-Newton algorithms).

It is also possible to combine the backtracking method together with a search by adjustment method, whose advantage is to include a "memory" of the previous steps. For example, if the stepsize μ found during the backtracking stage is lower than the initial step called μ_0, μ_0 is decreased (this new value will be used for the next backtracking stages) by a factor β. On the opposite, if it is higher than μ_0, μ_0 is increased by another factor $\alpha > 1$.

5. Computer simulations

Simulations are now provided to illustrate the behavior and the performances of the proposed NTF algorithms. With this goal, we address the problem of fluorescence analysis. If a solution is excited by an optical excitation, several effects may be produced: Rayleigh scatter , Raman scatter and Fluorescence. At low concentrations, the Beer-Lambert law can be linearized so that the fluorescence intensity rather accurately follows the model below [27, 16]:

$$I(\lambda_f, \lambda_e, k) = I_o\,\gamma(\lambda_f)\,\epsilon(\lambda_e)\,c_k$$

where ϵ denotes absorbance spectrum (sometimes called emission spectrum), λ_f is the fluorescence emission wavelength, λ_e the excitation wavelength, γ the fluorescence emission spectrum and c_k is the concentration of the fluorescent component in sample number k. Provided it can be separated from diffusion phenomena, the fluorescence phenomenon allows to determine the concentration of a diluted (fluorescent) chemical component, and possibly to recognize it, thanks to its fluorescent spectrum.

A difficulty appears when the solution contains more than one fluorescent solute. In such a case, the overall fluorescence intensity is an unknown linear combination of component fluorescence intensities:

$$I(\lambda_f, \lambda_e, k) = I_o \sum_{\ell} \gamma_\ell(\lambda_f)\, \epsilon_\ell(\lambda_e)\, c_{k,\ell} \tag{57}$$

$c_{k,\ell}$ stands for the concentration of the ℓ-th fluorescent solute in sample k. It is then necessary to separate each component contribution. Assuming that a finite number of excitation and emission frequencies are measured, so that the measurements are stored in a finite array of order 3 and finite dimensions, say $I \times J \times K$, $t_{ijk} = I(\lambda_f(i), \lambda_e(j), k)$. It is clear, by comparing equations (57) and (2), that thanks to uniqueness of the CP decomposition, one can identify $\gamma_\ell(\lambda_f(i))$ with a_{if}, $\epsilon_\ell(\lambda_e(j))$ with b_{jf} and $c_{k,\ell}$ with c_{kf}. Hence, the computation of the CP decomposition yields emission spectra of each component as well as their concentration. There is no need to know in advance what are the components expected to be present in the solution.

Two tensors T_1 and T_2 have been simulated, using $F = 4$ components whose 71×47 emission-excitation matrices ($\mathbf{a}_i \mathbf{b}_i^T$, $\forall i = 1, \ldots, 4$) were very similar to the ones displayed in Fig. 4. These images [20] were provided by the PROTEE-EA 3819 Laboratory at the South Toulon Var University, France. Two random positive matrices $\mathbf{C}$ have been used (a 10×4 matrix and a 128×4 matrix). The first tensor $\mathbf{T}_1$ is $71 \times 47 \times 10$, and the second tensor $\mathbf{T}_2$ is $71 \times 47 \times 128$. To establish a comparison between the different algorithms, we need an error index. We have chosen to use: $E = \|\mathbf{T} - \widehat{\mathbf{T}}\|_F^2$ or $E_{dB} = 10 \log_{10}(E)$, with $\widehat{\mathbf{T}} = \sum_{f=1}^{F} \widehat{\mathbf{a}}_f \circledast \widehat{\mathbf{b}}_f \circledast \widehat{\mathbf{c}}_f$ and $\widehat{\mathbf{a}}$, $\widehat{\mathbf{b}}$ and $\widehat{\mathbf{c}}$ the estimated factors. The best results are obtained when the error index E is found to be close to 0 in linear scale ($-\infty$ in logarithmic scale).

In the left column of Fig. 1, we have compared the results obtained with ELS versions of various algorithms (*i.e.* ELS is executed at each iteration, except for the so-called ALS-Cichocki and NTF-HALS algorithms, in which there is no ELS enhancement) versus iterations, while the results are represented versus the number of arithmetic operations in the right column of the Fig. 1. For Figures 1, 2 and 3, all the algorithms were initialized using Bro's DTLD algorithm [27]. For the ALS-Cichocki algorithm with either l_1-norm or l_2-norm regularization, we have chosen $\alpha_A = \alpha_B = \alpha_C = 10^{-6}$ (it is the reason why the performances are bounded). For the NTF-HALS algorithm, we have implemented the algorithm described p. 357 of [5] . We can observe that both quasi-Newton algorithms (BFGS and DFP) have nearly the same behavior. The conjugate gradient and gradient algorithms require more time to reach convergence. However, the conjugate gradient algorithm offers a good compromise between speed and performances and contrary to quasi-Newton algorithms, it does not require the estimation of the $(I + J + K)F \times (I + J + K)F$ Hessian matrices (or their approximation) and as a consequence it can be applied to very large tensors. Even though the NTF-ALS and NTF-HALS algorithms are often the fastest algorithms during the first iterations , we can observe in the bottom of Fig. 5, that the reconstructed emission-excitation matrices are not necessarily good (even if the reconstruction error was weak; the estimated emission-excitation matrices have to be compared with the true emission-excitation matrices that were perfectly estimated in the Fig. 4 when there is an error in the model (here, $F = 5$ was assumed whereas four components were effectively present in the mixture). In the chosen example (where all the algorithms were initialized using the same

random initialization), the algorithms we propose seem less sensitive to this kind of model error as observed in the top of the Fig. 5. Finally, a good way to diminish the global computational time consists of alternating between ELS (say, every 10 or 20 iterations) and backtracking, as it can be observed in the Fig. 2 and 3 .

