UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Tensor approximation by block term decomposition

Dissertation presented by
Guillaume OLIKIER

for obtaining the Master's degree in
Mathematical Engineering

Supervisors
Pierre-Antoine ABSIL, Lieven DE LATHAUWER

Reader
Yurii NESTEROV

Academic year 2016-2017

Abstract

Higher-order tensors have become a powerful tool in many areas of applied mathematics such as statistics or scientific computing. They also have found many applications in signal processing and machine learning. As suggested by the literature, higher-order tensors draw their power notably from the many tensor decompositions among which the block term decomposition holds a central place. The purpose of this master's thesis is to focus on the computation of the best approximation in the least-squares sense of a given third-order tensor by a block term decomposition. Using variable projection, the tensor approximation problem is expressed as a minimization of a cost function on a product of Stiefel manifolds. In this master's thesis, I apply two first-order algorithms from the framework of optimization on matrix manifolds to minimize this cost function. I investigate the performance of these two new methods and compare them with the already available ones.

Acknowledgments

This master's thesis was generated by two experts from two different disciplines. Pierre-Antoine Absil (Université catholique de Louvain) is expert in optimization on matrix manifolds. The expertise field of Lieven De Lathauwer (Katholieke Universiteit Leuven) is numerical multilinear algebra. They both have set up this project and so I would like to express my gratitude to them for making this master's thesis possible.

I would like to first thank Pierre-Antoine Absil for his constant support and his availability throughout the academic year. His advices were invaluable help for the production of this work, including during the debug periods.

Special thanks also go to Lieven De Lathauwer who helped me getting acquainted with the world of numerical multilinear algebra through his lectures in Leuven.

Yurii Nesterov (Université catholique de Louvain) and Mariya Ishteva (Vrije Universiteit Brussel) gave me helpful advices. Many thanks go to them as well.

Finally, I would like to thank Nico Vervliet for the overview of the available methods to compute block term decompositions.

Contents

Notation conventions

Sets of numbers

$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	set of integers
$\mathbb{N}$	set of nonnegative integers
$\mathbb{N}_*$	$\mathbb{N} \setminus \{0\}$ (set of positive integers)
$\{a, \dots, b\}$ (with $a, b \in \mathbb{Z}$ and $a \leq b$)	$\{z \in \mathbb{Z} : a \leq z \leq b\}$

Sets

$\times$	Cartesian product or symbol used to give the size of a tensor (the meaning is always clear from the context)
$\prod_{i=1}^{n} S_i$ (given $n \in \mathbb{N}_*$ and sets $S_1, \dots, S_n$)	n-ary Cartesian product
S^n (given a set S and $n \in \mathbb{N}_*$)	n-ary Cartesian power of the set S

Matrices

$\delta_{i,j} := \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$ for all $i, j \in \mathbb{Z}$	Kronecker symbol
$\mathbf{I}_n := [\delta_{i,j}]_{i,j=1}^{n}$ (with $n \in \mathbb{N}_*$)	identity matrix of size n
$\mathrm{diag}(a_1, \dots, a_n)$ (with $n \in \mathbb{N}_*$)	$[a_i \delta_{i,j}]_{i,j=1}^{n}$ (diagonal matrix)
$\mathrm{St}(p, n)$ (with $p, n \in \mathbb{N}_*$ and $p \leq n$)	$\{\mathbf{U} \in \mathbb{R}^{n \times p} : \mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}_p\}$ (Stiefel manifold)
O_n (with $n \in \mathbb{N}_*$)	$\mathrm{St}(n, n)$ (orthogonal group)
$\otimes$	Kronecker product

The sections, subsections or paragraphs with a star (*) in their title are given for further information; they can be skipped without affecting the flow of the other parts of the text.

Introduction

Matrix theory is one of the cornerstones of applied mathematics. Matrices are indeed a fundamental tool in numerical analysis, in optimization, in discrete mathematics and, more broadly, in most disciplines that use quantitative methods.

An important part of matrix theory is devoted to the study of the different matrix decompositions, among which is the Singular Value Decomposition (SVD) about which Golub and Van Loan [1, section 2.4] wrote: *The practical and theoretical importance of the SVD is hard to overestimate. It has a prominent role to play in data analysis and in the characterization of the many matrix "nearness problems."* Theoretically, the SVD has several applications, to compute the Moore–Penrose pseudoinverse of a matrix, to get the polar decomposition of a matrix, to compute the principal angles between two linear subspaces, to approximate a matrix by a matrix with smaller rank, to name but a few examples. This last application is, for instance, very important in practice in data compression or denoising.

However, matrices have inherent limitations. Foremost, the structure of a matrix only allows the description of binary interactions. For example, one can stack the second order partial derivatives of a function in a matrix but it does not seem natural to do so for the third-order partial derivatives. Working with matrices also present deeper drawbacks. For instance, the decomposition of a given matrix as a sum of rank-one matrices is never unique unless strong constraints such as orthogonality or triangularity, which are not necessarily relevant for the considered application, are imposed.

Higher-order tensors have been introduced to overcome the drawbacks described above, at least to allow the description of multiple interactions. A vector is a tensor of order 1, i.e., a one-dimensional array. In the same way, a matrix is a tensor of order 2, i.e., a two-dimensional array. In general, a tensor of order N is a N-dimensional array, i.e., an array whose entries are indexed by N indices, each of them describing a mode. A higher-order tensor is a tensor of order greater than 2.

Given this definition, it is clear that higher-order tensors are the natural tool to process data with multiple interactions. For instance, the third-order partial derivatives of a function can be stacked naturally in a (cubic) third-order tensor, as well as the third-order moments of a random vector.

However, as written in [2], higher-order tensors are more than "matrices with more indices"; multilinear algebra is much more structurally rich than linear algebra. For example, the other limitation inherent to matrices mentioned earlier, namely the non-uniqueness of the rank-one terms decomposition, almost no more exists with higher-order tensors. Indeed, surprisingly at first sight, the decomposition of a higher-order tensor as a sum of rank-one tensors is essentially unique under mild conditions. This suggests that higher-order tensors could be a very useful tool in blind source separation, especially in independent component analysis.

In fact, higher-order tensors have now become a powerful tool used, e.g., in statistics or scientific computing. They also have found many applications in signal processing and machine learning [2, 3].

As said earlier, multilinear algebra is structurally more rich and complex than linear algebra. Many simple matrix concepts cannot be extended to higher-order tensors in a unique way. One of the most important examples is probably the concept of rank. The rank of a matrix is

a well understood quantity. It can be defined in several equivalent ways but the respective generalizations of these definitions to higher-order tensors are not equivalent. Actually, two different definitions of the rank of a tensor, respectively called the rank and the multilinear rank, are used in practice.

It is easy to imagine that the absence of a unique definition for a concept as basic as the rank of a tensor will have important implications in theory. For instance, the SVD, which reveals the rank of a matrix, cannot be generalized to higher-order tensors in a unique way. In fact, the two definitions of the rank of a tensor will lead each to a different higher-order generalization of the SVD. The extension of the SVD based on the rank will lead to the Canonical Polyadic Decomposition (CPD) while the other one based on the multilinear rank will lead to the Tucker Decomposition (TKD). Actually, the CPD generalizes any rank-revealing matrix decomposition.

The CPD and the TKD are both particular cases of a recently introduced tensor decomposition called the Block Term Decomposition (BTD) [4, 5, 6] developed by Lieven De Lathauwer. From this point of view, the BTD has a unifying character among the different tensor decompositions. Moreover, its great flexibility completely opens new possibilities in the applications. In blind source separation, for instance, BTDs admit the modeling of more complex signal components than CPD [2].

My master's thesis focuses on the computation of the BTD of a given third-order tensor or, more generally, on the approximation of a given third-order tensor by a BTD. Using variable projection, I will show that this tensor approximation problem can be expressed as a minimization of a cost function on a product of Stiefel manifolds. Then, I will deal with this minimization problem using the gradient algorithm and the conjugate gradients algorithm in their manifold setting.

This project can be seen as a continuation of the work [7] performed by Mariya Ishteva in which the trust-region scheme had been successfully applied to approximate a given third-order tensor by a TKD. I will start by following the same steps as in [7] but we will see that the situation is not as simple for the BTD as for the TKD.

This master's thesis is structured as a report. The report comprises three chapters. The background in tensor computations needed to understand the problem is introduced in chapter one. I formulate the BTD approximation problem and present the algorithms in chapter two. Numerical results are presented in chapter three. Finally, I will conclude by highlighting the differences between the two methods developed in this work and the already available ones. I will also propose some ideas that could be worth exploring as opportunities for future research.

Chapter 1

Tensor computations

In this chapter, we give an overview of the main manipulations and operations that can be performed with tensors. The goal is not to be exhaustive but rather to introduce the background needed for the purpose of this work, even if some additional material is occasionally presented to give the reader a broader perspective on tensor computations. Our main references for this chapter are the overview papers [2, 3].

In the first section, we define formally what is a tensor and then introduce some useful representations of tensors. In the second section, we define algebraic operations for tensors defined over a field and study how those operations interact with the different tensor representations introduced in the first section. In the third section, we restrict ourselves to real tensors and introduce an inner product for these tensors. In the fourth section, we introduce three important tensor decompositions, namely the TKD, the CPD and finally the BTD.

In this chapter, tensors are successively viewed as multiway arrays used to store data described by possibly more than two indices, then as algebraic objects, and then as elements of a Euclidean space. In this last setting, tensor decompositions can be introduced.

1.1 Basic definitions

Formal definition Let $N \in \mathbb{N}_*$, $I_1, \ldots, I_N \in \mathbb{N}_*$, and S be an arbitrary set. A *tensor of order* N (or a Nth-order tensor) over S is a function from $\prod_{n=1}^{N} \{1, \ldots, I_n\}$ to S; N and $(I_1, \ldots, I_N)$ are respectively called the *order* and the *size* of the tensor. We let $S^{I_1 \times \cdots \times I_N}$ denote the set of Nth-order tensors of size $(I_1, \ldots, I_N)$ over S. In practice, we always identify a tensor to a multiway array. For instance, a third-order tensor of size $(5, 4, 3)$ is identified with a three-dimensional array like the one drawn in Figure 1.1.

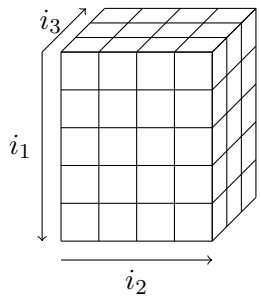


Figure 1.1: Third-order tensor of size $(5, 4, 3)$. Note the direction of evolution of each index.

Vectors and matrices Important examples of tensors are vectors and matrices which are respectively first-order and second-order tensors. Let us insist on the definition. Given $I \in \mathbb{N}_*$, a vector $\mathbf{v} \in S^I$ is a function from $\{1, \dots, I\}$ to S that we identify to the list $(\mathbf{v}(1), \dots, \mathbf{v}(I))$; I is called the *length* of the vector. In the same way, given $I, J \in \mathbb{N}_*$, a matrix $\mathbf{M} \in S^{I \times J}$ is a function from $\{1, \dots, I\} \times \{1, \dots, J\}$ to S that we identify to the rectangular array

$$\begin{bmatrix} \mathbf{M}(1,1) & \dots & \mathbf{M}(1,J) \\ \vdots & & \vdots \\ \mathbf{M}(I,1) & \dots & \mathbf{M}(I,J) \end{bmatrix}$$

which is also denoted $[\mathbf{M}(i,j)]_{i,j=1}^{I,J}$ in a more concise form. A matrix of size (I, J) is sometimes called a $I \times J$ matrix. In the sequel, vectors of length I are identified to matrices of size $(I, 1)$. By higher-order tensor, we mean tensor of order greater than or equal to three.

Notation In order to improve readability, vectors are written in bold-face lower-case ($\mathbf{a}$, $\mathbf{b}$, ...), matrices in bold-face capitals ($\mathbf{A}$, $\mathbf{B}$, ...) and higher-order tensors in calligraphic letters ($\mathcal{A}$, $\mathcal{B}$, ...). In the sequel, when we write $S^{I_1 \times \dots \times I_N}$, we assume implicitly that $N \in \mathbb{N}_*$ and $I_1, \dots, I_N \in \mathbb{N}_*$.

Examples Let us show that higher-order tensors appear naturally in various situations.

Higher-order partial derivatives Let $n > 1$ be an integer and $f : \mathbb{R}^n \to \mathbb{R}$ be an infinitely differentiable function. It is well known that the n first-order partial derivatives of f can be stacked in a vector of length n which is called the gradient of f. In the same way, the second-order partial derivatives of f are stacked in a matrix of size (n, n) called the Hessian of f. In general, the partial derivatives of order $N \in \mathbb{N}_*$ of f can be stacked in a Nth-order tensor of size $(n, \dots, n)$.

Higher-order moments Let $n > 1$ be an integer, $(\Omega, \mathcal{F}, P)$ be a probability space, and $\mathbf{x} : \Omega \to \mathbb{R}^n$ be a random vector. The first-order moments of $\mathbf{x}$, $\mathrm{E}[x_1], \dots, \mathrm{E}[x_n]$, are stored in the vector $\mathrm{E}[\mathbf{x}] := (\mathrm{E}[x_1], \dots, \mathrm{E}[x_n])$. In the same way, the second-order moments of $\mathbf{x}$, $\mathrm{E}[x_i x_j]$ for all $(i, j) \in \{1, \dots, n\}^2$, are stacked in the matrix $\mathrm{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]$. In general, the moments of order $N \in \mathbb{N}_*$ of $\mathbf{x}$, $\mathrm{E}[\prod_{n=1}^N x_{i_n}]$ for all $(i_1, \dots, i_N) \in \{1, \dots, n\}^N$, can be stacked in a Nth-order tensor of size $(n, \dots, n)$.

Image processing From the mathematical point of view, an image is a function $f : U \to C$, where $U \subset \mathbb{R}^2$ and C is a vector space called the color space.[1] In order to represent an image on a computer, it is necessary to discretize both U and C.

In practice, U is very often a rectangle of the plane that is discretized by being divided into small rectangles. Each rectangle contains a reference point, e.g., its center, which is called a pixel and can be represented using integer coordinates. In the discrete model, all the points in a given rectangle have the same color which is the color at the correpondig pixel, i.e., the value of f at this pixel. At this stage, the image can be represented by a matrix in which each entry is the value of f at the corresponding pixel.

It remains to discretize the color space C; this process is called the quantization. For a grayscale image, it is enough to give the level of gray which is a single number. However, for a color image, it is necessary to precise for each pixel the level of each of three basis colors (e.g., red, green and blue). Thus, we need to store a triple of numbers at each pixel. A possible way to store this information is to use three matrices (one for each basis color) and to work separately

[1] Our reference for this example is the web site `http://blog.kleinproject.org/?p=588` and the reference therein [8, chapter 6].

on each color. "However, processing by components does not take advantage of the correlation in color information that exists between the components of an image." [8, p.141] For this reason, it is better to represent the image as a third-order tensor (whose third index can take the values 1, 2 and 3) and to process the image using multilinear tools.

Symmetric tensors The tensors defined in the two first examples are symmetric. A Nth-order tensor $\mathcal{T}$ of size $(I, \ldots, I)$ is *symmetric* if and only if

$$\mathcal{T}(i_1, \ldots, i_N) = \mathcal{T}(j_1, \ldots, j_N)$$

for every $(i_1, \ldots, i_N) \in \{1, \ldots, I\}^N$ and every permutation $(j_1, \ldots, j_N)$ of $(i_1, \ldots, i_N)$.

Multiple interactions The third example above shows that some kinds of data are, by nature, the result of interactions between more than two components. The best way to process those kinds of data is to use objects that respect their multiway structure.

Mode-n vectors In matrix computations, it is often convenient to see a given matrix as a matrix of submatrices called *blocks*. Let us recall the standard notation (see, e.g., [1, subsections 1.1.7 & 1.1.8]). Let $p, q > 1$ be two integers, $m_1, \ldots, m_p, n_1, \ldots, n_q \in \mathbb{N}_*$, $m := \sum_{i=1}^{p} m_i$, and $n := \sum_{i=1}^{q} n_i$. A matrix of size (m, n) can be viewed as a matrix of size (p, q) in which each entry $(i, j) \in \{1, \ldots, p\} \times \{1, \ldots, q\}$ is a matrix of size (m_i, n_j).

Particularly useful blocks decompositions for matrices are rows and columns decompositions. The rows decomposition of $\mathbf{M} \in S^{I \times J}$ is

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}(1, :) \\ \vdots \\ \mathbf{M}(I, :) \end{bmatrix}$$

while the columns decomposition of $\mathbf{M}$ is

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}(:, 1) & \ldots & \mathbf{M}(:, J) \end{bmatrix}.$$

Given $i \in \{1, \ldots, I\}$ and $j \in \{1, \ldots, J\}$, $\mathbf{M}(i, :) \in S^{1 \times J}$ is called the ith row of $\mathbf{M}$ while $\mathbf{M}(:, j) \in S^{I \times 1}$ is called the jth column of $\mathbf{M}$. More generally, we let $\mathbf{M}(i_1 : i_2, j_1 : j_2)$ denote the submatrix $[\mathbf{M}(i, j)]_{i_1 \leq i \leq i_2, j_1 \leq j \leq j_2}$ of $\mathbf{M}$; this is the standard Matlab notation.

The so-called mode-n vectors, $n \in \{1, \ldots, N\}$, of a Nth-order tensor are the higher-order generalization of the rows and the columns of a matrix. They are obtained by varying the nth index while keeping the other indices fixed. Formally, the set of mode-n vectors of $\mathcal{A} \in S^{I_1 \times \cdots \times I_N}$ is

$$\{\mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) : (i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) \in \prod_{\substack{m=1 \\ m \neq n}}^{N} \{1, \ldots, I_m\}\} \subset S^{I_n},$$

where

$$\begin{aligned} &\mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) := \\ &(\mathcal{A}(i_1, \ldots, i_{n-1}, 1, i_{n+1}, \ldots, i_N), \ldots, \mathcal{A}(i_1, \ldots, i_{n-1}, I_n, i_{n+1}, \ldots, i_N)) \end{aligned}$$

for each $(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) \in \prod_{\substack{m=1 \\ m \neq n}}^{N} \{1, \ldots, I_m\}$. The mode-$n$ vectors of a third-order tensor of size $(5, 4, 3)$ are drawn in Figure 1.2.

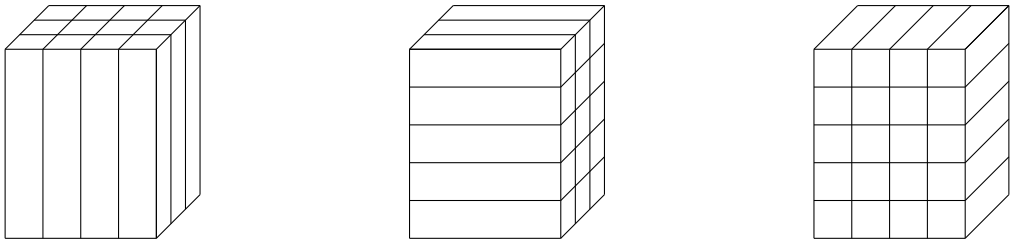


Figure 1.2: Mode-n vectors, $n = 1, 2, 3$ from left to right, of a third-order tensor of size $(5, 4, 3)$.

Vectorization and matricization Manipulating higher-order tensors is sometimes facilitated by using vector and matrix representations of them. Vectorizing (resp. matricizing) a tensor consists of stacking its mode-n vectors into a vector (resp. into a matrix). The only thing needed to complete the definition of a vectorization or a matricization is to precise the order in which the mode-n vectors are stacked. Several choices are possible. In this text, we follow the convention used in [9, 10, 7]. Let $\mathcal{A}$ be a Nth-order tensor of size $(I_1, \ldots, I_N)$.

The vectorization of $\mathcal{A}$, denoted $\text{vec}(\mathcal{A})$, is the vector of length $\prod_{n=1}^{N} I_n$ defined as follows:

$$(\text{vec}(\mathcal{A}))\left(\sum_{n=1}^{N-1}(i_n - 1)\prod_{m=n+1}^{N} I_m + i_N\right) := \mathcal{A}(i_1, \ldots, i_N)$$

for each $(i_1, \ldots, i_N) \in \prod_{n=1}^{N}\{1, \ldots, I_n\}$. For instance, the vectorization of a $I_1 \times I_2 \times I_3$ tensor $\mathcal{A}$ is

$$(\text{vec}(\mathcal{A}))(I_2 I_3(i_1 - 1) + I_3(i_2 - 1) + i_3) := \mathcal{A}(i_1, i_2, i_3)$$

for each $(i_1, i_2, i_3) \in \prod_{n=1}^{3}\{1, \ldots, I_n\}$. In other words, we construct $\text{vec}(\mathcal{A})$ by stacking the mode-3 vectors of $\mathcal{A}$ as shown on the left schema of Figure 1.3. In Tensorlab, the vectorization of a third-order tensor `A` is given by `tens2vec(A,[3,2,1])`.