6. Conclusion

In this article, we have described several algorithms able to compute the minimal polyadic decomposition of nonnegative three-way arrays. The calculation of gradient matrices has been performed, allowing to implement preconditioned nonlinear conjugate gradient, gradient and quasi-Newton approaches. Two versions of each algorithm have been studied: the enhanced line search (ELS) version, and a backtracking version alternating with ELS. The algorithmic complexity has been provided too. Finally, computer simulations have been performed in the context of data analysis, in order to demonstrate both the good behavior of the algorithms we proposed, compared to others put forward in the literature, and their usefulness in data mining applications. As demonstrated in Section 5, the judgement of an algorithm should not solely rely on the reconstruction error and on computational complexity, but should also take into account the error in the loading matrices obtained.

Appendix

Appendix A. Calculation of $\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})$

We use similar properties regarding the trace as those already used in [9]. Considering three $M \times M$ square matrices $\mathbf{D}_1$, $\mathbf{D}_2$ and $\mathbf{D}_3$ and four rectangular matrices $\mathbf{D}_4$, $\mathbf{D}_5$, $\mathbf{D}_6$ and $\mathbf{D}_7$ (resp. $M \times N$, $N \times M$, $M \times N$ and $M \times N$), we have the following properties [19]:

$\mathbf{P}_0$. $(\mathbf{D}_4\mathbf{D}_5)^T = \mathbf{D}_5^T\mathbf{D}_4^T$.
$\mathbf{P}_1$. $\mathsf{trace}\{\mathbf{D}_1\} = \mathsf{trace}\{\mathbf{D}_1^T\}$.
$\mathbf{P}_2$. $\mathsf{trace}\{\mathbf{D}_1 + \mathbf{D}_2\} = \mathsf{trace}\{\mathbf{D}_1\} + \mathsf{trace}\{\mathbf{D}_2\}$.
$\mathbf{P}_3$. $\mathsf{trace}\{\mathbf{D}_1\mathbf{D}_2\mathbf{D}_3\} = \mathsf{trace}\{\mathbf{D}_3\mathbf{D}_1\mathbf{D}_2\} = \mathsf{trace}\{\mathbf{D}_2\mathbf{D}_3\mathbf{D}_1\}$
$\Rightarrow \mathsf{trace}\{\mathbf{D}_1\mathbf{D}_2\} = \mathsf{trace}\{\mathbf{D}_2\mathbf{D}_1\}$.
$\mathbf{P}_4$. $\mathsf{trace}\{\mathbf{D}_4\mathbf{D}_5\} = \mathsf{trace}\{\mathbf{D}_5\mathbf{D}_4\}$.
$\mathbf{P}_5$. $\mathsf{d}(\mathbf{D}_1^T) = (\mathsf{d}\mathbf{D}_1)^T$.
$\mathbf{P}_6$. $\mathsf{d}(\mathbf{D}_1\mathbf{D}_2) = \mathsf{d}\mathbf{D}_1\mathbf{D}_2 + \mathbf{D}_1\mathsf{d}\mathbf{D}_2$.
$\mathbf{P}_7$. $\mathsf{d}(\mathbf{D}_1 + \mathbf{D}_2) = \mathsf{d}\mathbf{D}_1 + \mathsf{d}\mathbf{D}_2$.
$\mathbf{P}_8$. $\mathsf{d}(\mathsf{trace}\{\mathbf{D}_1\}) = \mathsf{trace}\{\mathsf{d}\mathbf{D}_1\}$.
$\mathbf{P}_9$. $\mathsf{d}(\mathbf{D}_1 \boxdot \mathbf{D}_2) = \mathsf{d}\mathbf{D}_1 \boxdot \mathbf{D}_2 + \mathbf{D}_1 \boxdot \mathsf{d}\mathbf{D}_2 \Rightarrow \mathsf{d}(\mathbf{D}_1 \boxdot \mathbf{D}_1) = 2\mathbf{D}_1 \boxdot \mathsf{d}\mathbf{D}_1$.
$\mathbf{P}_{10}$. $\mathbf{D}_4 \boxdot \mathbf{D}_6 = \mathbf{D}_6 \boxdot \mathbf{D}_4$.
$\mathbf{P}_{11}$. $(\mathbf{D}_4 \boxdot \mathbf{D}_6)^T = \mathbf{D}_4^T \boxdot \mathbf{D}_6^T$.
$\mathbf{P}_{12}$. $\mathsf{trace}\{\mathbf{D}_4^T(\mathbf{D}_6 \boxdot \mathbf{D}_7)\} = \mathsf{trace}\{(\mathbf{D}_4^T \boxdot \mathbf{D}_6^T)\mathbf{D}_7\}$.

Like in [8], our aim is to obtain:

$$\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \langle \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{A}}, \mathsf{d}\mathbf{A}\rangle + \langle \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{B}}, \mathsf{d}\mathbf{B}\rangle + \langle \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{C}}, \mathsf{d}\mathbf{C}\rangle, \quad (58)$$

where $\frac{\partial \cdot}{\partial \mathbf{A}}$ means the partial derivative with respect to the matrix $\mathbf{A}$.