For each $n \in \{1, \ldots, N\}$, we define the following matrix representation of $\mathcal{A}$ [9, Definition 1]:

$$\mathbf{A}_{(n)}\left(i_n, 1 + \sum_{k=1}^{n-1}(i_k - 1)\prod_{l=k+1}^{n-1} I_l + \left(\prod_{m=1}^{n-1} I_m\right)\sum_{k=n+1}^{N}(i_k - 1)\prod_{l=k+1}^{N} I_l\right) := \mathcal{A}(i_1, \ldots, i_N)$$

for each $(i_1, \ldots, i_N) \in \prod_{n=1}^{N}\{1, \ldots, I_n\}$. Thus, $\mathbf{A}_{(n)}$ is a matrix of size $(I_n, \prod_{\substack{m=1 \\ m\neq n}}^{N} I_m)$ whose columns are the mode-n vectors of $\mathcal{A}$. The indices that grow the faster are $i_{n-1}, \ldots, i_1, i_N, \ldots, i_{n+1}$ in this order. For instance, for $N = 3$,

$$(\mathbf{A}_{(1)})(i_1, I_3(i_2 - 1) + i_3) = (\mathbf{A}_{(2)})(i_2, I_1(i_3 - 1) + i_1) = (\mathbf{A}_{(3)})(i_3, I_2(i_1 - 1) + i_2) = \mathcal{A}(i_1, i_2, i_3)$$

for each $(i_1, i_2, i_3) \in \prod_{n=1}^{3}\{1, \ldots, I_n\}$. The construction of these matrices is shown in the right diagram of Figure 1.3. In Tensorlab, given a third-order tensor `A`, the matrices $\mathbf{A}_{(1)}$, $\mathbf{A}_{(2)}$, $\mathbf{A}_{(3)}$ are respectively given by `tens2mat(A,1,[3,2])`, `tens2mat(A,2,[1,3])` and `tens2mat(A,3,[2,1])`.

1.2 Tensor algebra

Tensors are not only useful because they allow the storage of data but also because we can perform algebraic operations with them. Indeed, in all applications, tensors are defined over a field.

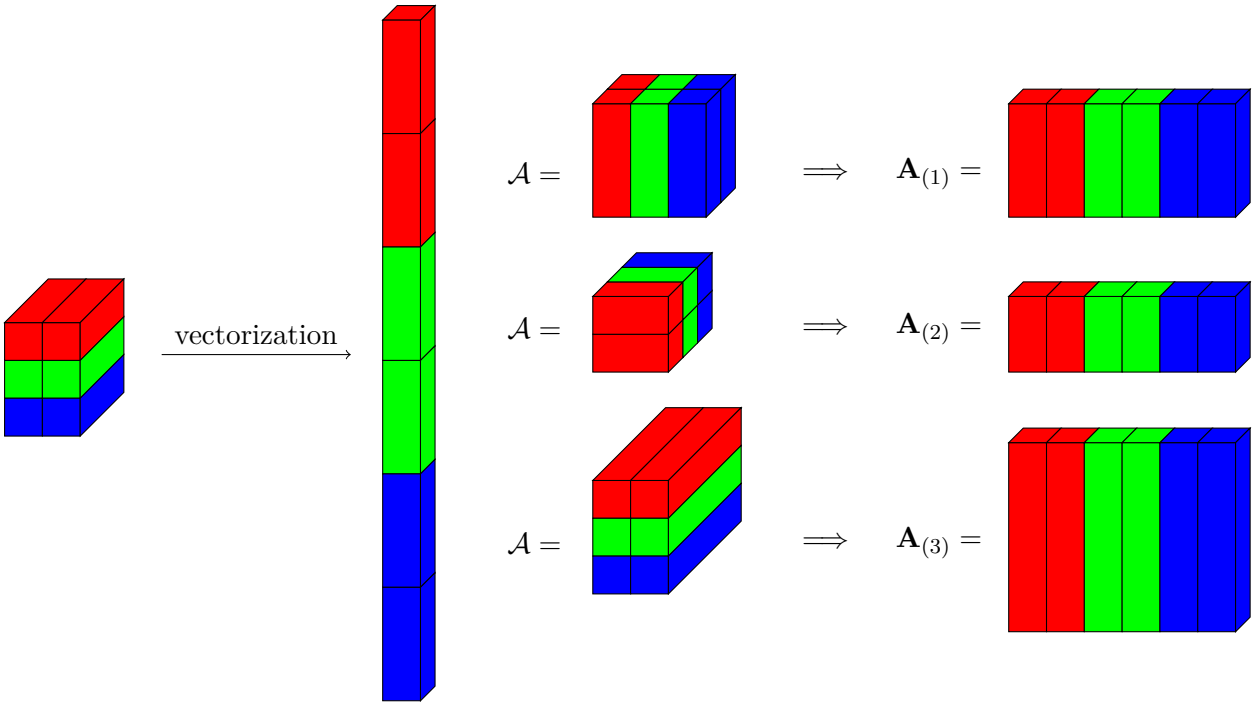


Figure 1.3: Schematic representations of vectorization (on the left) and matricizations (on the right).

In this section, we define some important algebraic operations for tensors defined over an arbitrary field F. In particular, we will see that $F^{I_1 \times \cdots \times I_N}$ can be endowed with a vector space structure in a very simple way. Another important topic is the problem of the definition of the rank of a tensor.

1.2.1 Algebraic operations on tensors

In this subsection, $N \in \mathbb{N}_*$ and $I_n, J_n \in \mathbb{N}_*$ for every $n \in \{1, \ldots, N\}$. For the sake of brevity, we do not always precise the values taken by the indices. Unless otherwise noted, an index i_n always varies from 1 to I_n.

Vector space The sum of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ with $\mathcal{B} \in F^{I_1 \times \cdots \times I_N}$ is defined elementwise:

$$(\mathcal{A} + \mathcal{B})(i_1, \ldots, i_N) := \mathcal{A}(i_1, \ldots, i_N) + \mathcal{B}(i_1, \ldots, i_N).$$

The product of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ by $a \in F$ is defined elementwise:

$$(a\mathcal{A})(i_1, \ldots, i_N) := a\mathcal{A}(i_1, \ldots, i_N).$$

Endowed with these operations, $F^{I_1 \times \cdots \times I_N}$ is a vector space of dimension $\prod_{n=1}^{N} I_n$.

Mode-n product The matrix product is a fundamental operation in linear algebra. Given two compatible matrices $\mathbf{A}$ and $\mathbf{B}$, the columns of $\mathbf{AB}$ are the columns of $\mathbf{B}$ multiplied (on the left) by $\mathbf{A}$; in the same way, the rows of $\mathbf{AB}$ are the rows of $\mathbf{A}$ multiplied (on the right) by $\mathbf{B}$. The operation that we introduce hereafter generalizes the matrix product to tensors of any order.

Let $n \in \{1, \ldots, N\}$. The *mode-n product* of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ by $\mathbf{B} \in F^{J_n \times I_n}$, denoted $\mathcal{A} \cdot_n \mathbf{B}$, is defined by

$$(\mathcal{A} \cdot_n \mathbf{B})(i_1, \ldots, i_{n-1}, j_n, i_{n+1}, \ldots, i_N) := \sum_{i_n=1}^{I_n} \mathcal{A}(i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N)\mathbf{B}(j_n, i_n).$$

In other words, the mode-n vectors of $\mathcal{A} \cdot_n \mathbf{B}$ are the mode-n vectors of $\mathcal{A}$ multiplied (on the left) by $\mathbf{B}$. That is,

$$(\mathcal{A} \cdot_n \mathbf{B})(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) = \mathbf{B}\mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N).$$

It is easy to verify that the mode-n product satisfies some kind of associativity property, as stated in the following result.

Proposition 1.2.1. *If $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ and $n \in \{1, \ldots, N\}$, then*

(a) for any $\mathbf{U}, \mathbf{V} \in F^{J_n \times I_n}$,

$$(\mathcal{A} \cdot_n \mathbf{U}) \cdot_n \mathbf{V} = \mathcal{A} \cdot_n (\mathbf{V}\mathbf{U}),$$

(b) for any $\mathbf{U} \in F^{J_n \times I_n}$ and $\mathbf{V} \in F^{J_m \times I_m}$ with $m \in \{1, \ldots, N\} \setminus \{n\}$,

$$(\mathcal{A} \cdot_n \mathbf{U}) \cdot_m \mathbf{V} = (\mathcal{A} \cdot_m \mathbf{V}) \cdot_n \mathbf{U}.$$

It is interesting to note that for any $\mathbf{A} \in F^{I \times J}$, $\mathbf{U} \in F^{M \times I}$, and $\mathbf{V} \in F^{N \times J}$,

$$\mathbf{A} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} = \mathbf{U}\mathbf{A}\mathbf{V}^{\mathrm{T}}.$$

Example Let us go back to the first example of section 1.1. Let $n > 1$ be an integer and $f : \mathbb{R}^n \to \mathbb{R}$ be an infinitely differentiable function. The Taylor polynomial of order $N \in \mathbb{N}_*$ of f at $\mathbf{a} \in \mathbb{R}^n$, denoted $T^N_{f,\mathbf{a}}$, can be described in terms of the partial derivatives of f but it is a bit cumbersome. For instance,

$$T^3_{f,\mathbf{a}}(\mathbf{x}) = f(\mathbf{a}) + \sum_{i=1}^{n} \partial_i f(\mathbf{a})(\mathbf{x}(i) - \mathbf{a}(i)) + \sum_{i=1}^{n}\sum_{j=1}^{n} \partial_{i,j} f(\mathbf{a})(\mathbf{x}(i) - \mathbf{a}(i))(\mathbf{x}(j) - \mathbf{a}(j))$$
$$+ \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \partial_{i,j,k} f(\mathbf{a})(\mathbf{x}(i) - \mathbf{a}(i))(\mathbf{x}(j) - \mathbf{a}(j))(\mathbf{x}(k) - \mathbf{a}(k)).$$

Using the mode-n product, we can simply write

$$T^3_{f,a}(\mathbf{x}) = f(\mathbf{a}) + f'(\mathbf{a}) \cdot_1 (\mathbf{x}-\mathbf{a})^{\mathrm{T}} + f''(\mathbf{a}) \cdot_1 (\mathbf{x}-\mathbf{a})^{\mathrm{T}} \cdot_2 (\mathbf{x}-\mathbf{a})^{\mathrm{T}} + f'''(\mathbf{a}) \cdot_1 (\mathbf{x}-\mathbf{a})^{\mathrm{T}} \cdot_2 (\mathbf{x}-\mathbf{a})^{\mathrm{T}} \cdot_3 (\mathbf{x}-\mathbf{a})^{\mathrm{T}},$$

where $f^{(N)}$ stands for the Nth-order tensor containing the Nth-order partial derivatives of f.

Vectorization and matricizations with mode-n products Thanks to the Kronecker product (see section A.3 for a short review), vectorization and matricizations interact nicely with the mode-n products, as shown by the two following results whose proofs are simple manipulations of indices.