Or, using circular permutations and the aforementioned properties $\mathbf{P}_1 - \mathbf{P}_9$, we have:

$$\begin{aligned}
\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \mathsf{trace}\left\{\mathsf{d}(\boldsymbol{\delta}_{(1)}^T)\boldsymbol{\delta}_{(1)}\right\} + \mathsf{trace}\left\{\boldsymbol{\delta}_{(1)}^T \mathsf{d}\boldsymbol{\delta}_{(1)}\right\} \\
&= 2\mathsf{trace}\left\{\boldsymbol{\delta}_{(1)}^T \mathsf{d}\boldsymbol{\delta}_{(1)}\right\} = 2\mathsf{trace}\left\{\boldsymbol{\delta}_{(2)}^T \mathsf{d}\boldsymbol{\delta}_{(2)}\right\} = 2\mathsf{trace}\left\{\boldsymbol{\delta}_{(3)}^T \mathsf{d}\boldsymbol{\delta}_{(3)}\right\} \\
&= 4\mathsf{trace}\left\{-\boldsymbol{\delta}_{(1)}^T(\mathbf{A}\boxdot \mathsf{d}\mathbf{A})\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]^T - \boldsymbol{\delta}_{(2)}^T(\mathbf{B}\boxdot \mathsf{d}\mathbf{B})\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T\right. \\
&\left. -\boldsymbol{\delta}_{(3)}^T(\mathbf{C}\boxdot \mathsf{d}\mathbf{C})\boldsymbol{\Lambda}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T\right\} \\
&= \mathsf{trace}\left\{4\left(\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]^T(-\boldsymbol{\delta}_{(1)})^T\right)(\mathbf{A}\boxdot \mathsf{d}\mathbf{A})\right\} \\
&+ \mathsf{trace}\left\{4\left(\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T(-\boldsymbol{\delta}_{(2)})^T\right)(\mathbf{B}\boxdot \mathsf{d}\mathbf{B})\right\} \\
&+ \mathsf{trace}\left\{4\left(\boldsymbol{\Lambda}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T(-\boldsymbol{\delta}_{(3)})^T\right)(\mathbf{C}\boxdot \mathsf{d}\mathbf{C})\right\}
\end{aligned}$$

Using property $\mathbf{P}_{10} - \mathbf{P}_{12}$ ([19], p. 53) and the fact that $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T$ since $\boldsymbol{\Lambda}$ is diagonal , we have:

$$\begin{aligned}
\mathsf{d}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \mathsf{trace}\left\{4\left[\left(\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]^T(-\boldsymbol{\delta}_{(1)})^T\right)\boxdot\mathbf{A}^T\right]\mathsf{d}\mathbf{A}\right\} \\
&+ \mathsf{trace}\left\{4\left[\left(\boldsymbol{\Lambda}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T(-\boldsymbol{\delta}_{(2)})^T\right)\boxdot\mathbf{B}^T\right]\mathsf{d}\mathbf{B}\right\} \\
&+ \mathsf{trace}\left\{4\left[\left(\boldsymbol{\Lambda}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]^T(-\boldsymbol{\delta}_{(3)})^T\right)\boxdot\mathbf{C}^T\right]\mathsf{d}\mathbf{C}\right\} \\
&= \mathsf{trace}\left\{4\left[\mathbf{A}\boxdot\left(-\boldsymbol{\delta}_{(1)}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]\boldsymbol{\Lambda}\right)\right]^T\mathsf{d}\mathbf{A}\right\} \\
&+ \mathsf{trace}\left\{4\left[\mathbf{B}\boxdot\left(-\boldsymbol{\delta}_{(2)}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\boldsymbol{\Lambda}\right)\right]^T\mathsf{d}\mathbf{B}\right\} \\
&+ \mathsf{trace}\left\{4\left[\mathbf{C}\boxdot\left(-\boldsymbol{\delta}_{(3)}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\boldsymbol{\Lambda}\right)\right]^T\mathsf{d}\mathbf{C}\right\} \\
&= \langle 4\left[\mathbf{A}\boxdot\left(-\boldsymbol{\delta}_{(1)}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B})\right]\boldsymbol{\Lambda}\right)\right], \mathsf{d}\mathbf{A}\rangle \\
&+ \langle 4\left[\mathbf{B}\boxdot\left(-\boldsymbol{\delta}_{(2)}\left[(\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\boldsymbol{\Lambda}\right)\right], \mathsf{d}\mathbf{B}\rangle \\
&+ \langle 4\left[\mathbf{C}\boxdot\left(-\boldsymbol{\delta}_{(3)}\left[(\mathbf{B}\boxdot\mathbf{B})\odot(\mathbf{A}\boxdot\mathbf{A})\right]\boldsymbol{\Lambda}\right)\right], \mathsf{d}\mathbf{C}\rangle
\end{aligned}$$

By identification with (58), it is finally found that:

$$\nabla_{\mathbf{A}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{A}} = 4\mathbf{A} \boxdot \left((-\boldsymbol{\delta}_{(1)})\left[(\mathbf{C}\boxdot\mathbf{C}) \odot (\mathbf{B}\boxdot\mathbf{B})\right]\boldsymbol{\Lambda}\right), \tag{59}$$

$$\nabla_{\mathbf{B}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{B}} = 4\mathbf{B} \boxdot \left((-\boldsymbol{\delta}_{(2)})[(\mathbf{C}\boxdot\mathbf{C}) \odot (\mathbf{A}\boxdot\mathbf{A})]\boldsymbol{\Lambda}\right), \tag{60}$$

$$\nabla_{\mathbf{C}}\mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{\partial \mathcal{H}(\mathbf{A},\mathbf{B},\mathbf{C})}{\partial \mathbf{C}} = 4\mathbf{C} \boxdot \left((-\boldsymbol{\delta}_{(3)})[(\mathbf{B}\boxdot\mathbf{B}) \odot (\mathbf{A}\boxdot\mathbf{A})]\boldsymbol{\Lambda}\right). \tag{61}$$

Appendix B. Enhanced line search

We intend to minimize the following expression with respect to μ:

$$\begin{aligned}\mathcal{H}(.) = \|\mathbf{T}_{(1)}^{I,JK} - &[(\mathbf{A}+\mu\mathbf{D_A})\boxdot(\mathbf{A}+\mu\mathbf{D_A})]\boldsymbol{\Lambda} \\ &[((\mathbf{C}+\mu\mathbf{D_C})\boxdot(\mathbf{C}+\mu\mathbf{D_C}))\odot((\mathbf{B}+\mu\mathbf{D_B})\boxdot(\mathbf{B}+\mu\mathbf{D_B}))]^T\|^2\end{aligned}$$

First, to clarify the expressions, we define some intermediate quantities:

$$\begin{aligned}
\mathbf{E_0} &= \mathbf{A}\boxdot\mathbf{A} \\
\mathbf{E_1} &= \mathbf{A}\boxdot\mathbf{D_A} + \mathbf{D_A}\boxdot\mathbf{A} = 2\mathbf{A}\boxdot\mathbf{D_A} \\
\mathbf{E_2} &= \mathbf{D_A}\boxdot\mathbf{D_A} \\
\mathbf{F_0} &= (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B}) \\
\mathbf{F_1} &= (\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{B}) + (\mathbf{D_C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{B}) \\
&\quad + (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{D_B}) + (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{D_B}\boxdot\mathbf{B}) \\
&= 2\left[(\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{B}) + (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{D_B})\right] \\
\mathbf{F_2} &= (\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{D_B}) + (\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{D_B}\boxdot\mathbf{B}) + (\mathbf{D_C}\boxdot\mathbf{C})\odot(\mathbf{B}\boxdot\mathbf{D_B}) \\
&\quad + (\mathbf{D_C}\boxdot\mathbf{C})\odot(\mathbf{D_B}\boxdot\mathbf{B}) + (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{B}) + (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B}) \\
&= 4\left[(\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{D_B})\right] + (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{B}) + (\mathbf{C}\boxdot\mathbf{C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B}) \\
\mathbf{F_3} &= (\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B}) + (\mathbf{D_C}\boxdot\mathbf{C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B}) \\
&\quad + (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{D_B}) + (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{D_B}\boxdot\mathbf{B}) \\
&= 2[(\mathbf{C}\boxdot\mathbf{D_C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B}) + (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{B}\boxdot\mathbf{D_B})] \\
\mathbf{F_4} &= (\mathbf{D_C}\boxdot\mathbf{D_C})\odot(\mathbf{D_B}\boxdot\mathbf{D_B})
\end{aligned}$$

By developing, it leads to:

$$\begin{aligned}
\mathcal{H}(.) &= \|\mathbf{T}_{(1)}^{I,JK} - [\mathbf{E_0} + \mathbf{E_1}\mu + \mathbf{E_2}\mu^2]\boldsymbol{\Lambda}[\mathbf{F_4}\mu^4 + \mathbf{F_3}\mu^3 + \mathbf{F_2}\mu^2 + \mathbf{F_1}\mu + \mathbf{F_0}]^T\|^2 \\
&= \|(-\mathbf{E_2}\boldsymbol{\Lambda}{\mathbf{F_4}}^T)\mu^6 + (-\mathbf{E_1}\boldsymbol{\Lambda}{\mathbf{F_4}}^T - \mathbf{E_2}\boldsymbol{\Lambda}{\mathbf{F_3}}^T)\mu^5 \\
&\quad + (-\mathbf{E_0}\boldsymbol{\Lambda}{\mathbf{F_4}}^T - \mathbf{E_1}\boldsymbol{\Lambda}{\mathbf{F_3}}^T - \mathbf{E_2}\boldsymbol{\Lambda}{\mathbf{F_2}}^T)\mu^4 + (-\mathbf{E_0}\boldsymbol{\Lambda}{\mathbf{F_3}}^T - \mathbf{E_1}\boldsymbol{\Lambda}{\mathbf{F_2}}^T - \mathbf{E_2}\boldsymbol{\Lambda}{\mathbf{F_1}}^T)\mu^3 \\
&\quad + (-\mathbf{E_0}\boldsymbol{\Lambda}{\mathbf{F_2}}^T - \mathbf{E_1}\boldsymbol{\Lambda}{\mathbf{F_1}}^T - \mathbf{E_2}\boldsymbol{\Lambda}{\mathbf{F_0}}^T)\mu^2 + (-\mathbf{E_0}\boldsymbol{\Lambda}{\mathbf{F_1}}^T - \mathbf{E_1}\boldsymbol{\Lambda}{\mathbf{F_0}}^T)\mu \\
&\quad + \mathbf{T}_{(1)}^{I,JK} - \mathbf{E_0}\boldsymbol{\Lambda}{\mathbf{F_0}}^T\|^2
\end{aligned}$$

Again, we define intermediate variables:

$\mathbf{K_0}$=$\mathbf{T}^{I,JK}_{(1)} - \mathbf{E_0}\mathbf{\Lambda}\mathbf{F_0}^T$
$\mathbf{K_1}$=$-\mathbf{E_0}\mathbf{\Lambda}\mathbf{F_1}^T - \mathbf{E_1}\mathbf{\Lambda}\mathbf{F_0}^T$
$\mathbf{K_2}$=$-\mathbf{E_0}\mathbf{\Lambda}\mathbf{F_2}^T - \mathbf{E_1}\mathbf{\Lambda}\mathbf{F_1}^T - \mathbf{E_2}\mathbf{\Lambda}\mathbf{F_0}^T$
$\mathbf{K_3}$=$-\mathbf{E_0}\mathbf{\Lambda}\mathbf{F_3}^T - \mathbf{E_1}\mathbf{\Lambda}\mathbf{F_2}^T - \mathbf{E_2}\mathbf{\Lambda}\mathbf{F_1}^T$
$\mathbf{K_4}$=$-\mathbf{E_0}\mathbf{\Lambda}\mathbf{F_4}^T - \mathbf{E_1}\mathbf{\Lambda}\mathbf{F_3}^T - \mathbf{E_2}\mathbf{\Lambda}\mathbf{F_2}^T$
$\mathbf{K_5}$=$-\mathbf{E_1}\mathbf{\Lambda}\mathbf{F_4}^T - \mathbf{E_2}\mathbf{\Lambda}\mathbf{F_3}^T$
$\mathbf{K_6}$=$-\mathbf{E_2}\mathbf{\Lambda}\mathbf{F_4}^T$