Proposition 1.2.2. *If $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ and $\mathbf{U}_n \in F^{J_n \times I_n}$ for each $n \in \{1, \ldots, N\}$, then*

$$\operatorname{vec}(\mathcal{A} \cdot_1 \mathbf{U}_1 \cdots \cdot_N \mathbf{U}_N) = (\mathbf{U}_1 \otimes \cdots \otimes \mathbf{U}_N) \operatorname{vec}(\mathcal{A}).$$

Proposition 1.2.3. *Let $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ and $\mathbf{U}_n \in F^{J_n \times I_n}$ for each $n \in \{1, \ldots, n\}$. If $\mathcal{B} := \mathcal{A} \cdot_1 \mathbf{U}_1 \cdots \cdot_N \mathbf{U}_N$, then*

$$\mathbf{B}_{(n)} = \mathbf{U}_n \otimes \mathbf{A}_{(n)} \left(\left(\bigotimes_{k=n+1}^{N} \mathbf{U}_k \right) \otimes \left(\bigotimes_{l=1}^{n-1} \mathbf{U}_l \right) \right)^{\mathrm{T}}$$

For third-order tensors, we have the following formulas:

$$\mathbf{B}_{(1)} = \mathbf{U}\mathbf{A}_{(1)}(\mathbf{V} \otimes \mathbf{W})^{\mathrm{T}},$$
$$\mathbf{B}_{(2)} = \mathbf{V}\mathbf{A}_{(2)}(\mathbf{W} \otimes \mathbf{U})^{\mathrm{T}},$$
$$\mathbf{B}_{(3)} = \mathbf{W}\mathbf{A}_{(3)}(\mathbf{U} \otimes \mathbf{V})^{\mathrm{T}}.$$

1.2.2 Multilinear rank

The rank of a matrix defined over a field can be thought as a measure of the "quantity of information" contained in that matrix at the light of the algebraic operations one can perform with matrices. The idea is that a matrix in which some columns can be recovered by performing linear combinations of other columns does not contain as much information as its size could suggest. For instance, the rank of the matrix associated to a linear system is the true size of the system. In this subsection, we try to extend the notion of rank to higher-order tensors.

Matrix rank Let us recall that the column space of a matrix $\mathbf{A} \in F^{I \times J}$, denoted $\text{range}(\mathbf{A})$, and its row space have the same dimension which is called the rank of $\mathbf{A}$ and denoted by $\text{rank}(\mathbf{A})$. The rank of $\mathbf{A}$ is also (see, e.g., [11, sections 0.3 & 0.4])

(a) the number of nonzero rows in the reduced row echelon form (RREF) of $\mathbf{A}$,

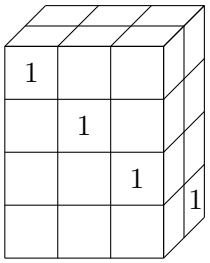
(b) the unique invariant of $\mathbf{A}$ under invertible transformations,

(c) the maximum $R \in \mathbb{N}$ such that some $R \times R$ submatrix of $\mathbf{A}$ has nonzero determinant,

(d) the minimum $R \in \mathbb{N}$ such that $\mathbf{A} = \sum_{r=1}^{R} \mathbf{a}_r \mathbf{b}_r^{\mathrm{T}}$ for some $\mathbf{a}_r \in F^I$ and $\mathbf{b}_r \in F^J$, $r \in \{1, \ldots, R\}$.

To define what is the rank of a higher-order tensor, it seems natural to try to generalize one of the above properties of the matrix rank. However, it is not clear whether properties (a), (b) and (c) can be generalized to higher-order tensors in a meaningful way. Therefore, we shall proceed as follows. First, we give a definition of the rank based on the generalization of rows and columns of matrices. Then, in the next subsection, we study the link between this definition and the extension of property (d).

Mode-n rank Let us try to define the tensor rank using the generalization of row and column spaces. The vector space spanned by the mode-n vectors of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ is denoted by $\text{range}_n(\mathcal{A})$ for each $n \in \{1, \ldots, N\}$. The dimension of $\text{range}_n(\mathcal{A})$, denoted $\text{rank}_n(\mathcal{A})$, is called the *mode-n rank* of $\mathcal{A}$. Let us consider a small example. Let $\mathcal{A} \in \mathbb{R}^{4 \times 3 \times 2}$ be defined by

$\mathcal{A} =$ 

where each empty (or invisible) box contains a zero. It is easy to see that $\text{rank}_1(\mathcal{A}) = 4$, $\text{rank}_2(\mathcal{A}) = 3$, and $\text{rank}_3(\mathcal{A}) = 2$. We conclude that, contrarily to the matrix case, the mode-n ranks of a higher-order tensor may not be equal. This leads to the following definition. The *multilinear rank* of a Nth-order tensor is the N-tuple of the mode-n ranks. If the multilinear rank of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ is $(R_1, \ldots, R_N)$, we say that $\mathcal{A}$ is a rank-$(R_1, \ldots, R_N)$ tensor. For instance, the tensor $\mathcal{A}$ defined above is a rank-$(4, 3, 2)$ tensor.

Mode-n rank and matricization Let $\mathcal{A}$ be a Nth-order tensor defined over a field. From the definitions, it is clear that $\text{range}_n(\mathcal{A}) = \text{range}(\mathbf{A}_{(n)})$ for every $n \in \{1, \ldots, N\}$; in particular, $\text{rank}_n(\mathcal{A}) = \text{rank}(\mathbf{A}_{(n)})$. This simple observation together with Proposition 1.2.3 will be particularly useful when it comes to studying the effect of mode-n products on the multilinear rank, as shown in the next paragraph.

Multilinear rank and mode-n product Let us study how the mode-n products affect the multilinear rank of a tensor. Before that, let us just recall that if $\mathbf{A}$ and $\mathbf{B}$ are compatible matrices over a field, then

$$\operatorname{range}(\mathbf{AB}) \subset \operatorname{range}(\mathbf{A})$$

and

$$\operatorname{rank}(\mathbf{AB}) \leq \min(\operatorname{rank}(\mathbf{A}), \operatorname{rank}(\mathbf{B})).$$

Moreover, if $\mathbf{B}$ has full column rank, then

$$\operatorname{range}(\mathbf{AB}) = \operatorname{range}(\mathbf{A})$$

since we can find $\mathbf{C}$ such that $\mathbf{CB} = \mathbf{I}$.

Proposition 1.2.4. *If $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$, $n \in \{1, \ldots, N\}$, $\mathbf{M} \in F^{J_n \times I_n}$ and $\mathcal{B} := \mathcal{A} \cdot_n \mathbf{M}$, then*

(a) $\operatorname{range}_m(\mathcal{B}) \subset \operatorname{range}_m(\mathcal{A})$ *for every* $m \in \{1, \ldots, N\} \setminus \{n\}$,

(b) $\operatorname{range}_n(\mathcal{B}) \subset \operatorname{range}(\mathbf{M})$,

(c) $\operatorname{rank}_n(\mathcal{B}) \leq \operatorname{rank}_n(\mathcal{A})$.

In particular, $\operatorname{rank}_m(\mathcal{B}) \leq \operatorname{rank}_m(\mathcal{A})$ *and* $\operatorname{rank}_n(\mathcal{B}) \leq \min(\operatorname{rank}_n(\mathcal{A}), \operatorname{rank}(\mathbf{M}))$.
Moreover, if $\mathbf{M}$ *has full column rank (which implies* $J_n \leq I_n$*), equality holds in both (a) and (b).*

Proof. For point (a), observe that the mode-m vectors of $\mathcal{B}$ are linear combinations of the mode-m vectors of $\mathcal{A}$. Points (b) and (c) follow from the equation $\mathbf{B}_{(n)} = \mathbf{M}\mathbf{A}_{(n)}$.

Let us now assume that $\mathbf{M}$ has full column rank. We can find $\mathbf{P} \in F^{J_n \times I_n}$ that is full row rank and such that $\mathbf{PM} = \mathbf{I}_{J_n}$. Taking the mode-$n$ product with $\mathbf{P}$ on both sides of the equality $\mathcal{A} = \mathcal{B} \cdot_n \mathbf{M}$ yields $\mathcal{B} = \mathcal{A} \cdot_n \mathbf{P}$ thanks to Proposition 1.2.1. The result follows. □

The preceding result admits a kind of converse.

Proposition 1.2.5. *Let $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$, $n \in \{1, \ldots, N\}$ and $R_n \leq I_n$. The following statements are equivalent:*

(a) $\operatorname{rank}_n(\mathcal{A}) \leq R_n$,

(b) $\mathcal{A} = \mathcal{B} \cdot_n \mathbf{M}$ *for some* $\mathcal{B} \in F^{I_1 \times \cdots \times I_{n-1} \times R_n \times I_{n+1} \times \cdots \times I_N}$ *and* $\mathbf{M} \in F^{I_n \times R_n}$.

Proof. In view of Proposition 1.2.4, it is clear that (b) implies (a). Conversely, assume that (a) holds. We can find $\mathbf{M} \in F^{I_n \times R_n}$ such that $\operatorname{range}_n(\mathcal{A}) \subset \operatorname{range}(\mathbf{M})$. This means there is $\mathcal{B} \in F^{I_1 \times \cdots \times I_{n-1} \times R_n \times I_{n+1} \times \cdots \times I_N}$ such that $\mathcal{A} = \mathcal{B} \cdot_n \mathbf{M}$. □

1.2.3 Tensor rank*

Let us now go back to the property (d) of the matrix rank. As the mode-n ranks are not necessarily equal, we can already expect that there will be no more equivalence between the multilinear rank and what could be a meaningful generalization of property (d).

Tensor product Let us start by generalizing the product $\mathbf{ab}^{\mathrm{T}}$ that appears in property (d) to more than two vectors. The *tensor product* (or *outer product*) of $\mathcal{A} \in F^{I_1 \times \cdots \times I_M}$ and $\mathcal{B} \in F^{J_1 \times \cdots \times J_N}$, denoted $\mathcal{A} \circ \mathcal{B}$, is defined by

$$(\mathcal{A} \circ \mathcal{B})(i_1, \ldots, i_M, j_1, \ldots, j_N = 1) := \mathcal{A}(i_1, \ldots, i_M)\mathcal{B}(j_1, \ldots, j_N)$$

for each $(i_1, \ldots, i_M) \in \prod_{m=1}^{M} \{1, \ldots, I_m\}$ and $(j_1, \ldots, j_N) \in \prod_{n=1}^{N} \{1, \ldots, J_n\}$.

Rank-one tensor A tensor $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ is a *rank-one* tensor if and only if there are N nonzero vectors $\mathbf{a}_1, \ldots, \mathbf{a}_N$ with $\mathbf{a}_n \in F^{I_n}$ for each $n \in \{1, \ldots, N\}$ such that

$$\mathcal{A} = \mathbf{a}_1 \circ \cdots \circ \mathbf{a}_N.$$

A schematic representation of a rank-one tensor of order 3 is given in Figure 1.4.

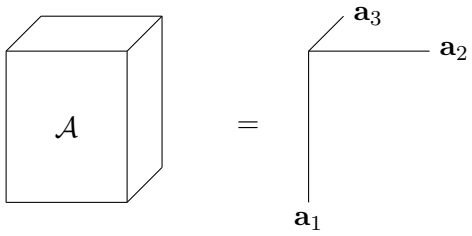


Figure 1.4: Schematic representation of a rank-one tensor of order 3.