$$
\begin{aligned}
\mathcal{H}(.) = \mathsf{trace}\big\{&(\mathbf{K_6}\mu^6 + \mathbf{K_5}\mu^5 + \mathbf{K_4}\mu^4 + \mathbf{K_3}\mu^3 + \mathbf{K_2}\mu^2 + \mathbf{K_1}\mu + \mathbf{K_0}) \\
&\big(\mathbf{K_6}\mu^6 + \mathbf{K_5}\mu^5 + \mathbf{K_4}\mu^4 + \mathbf{K_3}\mu^3 + \mathbf{K_2}\mu^2 + \mathbf{K_1}\mu + \mathbf{K_0}\big)^T\big\} \\
= \mathsf{trace}\big\{&(\mathbf{K_6}\mathbf{K_6}^T)\mu^{12} \\
&+ (\mathbf{K_6}\mathbf{K_5}^T + \mathbf{K_5}\mathbf{K_6}^T)\mu^{11} \\
&+ (\mathbf{K_6}\mathbf{K_4}^T + \mathbf{K_5}\mathbf{K_5}^T + \mathbf{K_4}\mathbf{K_6}^T)\mu^{10} \\
&+ (\mathbf{K_6}\mathbf{K_3}^T + \mathbf{K_5}\mathbf{K_4}^T + \mathbf{K_4}\mathbf{K_5}^T + \mathbf{K_3}\mathbf{K_6}^T)\mu^9 \\
&+ (\mathbf{K_6}\mathbf{K_2}^T + \mathbf{K_5}\mathbf{K_3}^T + \mathbf{K_4}\mathbf{K_4}^T + \mathbf{K_3}\mathbf{K_5}^T + \mathbf{K_2}\mathbf{K_6}^T)\mu^8 \\
&+ (\mathbf{K_6}\mathbf{K_1}^T + \mathbf{K_5}\mathbf{K_2}^T + \mathbf{K_4}\mathbf{K_3}^T + \mathbf{K_3}\mathbf{K_4}^T + \mathbf{K_2}\mathbf{K_5}^T + \mathbf{K_1}\mathbf{K_6}^T)\mu^7 \\
&+ (\mathbf{K_6}\mathbf{K_0}^T + \mathbf{K_5}\mathbf{K_1}^T + \mathbf{K_4}\mathbf{K_2}^T + \mathbf{K_3}\mathbf{K_3}^T + \mathbf{K_2}\mathbf{K_4}^T + \mathbf{K_1}\mathbf{K_5}^T + \mathbf{K_0}\mathbf{K_6}^T)\mu^6 \\
&+ (\mathbf{K_5}\mathbf{K_0}^T + \mathbf{K_4}\mathbf{K_1}^T + \mathbf{K_3}\mathbf{K_2}^T + \mathbf{K_2}\mathbf{K_3}^T + \mathbf{K_1}\mathbf{K_4}^T + \mathbf{K_0}\mathbf{K_5}^T)\mu^5 \\
&+ (\mathbf{K_4}\mathbf{K_0}^T + \mathbf{K_3}\mathbf{K_1}^T + \mathbf{K_2}\mathbf{K_2}^T + \mathbf{K_1}\mathbf{K_3}^T + \mathbf{K_0}\mathbf{K_4}^T)\mu^4 \\
&+ \mathbf{K_3}\mathbf{K_0}^T + \mathbf{K_2}\mathbf{K_1}^T + \mathbf{K_1}\mathbf{K_2}^T + \mathbf{K_0}\mathbf{K_3}^T)\mu^3 \\
&+ (\mathbf{K_2}\mathbf{K_0}^T + \mathbf{K_1}\mathbf{K_1}^T + \mathbf{K_0}\mathbf{K_2}^T)\mu^2 \\
&+ (\mathbf{K_1}\mathbf{K_0}^T + \mathbf{K_0}\mathbf{K_1}^T)\mu \\
&+ \mathbf{K_0}\mathbf{K_0}^T\big\}
\end{aligned}
$$

The thirteen coefficients $a_0, \ldots, a_{12}$ are finally obtained by identification.

References

[1] Boyd, S., Vandenberghe, L., Mar. 2004. Convex optimization. Cambridge University Press.

[2] Bro, R., 1997. Parafac: tutorial and applications. Chemometr. Intell. Lab. 38, 149–171.

[3] Bro, R., 1998. Multi-way analysis in the food industry: models, algorithms and applications. Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands.

[4] Carroll, P., Chang, J. J., 1970. Analysis of individual differences in multi-dimensional scaling via n-way generalization of Eckart-Young decomposition. Psychometrika 35, 283–319.

[5] Cichocki, A., Zdunek, R., Phan, A. H., Amari, S. I., 2009. Non negative matrix and tensor factorizations: Application to exploratory multi-way data analysis and blind separation. Wiley.

[6] Comon, P., Aug. 31 - Sep. 3 2009. Tensors, usefulness and unexpected properties. In: 15th IEEE Workshop on Statistical Signal Processing (SSP'09). Cardiff, UK, pp. 781–788, keynote.

[7] Comon, P., Luciani, X., De Almeida, A. L. F., Aug. 2009. Tensor decompositions, alternating least squares and other tales. Jour. Chemometrics 23, 393–405.

[8] Franc, A., 1992. Etude algébrique des multi-tableaux : apport de l'algèbre tensorielle. Phd thesis, University of Montepellier II, Montpellier, France.

[9] Ghennioui, H., Thirion-Moreau, N., Moreau, E., Aboutajdine, D., June 2010. Gradient based joint block diagonalization algorithms: application to blind separation of fir convolutive sources mixtures. Eurasip Signal Processing 90 (6), 1836–1849, doi:10.1016/j.sigpro.2009.12.002.

[10] Harshman, R. A., 1970. Foundation of the Parafac procedure: models and conditions for an explanatory multimodal factor analysis. UCLA Working papers in phonetics 16, 1–84.

[11] Hitchcock, F. L., 1927. The expression of a tensor or a polyadic as a sum of products. J. Math. and Phys. 6, 165–189.

[12] Jiang, J. H., Wu, H. L., Li, Y., Yu, R. Q., 1999. Alternating coupled vectors resolution (acover) method for trilinear analysis of three-way data. Journal of Chemometrics 13 (6), 557–578.

[13] Kolda, T. G., Bader, B. W., Sep. 2009. Tensor decompositions and applications. Siam Review 51 (3), 455–500.

[14] Lickteig, T., 1985. Typical tensorial rank. Linear Algebra Appl. 69, 95–120.

[15] Lim, L. H., Comon, P., Aug. 2009. Nonnegative approximations of nonnegative tensors. Jour. Chemometrics 23, 432–441.

[16] Luciani, X., Mounier, S., Redon, R., Bois, A., 2009. A simple correction method of inner filter effects affecting FEEM and its application to the Parafac decomposition. Chemometrics and Intel. Lab. Syst. 96 (2), 227–238.

[17] Luenberger, D. G., 1969. Optimization by vector space methods. Wiley.

[18] Luenberger, D. G., Ye, Y., 2008. Linear and non linear programming, 3rd Edition. Wiley.

[19] Magnus, J. R., Neudecker, H., 2007. Matrix differential calculus with applications in statistics and econometrics, 3rd Edition. Wiley.

[20] Mounier, S., Zhao, H., Garnier, C., Redon, R., August 2010. Copper complexing properties of dissolved organic matter: Parafac treatment of fluorescence quenching. Biochemistry.

[21] Paatero, P., 1997. A weighted non-negative least squares algorithm for three-way Parafac factor analysis. Chemometrics Intell. Lab. Systems 38 (2), 223–242.

[22] Paatero, P., Dec. 1999. The multilinear engine - a table-driven, least squares program for solving multi-linear problems, including the n-way parallel factor analysis model. J. Comput. Graph. Stat. 8 (4), 854–888.

[23] Paatero, P., 2000. Construction and analysis of degenerate Parafac models. J. Chemometrics 14 (3), 285–299.

[24] Polak, E., 1997. Optimization algorithms and consistent approximations. Springer.

[25] Rajih, M., Comon, P., Harshman, R. A., Sep. 2008. Enhanced line search: a novel method to accelerate Parafac. SIAM Journal of Matrix analysis applications 30 (3), 1148–1171.

[26] Shewchuk, J. R., Aug. 1994. introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

[27] Smilde, A., Bro, R., Geladi, P., 2004. Multi-Way Analysis with applications in the chemical sciences. Wiley.

[28] Tomasi, G., Bro, R., Feb. 2005. Parafac and missing values. Chemometrics Intell. Lab. Systems 75 (2), 163–180.

[29] Tomasi, G., Bro, R., Apr. 2006. A comparison of algorithms for fitting the Parafac model. Computational Statistics & Data Analysis 50 (7), 1700–1734, elsevier Sciences Publishers B. V.

[30] Zhang, Q., Wang, H., Plemmons, R., Pauca, P., Dec. 2008. Tensors methods for hyperspectral data processing: a space object identification study. Journal of Optical Society of America A 25 (12), 3001–3012.

Method	Cost per iteration	
	General case	Case $I = J = K$
ALS (without positivity constraint)	$7(JK + KI + IJ)F^2 + 3IJKF$	$21(IF)^2 + 3FI^3$
ALS-Cichocki	$3IJKF$	$3FI^3$
Gradient	$6IJKF$	$6FI^3$
Nonlinear conjugate gradient	$6IJKF + 2(I + J + K)F^2$	$6FI^3 + 6IF^2$
Gauss-Newton (BFGS)	$(I + J + K)^3F^3$	$27I^3F^3$
BFGS using (49)	$4(I + J + K)^2F^2$	$36I^2F^2$
Gauss-Newton (DFP)	$4(I + J + K)^2F^2$	$36I^2F^2$
Levenberg-Marquardt	$(I + J + K)^3F^3$	$27I^3F^3$
Preconditioned nonlinear conjugate gradient	$(I + J + K)^3F^3$	$27I^3F^3$

Table 1: Algorithmic complexity of various algorithms

Method	Cost per iteration	
	General case	Case $I = J = K$
ALS without positivity constraint	$7(JK + KI + IJ)F^2 + 11IJKF + 9IJK$	$21I^2F^2 + 11I^3F + 9I^3$
Gradient	$49KJI^2 + 13IJKF$	$49I^4 + 13I^3F$
Nonlinear conjugate gradient	$2(I + J + K)F^2 + 49KJI^2 + 13IJKF$	$6IF^2 + 49I^4 + 13I^3F$
Gauss-Newton (BFGS)	$(I + J + K)^3F^3 + 49KJI^2 + 13IJKF$	$27I^3F^3 + 49I^4 + 13I^3F$
Gauss-Newton (BFGS with (49))	$4(I + J + K)^2F^2 + 49KJI^2 + 13IJKF$	$36I^2F^2 + 49I^4 + 13I^3F$
Gauss-Newton (DFP)	$4(I + J + K)^2F^2 + 49KJI^2 + 13IJKF$	$36I^2F^2 + 49I^4 + 13I^3F$
Levenberg-Marquardt	$(I + J + K)^3F^3 + 49KJI^2 + 13IJKF$	$27I^3F^3 + 49I^4 + 13I^3F$
Preconditioned conjugate gradient	$(I + J + K)^3F^3 + 49KJI^2 + 13IJKF$	$27I^3F^3 + 49I^4 + 13I^3F$