Tensor rank The *rank* of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$, denoted $\operatorname{rank}(\mathcal{A})$, is the minimum number of rank-one tensors needed to produce $\mathcal{A}$ as their sum, i.e., the minimum $R \in \mathbb{N}$ such that there exist rank-one tensors $\mathcal{A}_1, \ldots, \mathcal{A}_R \in F^{I_1 \times \cdots \times I_N}$ such that

$$\mathcal{A} = \sum_{r=1}^{R} \mathcal{A}_r.$$

More explicitly, the rank of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ is the minimum $R \in \mathbb{N}$ such that there exist $\mathbf{a}_r^n \in F^{I_n}$ for each $n \in \{1, \ldots, N\}$ and $r \in \{1, \ldots, R\}$ such that

$$\mathcal{A} = \sum_{r=1}^{R} \mathbf{a}_r^1 \circ \cdots \circ \mathbf{a}_r^N.$$

If the rank of $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ is R, we say that $\mathcal{A}$ is a rank-R tensor.

Computing tensor rank We saw that computing the multilinear rank of a tensor reduces to computing the rank of a matricization of that tensor which is an easy task. Therefore, it may seem surprising that determining the rank of a particular real higher-order tensor is NP-hard [12].

Rank and mode-n product Let us study how mode-n products affect the rank of a tensor.

Proposition 1.2.6. *If $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$ and $\mathbf{B} \in F^{J_n \times I_n}$ for some $n \in \{1, \ldots, N\}$, then*

$$\operatorname{rank}(\mathcal{A} \cdot_n \mathbf{B}) \leq \operatorname{rank}(\mathcal{A}).$$

Proof. Let $R := \operatorname{rank}(\mathcal{A})$. There are $\mathbf{a}_r^n \in F^{I_n}$ for each $n \in \{1, \ldots, N\}$ and $r \in \{1, \ldots, R\}$ such that

$$\mathcal{A} = \sum_{r=1}^{R} \mathbf{a}_r^1 \circ \cdots \circ \mathbf{a}_r^N.$$

Now, observe that

$$\mathcal{A} \cdot_n \mathbf{B} = \sum_{r=1}^{R} \mathbf{a}_r^1 \circ \cdots \circ (\mathbf{B}\mathbf{a}_r^n) \circ \cdots \circ \mathbf{a}_r^N$$

which shows that $\operatorname{rank}(\mathcal{A} \cdot_n \mathbf{B}) \leq R$. □

Bounds on tensor rank In general, the best thing that can be done is bounding the rank of a tensor.

Proposition 1.2.7. *For any* $\mathcal{A} \in F^{I_1 \times \cdots \times I_N}$,

$$\max_{n \in \{1,\ldots,N\}} \operatorname{rank}_n(\mathcal{A}) \leq \operatorname{rank}(\mathcal{A}) \leq \min_{m \in \{1,\ldots,N\}} \prod_{\substack{n=1 \\ n \neq m}}^{N} \operatorname{rank}_n(\mathcal{A})$$

Proof. Let $R := \operatorname{rank}(\mathcal{A})$. There exist $\mathbf{a}_r^n \in F^{I_n}$ for each $n \in \{1, \ldots, N\}$ and $r \in \{1, \ldots, R\}$ such that

$$\mathcal{A} = \sum_{r=1}^{R} \mathbf{a}_r^1 \circ \cdots \circ \mathbf{a}_r^N.$$

In particular, each mode-n vector of $\mathcal{A}$ is a linear combination of $\mathbf{a}_1^n, \ldots, \mathbf{a}_R^n$. The first inequality follows. The proof of the second inequality is a little more tricky. Here we restrict ourselves to the case $N := 3$.

We prove that $\operatorname{rank}(\mathcal{A}) \leq R_3 \min(R_1, R_2)$ where $R_n := \operatorname{rank}_n(\mathcal{A})$ for each $n \in \{1, 2, 3\}$. The inequality $\operatorname{rank}(\mathcal{A}) \leq R_1 R_2$ can be proved with a similar argument. In view of the definition of R_3, we can find $\mathbf{U} \in F^{I_3 \times R_3}$ that has full column rank and $\mathcal{B} \in F^{I_1 \times I_2 \times R_3}$ such that $\mathcal{A} = \mathcal{B} \cdot_3 \mathbf{U}$. By Proposition 1.2.6, $\operatorname{rank}(\mathcal{A}) \leq \operatorname{rank}(\mathcal{B})$. For each $k \in \{1, \ldots, R_3\}$, let $r_k := \operatorname{rank}(\mathcal{B}(:, :, k))$, with $\mathcal{B}(:, :, k) := [\mathcal{B}(i, j, k)]_{i,j=1}^{I_1, I_2}$, so that there exist $\mathbf{a}_1^k, \ldots, \mathbf{a}_{r_k}^k \in F^{I_1}$ and $\mathbf{b}_1^k, \ldots, \mathbf{b}_{r_k}^k \in F^{I_2}$ such that

$$\mathcal{B}(:, :, k) = \sum_{r=1}^{r_k} \mathbf{a}_r^k (\mathbf{b}_r^k)^{\mathrm{T}} = \sum_{r=1}^{r_k} \mathbf{a}_r^k \circ \mathbf{b}_r^k.$$

Defining $\mathbf{c}_k \in F^{R_3}$ by $\mathbf{c}_k(i) := \delta_{i,k}$ for every $i \in \{1, \ldots, R_3\}$, we have

$$\mathcal{B} = \sum_{k=1}^{R_3} \mathcal{B}(:, :, k) \circ \mathbf{c}_k.$$

Consequently,

$$\mathcal{B} = \sum_{k=1}^{R_3} \left(\sum_{r=1}^{r_k} \mathbf{a}_r^k \circ \mathbf{b}_r^k \right) \circ \mathbf{c}_k = \sum_{k=1}^{R_3} \sum_{r=1}^{r_k} \mathbf{a}_r^k \circ \mathbf{b}_r^k \circ \mathbf{c}_k.$$

Is is clear that $r_k \leq \min(\operatorname{rank}_1(\mathcal{B}), \operatorname{rank}_2(\mathcal{B}))$. By Proposition 1.2.4, $\operatorname{rank}_n(\mathcal{B}) = R_n$ for each $n \in \{1, 2\}$. Therefore, the latter sum contains at most $R_3 \min(R_1, R_2)$ terms, which completes the proof. □

1.3 Inner product for tensors

We saw in the previous section that the set of tensors defined over a field can be endowed with a vector space structure. In this section, we define an inner product in $\mathbb{R}^{I_1 \times \cdots \times I_N}$. This will allow us to talk about orthogonality and norm.

The function

$$\langle \cdot, \cdot \rangle : \mathbb{R}^{I_1 \times \cdots \times I_N} \times \mathbb{R}^{I_1 \times \cdots \times I_N} \to \mathbb{R} : (\mathcal{A}, \mathcal{B}) \mapsto \langle \mathcal{A}, \mathcal{B} \rangle := \sum_{\mathbf{i} \in \prod_{n=1}^{N} \{1,\ldots,I_n\}} \mathcal{A}(\mathbf{i}) \mathcal{B}(\mathbf{i}).$$

is an inner product on $\mathbb{R}^{I_1 \times \cdots \times I_N}$, called the standard inner product in $\mathbb{R}^{I_1 \times \cdots \times I_N}$. Let us recall that for $N = 2$, we have for every $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I_1 \times I_2}$ that

$$\langle \mathbf{A}, \mathbf{B} \rangle = \operatorname{tr}(\mathbf{B}^{\mathrm{T}} \mathbf{A}),$$

where $\mathrm{tr}(\mathbf{M})$ denotes the trace of the square matrix $\mathbf{M}$ (see section A.2).

We let $\|\cdot\|$ denote the norm induced by the standard inner product, i.e., the Frobenius norm:

$$\|\cdot\| : \mathbb{R}^{I_1 \times \cdots \times I_N} \to \mathbb{R} : \mathcal{A} \mapsto \|\mathcal{A}\| := \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}.$$

The following result allows us to compute the inner product between two tensors by computing the inner product between their matricizations. The proof is a simple manipulation of indices.

Proposition 1.3.1. *For any tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $n \in \{1, \ldots, N\}$,*

$$\langle \mathcal{A}, \mathcal{B} \rangle = \langle \mathbf{A}_{(n)}, \mathbf{B}_{(n)} \rangle = \mathrm{tr}(\mathbf{B}_{(n)}^{\mathrm{T}} \mathbf{A}_{(n)}).$$

In particular, $\|\mathcal{A}\| = \|\mathbf{A}_{(n)}\|$.

1.4 Tensor decompositions

The singular value decomposition (SVD) is one the most important matrix decompositions. Therefore, it is not surprising that trying to extend the notion of SVD from matrices to higher-order tensors was one of the first problems of interest in numerical multilinear algebra. However, at this stage, we can already expect that it will be impossible to find a higher-order equivalent of the matrix SVD that retains all its properties for the simple reason that even the matrix rank cannot be extended in a unique way.

In this section, we introduce three tensor decompositions. The two first ones, the Tucker decomposition (TKD) and the canonical polyadic decomposition (CPD), can be thought as higher-order generalizations of the matrix SVD, respectively related to the notions of multilinear rank and tensor rank. The third decomposition is the block term decomposition (BTD) and is at the heart of the present work. It can be seen as a unifying generalization of the two first ones.

We consider only real tensors in order to be able to talk about orthogonality. Moreover, we restrict ourselves to third-order tensors for simplicity; extending the concepts to general higher-order tensors is straightforward.

1.4.1 Tucker decomposition

We start by introducing a decomposition that reveals the multilinear rank of a tensor. A *Tucker decomposition* of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is a decomposition of the form

$$\mathcal{A} = \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W},$$

where $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, with $R_n \leq I_n$ for each $n \in \{1, 2, 3\}$, is often called the *core tensor*, and $\mathbf{U} \in \mathrm{St}(R_1, I_1)$, $\mathbf{V} \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W} \in \mathrm{St}(R_3, I_3)$ are often referred to as the *factor matrices*.[2] Given positive integers $R \leq I$, we let $\mathrm{St}(R, I) := \{\mathbf{U} \in \mathbb{R}^{I \times R} : \mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}_R\}$ denote the *Stiefel manifold*[3], i.e., the set of all columnwise orthonormal $I \times R$ matrices. A schematic representation of the Tucker decomposition is given in Figure 1.5.

The TKD can be used, among others, for dimensionality reduction and for signal subspace estimation [2]. In this subsection, we discuss some of its properties in more detail.

[2] It seems there is no general agreement on the precise definition of the Tucker decomposition. For instance, the Tucker decomposition defined in this text is referred to as the compact Tucker model in [3]. In [2], the factor matrices do not necessarily have orthonormal columns. In [9], "Tucker decomposition" is a synonym of "HOSVD".

[3] This notion will be explained later.