Table 2: Algorithmic complexity for the ELS version of the different algorithms

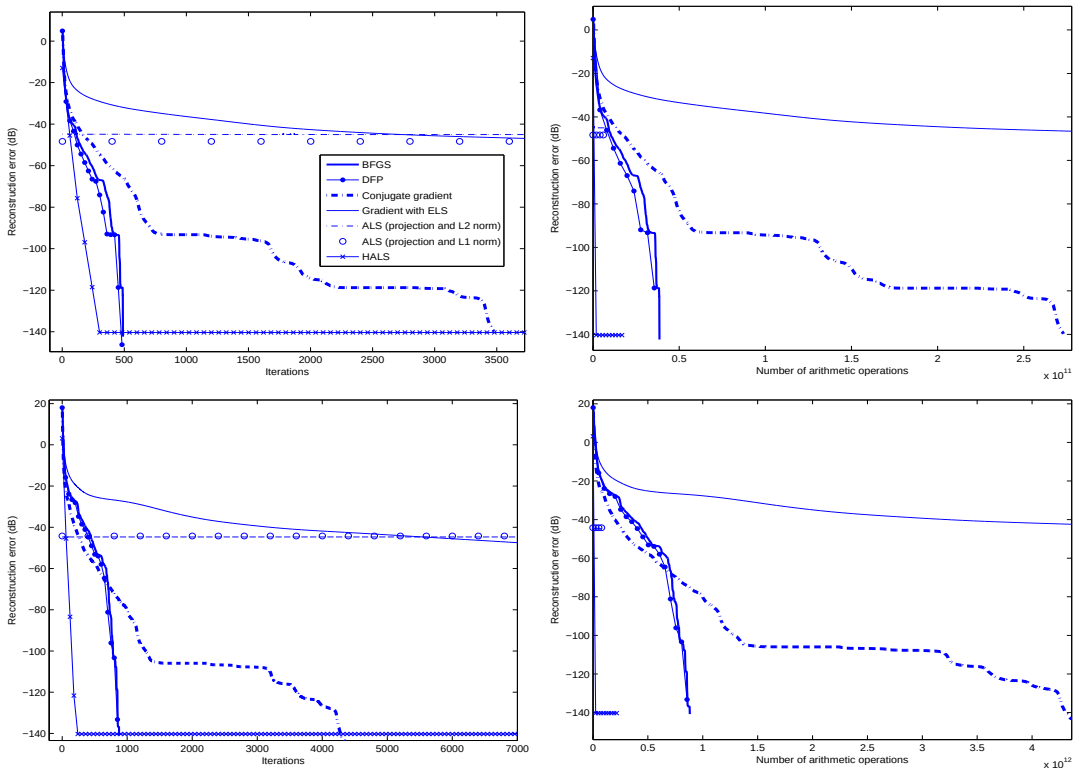


Figure 1: Reconstruction error (dB) versus the number of iterations (left) using a nonnegative $71 \times 47 \times 10$ tensor (top left), a nonnegative $71 \times 47 \times 128$ tensor (bottom left). Reconstruction error (dB) versus the number of arithmetic operations (right) using a nonnegative $71 \times 47 \times 10$ tensor (top right), a nonnegative $71 \times 47 \times 128$ tensor (bottom right). The same legend is used for the 4 charts. We should pay attention to the fact that a small reconstruction error does not mean that loading matrices are correctly estimated; in fact, the number of components should also be correctly detected (cf. Figs. 4-5).

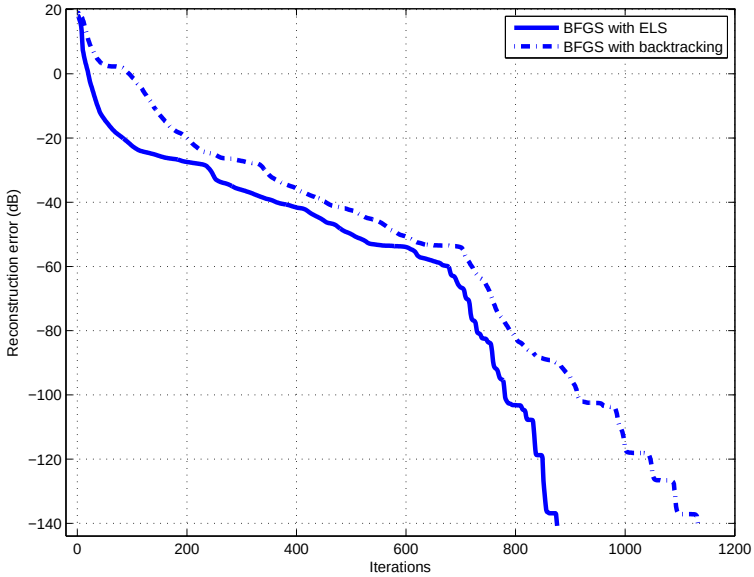


Figure 2: Comparison BFGS with backtracking (ELS every 10 iterations) and BFGS with ELS at each iteration: reconstruction error as a function of the number of arithmetic operations.