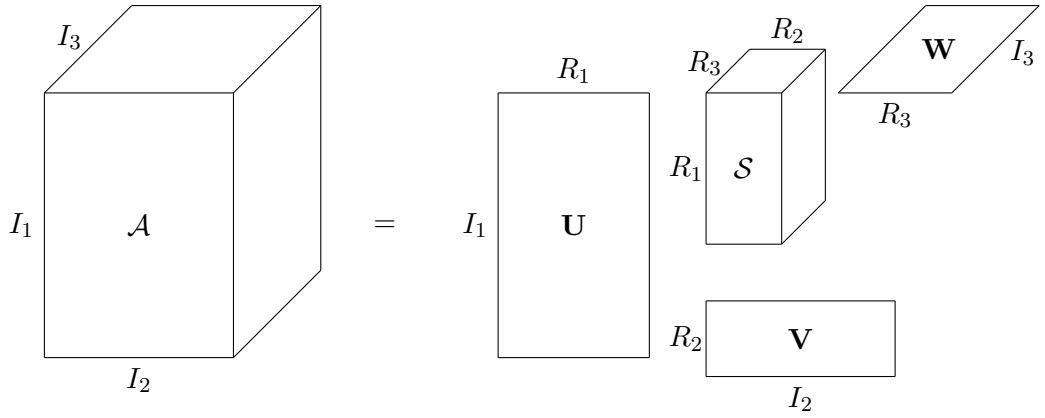


Figure 1.5: Schematic representation of a Tucker decomposition. By convention, we write the number of rows (I_n) and columns (R_n) of each of the three factor matrices respectively on the left and on the top of them. The diagram should be read as follows. Recall that a mode-n product consists of the multiplication of each mode-n vector with the corresponding factor matrix. On the diagram, when looking each matrix in frontal view (i.e., with the top and the left of each matrix positioned in agreement with the convention), one can see that each matrix is positioned so that it can multiply the corresponding mode-n vectors of $\mathcal{S}$ as in a usual schema of a matrix-vector product. Thus, to read the mode-1 product, one just has to look the schema in frontal view. To read the mode-2 product, one should look after a quarter turn counterclockwise. Finally, to read the mode-3 product, one should look the schema in top view and so that the "R_3" is at the top of $\mathbf{W}$.

Let us investigate the link between the multilinear rank and the Tucker decomposition.

Proposition 1.4.1. *Let $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $R_n \in \{1, \dots, I_n\}$ for each $n \in \{1, 2, 3\}$. The following statements are equivalent:*

(a) $\operatorname{rank}_n(\mathcal{A}) \leq R_n$ *for each* $n \in \{1, 2, 3\}$,

(b) there exist $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U} \in \operatorname{St}(R_1, I_1)$, $\mathbf{V} \in \operatorname{St}(R_2, I_2)$ *and* $\mathbf{W} \in \operatorname{St}(R_3, I_3)$ *such that*

$$\mathcal{A} = \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$$

with $\operatorname{rank}_n(\mathcal{S}) = \operatorname{rank}_n(\mathcal{A})$ *for each* $n \in \{1, 2, 3\}$.

Moreover, if equalities hold in (a), then $\operatorname{range}_1(\mathcal{A}) = \operatorname{range}(\mathbf{U})$, $\operatorname{range}_2(\mathcal{A}) = \operatorname{range}(\mathbf{V})$ *and* $\operatorname{range}_3(\mathcal{A}) = \operatorname{range}(\mathbf{W})$.

Proof. In view of Proposition 1.2.4, it is clear that (b) implies (a). Conversely, assume that (a) holds. We can find $\mathbf{U} \in \operatorname{St}(R_1, I_1)$ such that $\operatorname{range}_1(\mathcal{A}) \subset \operatorname{range}(\mathbf{U})$. This means there is $\mathcal{S}_1 \in \mathbb{R}^{R_1 \times I_2 \times I_3}$ such that $\mathcal{A} = \mathcal{S}_1 \cdot_1 \mathbf{U}$. Actually, this $\mathcal{S}_1$ is unique and given by $\mathcal{S}_1 = \mathcal{A} \cdot_1 \mathbf{U}^{\mathrm{T}}$. Therefore, by Proposition 1.2.4, $\operatorname{rank}_1(\mathcal{S}_1) = \operatorname{rank}_1(\mathcal{A})$ and $\operatorname{range}_n(\mathcal{S}_1) = \operatorname{range}_n(\mathcal{A})$ for each $n \in \{2, 3\}$. Thus, we can find $\mathbf{V} \in \operatorname{St}(R_2, I_2)$ such that $\operatorname{range}_2(\mathcal{S}_1) = \operatorname{range}_2(\mathcal{A}) \subset \operatorname{range}(\mathbf{V})$. This means there is $\mathcal{S}_2 \in \mathbb{R}^{R_1 \times R_2 \times I_3}$ such that $\mathcal{S}_1 = \mathcal{S}_2 \cdot_2 \mathbf{V}$. Again, this $\mathcal{S}_2$ in unique and given by $\mathcal{S}_2 = \mathcal{S}_1 \cdot_2 \mathbf{V}^{\mathrm{T}} = \mathcal{A} \cdot_1 \mathbf{U}^{\mathrm{T}} \cdot_2 \mathbf{V}^{\mathrm{T}}$. By Proposition 1.2.4, $\operatorname{rank}_2(\mathcal{S}_2) = \operatorname{rank}_2(\mathcal{A})$ and $\operatorname{range}_n(\mathcal{S}_2) = \operatorname{range}_n(\mathcal{S}_1)$ for each $n \in \{1, 3\}$. Therefore, there is $\mathbf{W} \in \operatorname{St}(R_3, I_3)$ such that $\operatorname{range}_3(\mathcal{S}_2) = \operatorname{range}_3(\mathcal{A}) \subset \operatorname{range}(\mathbf{W})$. Thus, there is $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ such that $\mathcal{S}_2 = \mathcal{S} \cdot_3 \mathbf{W}$. This $\mathcal{S}$ is unique and given by $\mathcal{S} = \mathcal{S}_2 \cdot_3 \mathbf{W}^{\mathrm{T}}$. By Proposition 1.2.4, $\operatorname{rank}_3(\mathcal{S}) = \operatorname{rank}_3(\mathcal{A})$ and $\operatorname{range}_n(\mathcal{S}) = \operatorname{range}_n(\mathcal{S}_2)$ for each $n \in \{1, 2\}$. In conclusion, (b) holds and $\mathcal{S}$ can be computed as $\mathcal{S} = \mathcal{A} \cdot_1 \mathbf{U}^{\mathrm{T}} \cdot_2 \mathbf{V}^{\mathrm{T}} \cdot_3 \mathbf{W}^{\mathrm{T}}$. Finally, if equalities hold in (a), it is clear that $\operatorname{range}_1(\mathcal{A}) = \operatorname{range}(\mathbf{U})$, $\operatorname{range}_2(\mathcal{A}) = \operatorname{range}(\mathbf{V})$ and $\operatorname{range}_3(\mathcal{A}) = \operatorname{range}(\mathbf{W})$. □

The preceding result shows that the multilinear rank of a tensor $\mathcal{A}$ is the size of the smallest core tensor that can be used to write a Tucker decomposition of $\mathcal{A}$. The important information contained in a factor matrix is its range, not the particular values of its entries. Indeed, from a given Tucker decomposition

$$\mathcal{A} = \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W},$$

we can write many others by performing orthogonal transformations on the columns of $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$. More precisely, if $\mathbf{Q}_i \in O_{R_i}$ for each $i \in \{1, 2, 3\}$, we can write

$$\mathcal{A} = (\underbrace{\mathcal{S} \cdot_1 \mathbf{Q}_1^{\mathrm{T}} \cdot_2 \mathbf{Q}_2^{\mathrm{T}} \cdot_3 \mathbf{Q}_3^{\mathrm{T}}}_{\tilde{\mathcal{S}}}) \cdot_1 (\underbrace{\mathbf{U}\mathbf{Q}_1}_{\tilde{\mathbf{U}}}) \cdot_2 (\underbrace{\mathbf{V}\mathbf{Q}_2}_{\tilde{\mathbf{V}}}) \cdot_3 (\underbrace{\mathbf{W}\mathbf{Q}_3}_{\tilde{\mathbf{W}}}),$$

where $\tilde{\mathcal{S}}$ and $\mathcal{S}$ have the same multilinear rank and $\mathrm{range}(\tilde{\mathbf{U}}) = \mathrm{range}(\mathbf{U})$, $\mathrm{range}(\tilde{\mathbf{V}}) = \mathrm{range}(\mathbf{V})$, $\mathrm{range}(\tilde{\mathbf{W}}) = \mathrm{range}(\mathbf{W})$. Using such orthogonal transformations, it is possible to get orthogonality properties for the core tensor. The decomposition is then called the higher-order singular value decomposition (HOSVD) or the multilinear singular value decomposition (MLSVD). We refer to [3, section V] for the details.

The next result shows that mode-n products with columnwise orthonormal matrices do not affect the norm.

Proposition 1.4.2. *If $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ can be written as*

$$\mathcal{A} = \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}.$$

for some $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U} \in \mathrm{St}(R_1, I_1)$, $\mathbf{V} \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W} \in \mathrm{St}(R_3, I_3)$, then

$$\|\mathcal{A}\| = \|\mathcal{S}\|.$$

Proof. Using Proposition 1.3.1 and Proposition 1.2.3, we have

$$\|\mathcal{A}\| = \|\mathbf{A}_{(1)}\| = \left\|\mathbf{U}\mathbf{S}_{(1)}(\mathbf{V} \otimes \mathbf{W})^{\mathrm{T}}\right\|.$$

We compute

$$\begin{aligned}
\left\|\mathbf{U}\mathbf{S}_{(1)}(\mathbf{V} \otimes \mathbf{W})^{\mathrm{T}}\right\|^2 &= \mathrm{tr}\left((\mathbf{V} \otimes \mathbf{W})\mathbf{S}_{(1)}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{S}_{(1)}(\mathbf{V} \otimes \mathbf{W})^{\mathrm{T}}\right) \\
&= \mathrm{tr}\left((\mathbf{V} \otimes \mathbf{W})^{\mathrm{T}}(\mathbf{V} \otimes \mathbf{W})\mathbf{S}_{(1)}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{S}_{(1)}\right) \\
&= \mathrm{tr}(\mathbf{S}_{(1)}^{\mathrm{T}}\mathbf{S}_{(1)}) \\
&= \|\mathbf{S}_{(1)}\|^2.
\end{aligned}$$

We conclude using again Proposition 1.3.1. □

Tensor approximation by TKD The problem of computing the best low multilinear rank approximation of a third-order tensor in the least-squares sense was discussed in [10, section 4]. It was shown that this problem can be made easier to solve using variable projection. This was used in the paper [7] to develop an algorithm based on the Riemannian trust-region scheme.

The low multilinear rank approximation problem is much more complex for third-order tensors than for matrices. Indeed, the best low rank approximation of a matrix is obtained by truncating its SVD, as stated by the Eckart–Young theorem (see, e.g., [1, Theorem 2.4.8]). However, the truncated HOSVD of a higher-order tensor is not necessarily its best low multilinear rank approximation (see, e.g., [3, section V]). Nevertheless, the truncated HOSVD can be used as a sarting point for the numerical methods. Furthermore, the paper [13] showed that the cost function associated to the low multilinear rank approximation problem may have several different local minima.

1.4.2 Canonical polyadic decomposition*

This subsection is a quick disgression to introduce the canonical polyadic decomposition, an important tensor decomposition that is associated with the tensor rank, as defined in subsection 1.2.3. The aim of this subsection is to contextualize the origin of the BTD.

A *polyadic decomposition* of a tensor is a decomposition of a tensor as a sum of rank-one tensors. If the number of terms in a polyadic decomposition is minimum, it is called a *canonical polyadic decomposition*.

Uniqueness of CPD A big difference between CPD of matrices and CPD of higher-order tensors is that the second is essentially unique under mild conditions while the former is never essentially unique unless stronger conditions such as orthogonality or triangularity are imposed.[4] The precise statement of this result will not be given here but can be found in the overview papers [2, 3] and the references therein.

Applications of CPD Many applications of the CPD are presented in [2]: *The CPD has already been established as an advanced tool for signal separation in vastly diverse branches of signal processing and data analysis, such as in audio and speech processing, biomedical engineering, chemometrics and machine learning. The CPD is also heavily used in exploratory data analysis, where the rank-one terms capture the essential properties of dynamically complex signals.*

1.4.3 Block term decomposition

The block term decomposition was introduced in the three-part paper [4, 5, 6]. A BTD is a decomposition of a given tensor in a sum of Tucker terms. More precisely, a *block term decomposition* of $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is a decomposition of the form

$$\mathcal{A} = \sum_{r=1}^{R} \mathcal{S}_r \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r$$

where $R \in \mathbb{N}_*$ and, for each $r \in \{1, \ldots, R\}$, $\mathcal{S}_r \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U}_r \in \mathrm{St}(R_1, I_1)$, $\mathbf{V}_r \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W}_r \in \mathrm{St}(R_3, I_3)$ with $R_n \leq I_n$ for each $n \in \{1, 2, 3\}$.

From atoms to molecules in data analysis [5] The BTD unifies the two most well-known tensor decompositions, namely the TKD and the CPD. The BTD framework also gives a unified view on how the basic concept of rank can be generalized from matrices to tensors. While in CPD, as well as in classical matrix decompositions, the components are rank-one terms, the terms in a BTD have "low" (multilinear) rank. It turns out that BTDs are still unique under mild conditions [5, section 5]. Conventional signal separation and data analysis (relying on matrix decompositions or CPD) involve rank-one components. Such techniques decompose data in "atoms" (most elementary parts). In contrast, BTDs decompose data in "molecules" (consisting of several atoms). Rank-one terms can only model data components that are proportional along columns, rows, ... and this assumption may not be realistic. On the other hand, block terms can model multidimensional sources, variations around mean activity, mildly nonlinear phenomena, drifts of setting points, frequency shifts, mildly convolutive mixtures, and so on. Such a molecular analysis is not possible in the matrix setting. For recent applications of BTDs, the reader can refer to [14, 15, 16, 17]. A comparison between the TKD, the CPD and the BTD for a problem of blind source separation can be found in [2, Example 2].

[4] The only exception is the rank-ones matrices whose CPD is obviously unique.

[5] This paragraph is based on a text written by Lieven De Lathauwer.

Chapter 2

Computation of the BTD

The previous chapter introduced some background on higher-order tensors. In particular, we defined an important tensor decomposition called the BTD. This chapter and the next are the heart of the present project in the sense that they form the main contribution of this work. In this chapter, we apply two first-order algorithms from the framework of optimization on matrix manifolds to compute the best approximation of a given third-order tensor by a BTD. The numerical tests are described in the next chapter.

The fundamental idea of this work is to solve the BTD approximation problem using the tools from optimization on matrix manifolds. This idea was already successfully applied in [7] for the TKD. Here, we try to extend the procedure to the BTD. As in [7], we will use variable projection to reduce the BTD approximation problem to a minimization problem whose objective function is defined on a Cartesian product of Stiefel manifolds. The big difference with [7] is that the variable projection can be performed only numerically. Therefore, we will be able to compute only the first-order derivatives of the objective function.

In the first section, we state formally the optimization problem to be solved. In the second section, we compute the derivative of the objective function. In the third section, we perform the variable projection that will allow us to keep only matrix variables in our optimization problem. In the fourth section, we take the constraints into account and compute the gradient of the objective function in the manifold meaning. In the fifth section, we present the two algorithms that we will use to solve the BTD problem. The last section contains additional remarks on the variable projection.

2.1 Problem formulation and solving strategy

The problem Let $(I_1, I_2, I_3) \in \mathbb{N}_*^3$ and $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. Given $R \in \mathbb{N}_*$ and $(R_1, R_2, R_3) \in \mathbb{N}_*^3$ such that $R_n \leq \operatorname{rank}_n(\mathcal{A})$ for each $n \in \{1, 2, 3\}$, we look for tensors $\mathcal{S}_1, \ldots, \mathcal{S}_R \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, matrices $\mathbf{U}_1, \ldots, \mathbf{U}_R \in \mathrm{St}(R_1, I_1)$, $\mathbf{V}_1, \ldots, \mathbf{V}_R \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W}_1, \ldots, \mathbf{W}_R \in \mathrm{St}(R_3, I_3)$ that minimize the cost function

$$
\begin{aligned}
f_{\mathcal{A}} : (\mathbb{R}^{R_1 \times R_2 \times R_3})^R \times (\mathbb{R}^{I_1 \times R_1})^R \times (\mathbb{R}^{I_2 \times R_2})^R \times (\mathbb{R}^{I_3 \times R_3})^R &\to \mathbb{R} \\
(\mathcal{S}_1, \ldots, \mathcal{S}_R, \mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R) &\mapsto \left\| \mathcal{A} - \sum_{r=1}^{R} \mathcal{S}_r \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r \right\|^2 .
\end{aligned}
$$

In other words, we want to solve the following optimization problem:

$$
\begin{aligned}
\underset{\mathcal{S}_r, \mathbf{U}_r, \mathbf{V}_r, \mathbf{W}_r}{\text{minimize}} \quad & f_{\mathcal{A}}(\mathcal{S}_1, \ldots, \mathcal{S}_R, \mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R) \\
\text{subject to} \quad & \mathbf{U}_r \in \mathrm{St}(R_1, I_1), \mathbf{V}_r \in \mathrm{St}(R_2, I_2), \mathbf{W}_r \in \mathrm{St}(R_3, I_3) \text{ for each } r \in \{1, \ldots, R\}.
\end{aligned}
\tag{2.1}
$$

Solving strategy As in [7], we shall solve problem (2.1) using variable projection, i.e., following these two steps:

1. computation of the optimal core tensors as functions of the factor matrices;
2. resolution of the optimization problem with only the factor matrices as variables.

2.2 Gradient of the objective function

In this section, we compute the gradient of the objective function. In order to do so in a convenient way, we first recall some basic facts on differentiation.

2.2.1 Reminder on differentiation

Given two vector spaces X and Y over a same field, we let $\mathrm{Lin}(X, Y)$ denote the vector space of linear mappings from X to Y.

Total derivative and gradient Let $(X, \langle \cdot, \cdot \rangle)$ be a pre-Hilbert space and let $\|\cdot\|$ denote the norm induced by the inner product $\langle \cdot, \cdot \rangle$. A function $f : X \to \mathbb{R}$ is *differentiable* at $x \in X$ if and only if there is $L \in \mathrm{Lin}(X, \mathbb{R})$ such that

$$\lim_{h \to 0} \frac{f(x+h) - f(x) - L(h)}{\|h\|} = 0,$$

which means that for every $\epsilon > 0$, there is $\delta > 0$ such that for any $h \in X$, $\|h\| \leq \delta$ implies

$$\frac{|f(x+h) - f(x) - L(h)|}{\|h\|} \leq \epsilon.$$

If such a L exists, it is unique, denoted by $\mathrm{D}f(x)$, and called the *total derivative* of f at x. The *gradient* of f at x is the only $g \in X$ such that

$$\mathrm{D}f(x)[h] = \langle g, h \rangle$$

for any $h \in X$; it is denoted by $(\operatorname{grad} f)(x)$. If f is differentiable at $x \in X$, then

$$\mathrm{D}f(x)[h] = \lim_{t \to 0} \frac{f(x+th) - f(x)}{t}.$$

for every $h \in X$.

An important example Let $f : X \to \mathbb{R} : x \mapsto f(x) := \|x\|^2$. For any $x, h \in X$ and any real $t \neq 0$,

$$\frac{f(x+th) - f(x)}{t} = \frac{2t\langle x, h \rangle + t^2 \|h\|^2}{t} = 2\langle x, h \rangle + t \|h\|^2 .$$

It follows that $\mathrm{D}f(x)[h] = 2\langle x, h \rangle$ and so that $(\operatorname{grad} f)(x) = 2x$.

Affine transformation Let $(X, \langle \cdot, \cdot \rangle_X)$ and $(Y, \langle \cdot, \cdot \rangle_Y)$ be two pre-Hilbert spaces, $g : Y \to \mathbb{R}$ be differentiable, $L \in \mathrm{Lin}(X, Y)$, $b \in Y$, $A : X \to Y : x \mapsto A(x) := L(x) + b$, and $f := g \circ A$. For any $x, h \in X$,

$$\begin{aligned}
\langle (\operatorname{grad} f)(x), h \rangle_X &= \lim_{t \to 0} \frac{f(x+th) - f(x)}{t} \\
&= \lim_{t \to 0} \frac{g(L(x) + b + tL(h)) - g(L(x) + b)}{t} \\
&= \langle (\operatorname{grad} g)(L(x) + b), L(h) \rangle_Y .
\end{aligned}$$

From now on, let us assume that X and Y have finite dimension so that L has an *adjoint*, which means that there is a (unique) $L^* \in \mathrm{Lin}(Y, X)$ such that

$$\langle y, L(x)\rangle_Y = \langle L^*(y), x\rangle_X$$

for any $x \in X$ and $y \in Y$. This allows us to conclude that for any $x \in X$,

$$(\operatorname{grad} f)(x) = L^*((\operatorname{grad} g)(L(x) + b)).$$

Adjoint of some useful linear mappings We give the adjoint of some useful linear mappings in $\mathbb{R}^{I_1 \times \cdots \times I_N}$ endowed with the standard inner product.