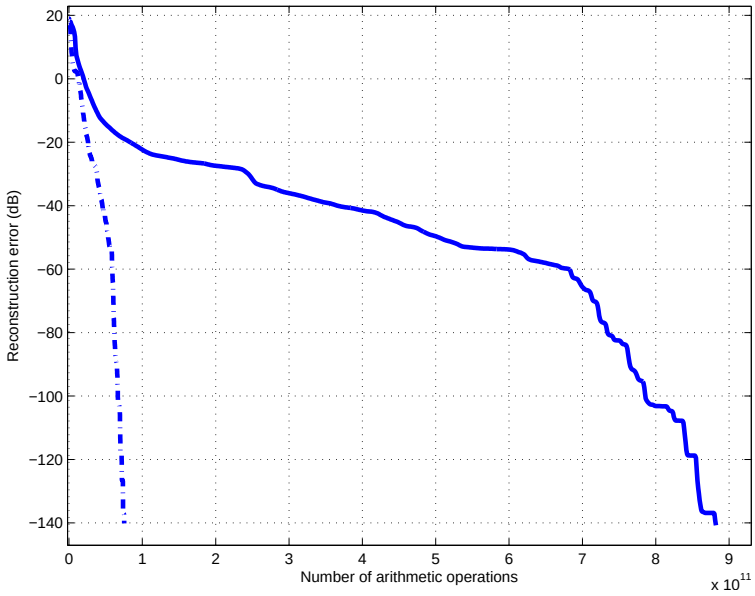


Figure 3: Comparison BFGS with backtracking (ELS every 10 iterations) and BFGS with ELS at each iteration: reconstruction error as a function of complexity

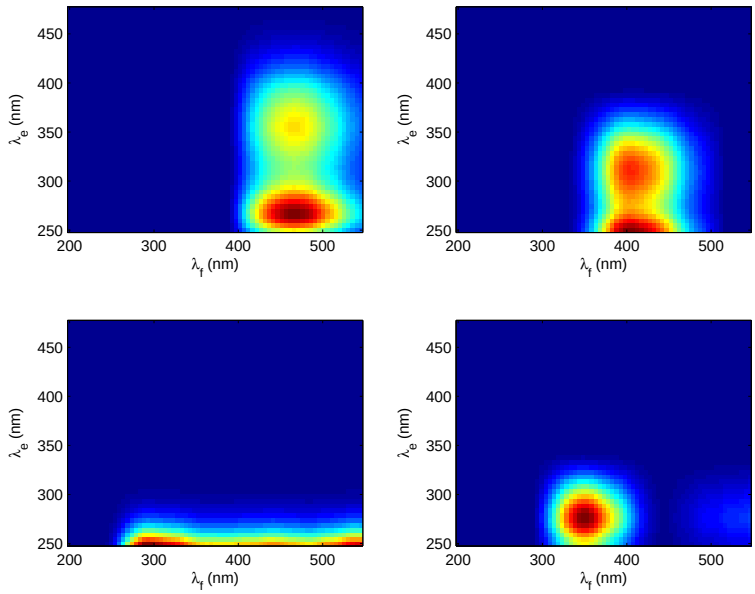


Figure 4: Case 4 factors, assuming $F = 4$, the 4 estimated emission-excitation images that perfectly fit the emission-excitation images of the 4 considered fluorophores.

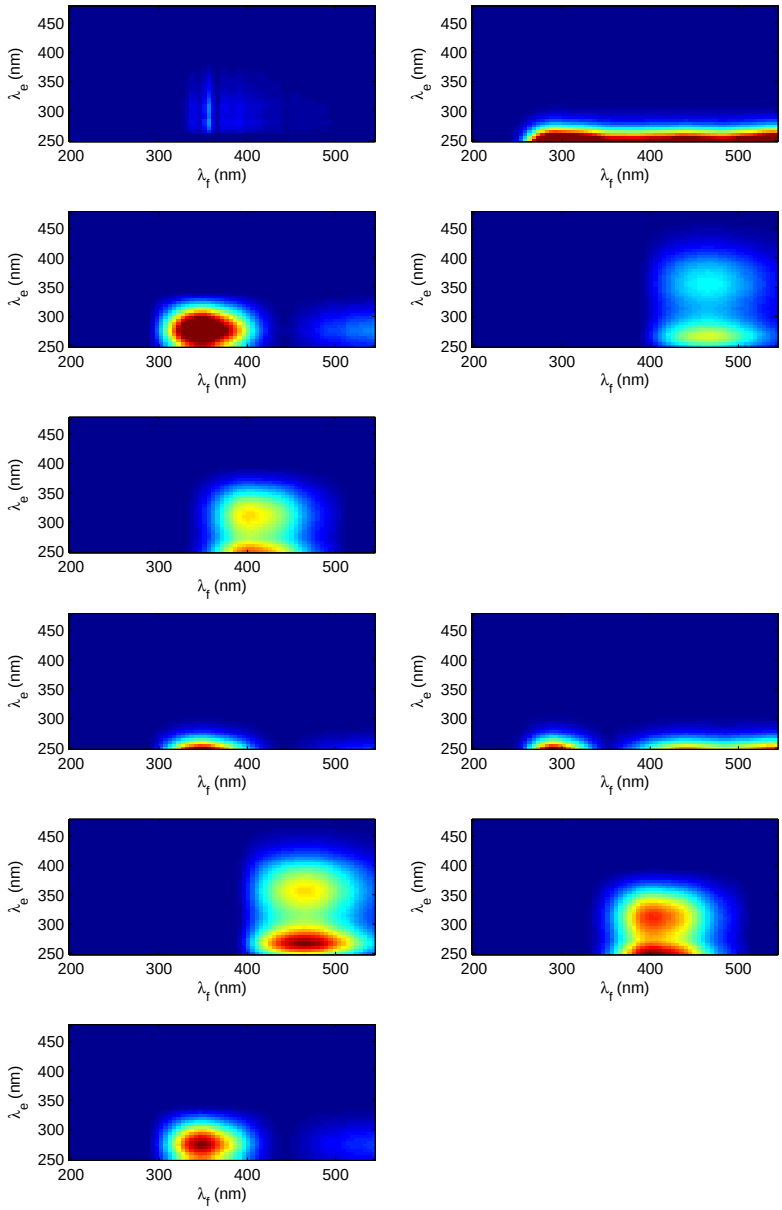


Figure 5: Case 4 factors, assuming $F = 5$, the 5 estimated emission-excitation images using the conjugate gradient algorithm with positivity constraint (top) and the ALS algorithm with positivity constraint projection based (bottom).