1. Let $n \in \{1, \ldots, N\}$ and $\mathbf{U} \in \mathbb{R}^{J_n \times I_n}$. The adjoint of

   $$L : \mathbb{R}^{I_1 \times \cdots \times I_N} \to \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N} : \mathcal{X} \mapsto \mathcal{X} \cdot_n \mathbf{U}$$

   is

   $$L^* : \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N} \to \mathbb{R}^{I_1 \times \cdots \times I_N} : \mathcal{Y} \mapsto \mathcal{Y} \cdot_n \mathbf{U}^{\mathrm{T}}.$$

2. Let $\mathbf{A} \in \mathbb{R}^{m \times p}$ and $\mathbf{B} \in \mathbb{R}^{q \times n}$. The adjoint of

   $$L : \mathbb{R}^{p \times q} \to \mathbb{R}^{m \times n} : \mathbf{X} \mapsto \mathbf{AXB}$$

   is

   $$L^* : \mathbb{R}^{m \times n} \to \mathbb{R}^{p \times q} : \mathbf{Y} \mapsto \mathbf{A}^{\mathrm{T}}\mathbf{YB}^{\mathrm{T}}.$$

2.2.2 Gradient of $f_{\mathcal{A}}$

After the reminder of the previous subsection, it is easy to compute the gradient of $f_{\mathcal{A}}$.

Gradient with respect to the core tensors We compute for each $i \in \{1, \ldots, R\}$

$$(\operatorname{grad}_{\mathcal{S}_i} f_{\mathcal{A}})(\ldots) = 2\left(\sum_{j=1}^{R} \mathcal{S}_j \cdot_1 (\mathbf{U}_i^{\mathrm{T}}\mathbf{U}_j) \cdot_2 (\mathbf{V}_i^{\mathrm{T}}\mathbf{V}_j) \cdot_3 (\mathbf{W}_i^{\mathrm{T}}\mathbf{W}_j) - \mathcal{A} \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}}\right).$$

For the sake of brevity, we wrote $(\ldots)$ instead of $(\mathcal{S}_1, \ldots, \mathcal{S}_R, \mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R)$ for the argument of $\operatorname{grad}_{\mathcal{S}_i} f_{\mathcal{A}}$. We will often use this shortcurt in the sequel.

Gradient with respect to the factor matrices Using Proposition 1.3.1 and Proposition 1.2.3, we can express $f_{\mathcal{A}}$ as follows:

$$\begin{aligned} f_{\mathcal{A}}(\mathcal{S}_1, \ldots, \mathcal{S}_R, \mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R) &= \left\| \sum_{r=1}^{R} \mathbf{U}_r (\mathbf{S}_r)_{(1)} (\mathbf{V}_r \otimes \mathbf{W}_r)^{\mathrm{T}} - \mathbf{A}_{(1)} \right\|^2 \\ &= \left\| \sum_{r=1}^{R} \mathbf{V}_r (\mathbf{S}_r)_{(2)} (\mathbf{W}_r \otimes \mathbf{U}_r)^{\mathrm{T}} - \mathbf{A}_{(2)} \right\|^2 \\ &= \left\| \sum_{r=1}^{R} \mathbf{W}_r (\mathbf{S}_r)_{(3)} (\mathbf{U}_r \otimes \mathbf{V}_r)^{\mathrm{T}} - \mathbf{A}_{(3)} \right\|^2. \end{aligned}$$

We deduce that (see also Proposition A.3.1)

$$
\begin{aligned}
(\operatorname{grad}_{\mathbf{U}_i} f_{\mathcal{A}})(\dots) &= 2\left(\sum_{j=1}^{R} \mathbf{U}_j(\mathbf{S}_j)_{(1)}(\mathbf{V}_j \otimes \mathbf{W}_j)^{\mathrm{T}} - \mathbf{A}_{(1)}\right)(\mathbf{V}_i \otimes \mathbf{W}_i)(\mathbf{S}_i)_{(1)}^{\mathrm{T}} \\
&= 2\left(\sum_{j=1}^{R} \mathbf{U}_j(\mathbf{S}_j)_{(1)}(\mathbf{V}_j^{\mathrm{T}}\mathbf{V}_i \otimes \mathbf{W}_j^{\mathrm{T}}\mathbf{W}_i)(\mathbf{S}_i)_{(1)}^{\mathrm{T}} - \mathbf{A}_{(1)}(\mathbf{V}_i \otimes \mathbf{W}_i)(\mathbf{S}_i)_{(1)}^{\mathrm{T}}\right) \\
(\operatorname{grad}_{\mathbf{V}_i} f_{\mathcal{A}})(\dots) &= 2\left(\sum_{j=1}^{R} \mathbf{V}_j(\mathbf{S}_j)_{(2)}(\mathbf{W}_j \otimes \mathbf{U}_j)^{\mathrm{T}} - \mathbf{A}_{(2)}\right)(\mathbf{W}_i \otimes \mathbf{U}_i)(\mathbf{S}_i)_{(2)}^{\mathrm{T}} \\
&= 2\left(\sum_{j=1}^{R} \mathbf{V}_j(\mathbf{S}_j)_{(2)}(\mathbf{W}_j^{\mathrm{T}}\mathbf{W}_i \otimes \mathbf{U}_j^{\mathrm{T}}\mathbf{U}_i)(\mathbf{S}_i)_{(2)}^{\mathrm{T}} - \mathbf{A}_{(2)}(\mathbf{W}_i \otimes \mathbf{U}_i)(\mathbf{S}_i)_{(2)}^{\mathrm{T}}\right) \\
(\operatorname{grad}_{\mathbf{W}_i} f_{\mathcal{A}})(\dots) &= 2\left(\sum_{j=1}^{R} \mathbf{W}_j(\mathbf{S}_j)_{(3)}(\mathbf{U}_j \otimes \mathbf{V}_j)^{\mathrm{T}} - \mathbf{A}_{(3)}\right)(\mathbf{U}_i \otimes \mathbf{V}_i)(\mathbf{S}_i)_{(3)}^{\mathrm{T}} \\
&= 2\left(\sum_{j=1}^{R} \mathbf{W}_j(\mathbf{S}_j)_{(3)}(\mathbf{U}_j^{\mathrm{T}}\mathbf{U}_i \otimes \mathbf{V}_j^{\mathrm{T}}\mathbf{V}_i)(\mathbf{S}_i)_{(3)}^{\mathrm{T}} - \mathbf{A}_{(3)}(\mathbf{U}_i \otimes \mathbf{V}_i)(\mathbf{S}_i)_{(3)}^{\mathrm{T}}\right).
\end{aligned}
$$

It is worthwhile noting that $\operatorname{grad} f_{\mathcal{A}}$ is *not* Lipschitz continuous.

2.2.3 Numerical check

Let us check numerically that our expression for $\operatorname{grad} f_{\mathcal{A}}$ is correct. Let

$$
\begin{aligned}
x &:= (\mathcal{S}_1, \dots, \mathcal{S}_R, \mathbf{U}_1, \dots, \mathbf{U}_R, \mathbf{V}_1, \dots, \mathbf{V}_R, \mathbf{W}_1, \dots, \mathbf{W}_R), \\
\dot{x} &:= (\dot{\mathcal{S}}_1, \dots, \dot{\mathcal{S}}_R, \dot{\mathbf{U}}_1, \dots, \dot{\mathbf{U}}_R, \dot{\mathbf{V}}_1, \dots, \dot{\mathbf{V}}_R, \dot{\mathbf{W}}_1, \dots, \dot{\mathbf{W}}_R) \\
&\in (\mathbb{R}^{R_1 \times R_2 \times R_3})^R \times (\mathbb{R}^{I_1 \times R_1})^R \times (\mathbb{R}^{I_2 \times R_2})^R \times (\mathbb{R}^{I_3 \times R_3})^R
\end{aligned}
$$

such that $\|\dot{x}\| = 1$. Let us look at the behaviour around zero of the function

$$
\phi : \mathbb{R} \to \mathbb{R} : t \mapsto \phi(t) := f_{\mathcal{A}}(x + t\dot{x}) - f(x) - \langle (\operatorname{grad} f_{\mathcal{A}})(x), \dot{x} \rangle t.
$$

The graph of $\log|\phi(t)|$ as a function of $\log t$ is plotted in Figure 2.1. We observe that the graph is a straight line whose slope is equal to 2. We deduce that $\phi(t) \propto t^2$ when t is small. This is only possible if our expression for the gradient is correct.

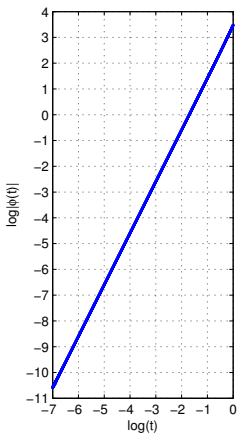


Figure 2.1: Graph of $\log|\phi(t)|$ as a function of $\log t$.

2.3 Variable projection

In this section, we compute the optimal core tensors for given factor matrices. In other words, we look for the function

$$\begin{aligned} g_{\mathcal{A}} : (\mathbb{R}^{I_1 \times R_1})^R \times (\mathbb{R}^{I_2 \times R_2})^R \times (\mathbb{R}^{I_3 \times R_3})^R &\to \mathbb{R} \\ (\mathbf{U}_1, \dots, \mathbf{U}_R, \mathbf{V}_1, \dots, \mathbf{V}_R, \mathbf{W}_1, \dots, \mathbf{W}_R) & \\ \mapsto \min_{\mathcal{S}_1, \dots, \mathcal{S}_R \in \mathbb{R}^{R_1 \times R_2 \times R_3}} & f_{\mathcal{A}}(\mathcal{S}_1, \dots, \mathcal{S}_R, \mathbf{U}_1, \dots, \mathbf{U}_R, \mathbf{V}_1, \dots, \mathbf{V}_R, \mathbf{W}_1, \dots, \mathbf{W}_R). \end{aligned}$$

Let us apply the first-order optimality condition. We just have to find where the gradient of $f_{\mathcal{A}}$ with respect to the core tensors is zero, which consists of solving the following equations:

$$\sum_{j=1}^{R} \mathcal{S}_j \cdot_1 (\mathbf{U}_i^{\mathrm{T}} \mathbf{U}_j) \cdot_2 (\mathbf{V}_i^{\mathrm{T}} \mathbf{V}_j) \cdot_3 (\mathbf{W}_i^{\mathrm{T}} \mathbf{W}_j) = \mathcal{A} \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}} \qquad \text{for each } i \in \{1, \dots, R\}.$$

This set of tensor equations forms a linear system. This can be highlighted using vectorization. Defining $\mathbf{a} := \mathrm{vec}(\mathcal{A})$, $\mathbf{P}_r := \mathbf{U}_r \otimes \mathbf{V}_r \otimes \mathbf{W}_r$ and $\mathbf{s}_r := \mathrm{vec}(\mathcal{S}_r)$ for each $r \in \{1, \dots, R\}$, the system can be rewritten in a more standard form thanks to Proposition 1.2.2:

$$\left[\mathbf{P}_i^{\mathrm{T}} \mathbf{P}_j\right]_{i,j=1}^{R} [\mathbf{s}_i]_{i,j=1}^{R,1} = \left[\mathbf{P}_i^{\mathrm{T}} \mathbf{a}\right]_{i,j=1}^{R,1}. \tag{2.2}$$

Note that the coefficient matrix is symmetric. We postpone the study of this system to section 2.6. By then, we assume that it has a unique solution denoted $\mathcal{S}_r^*(\mathbf{U}_1, \dots, \mathbf{U}_R, \mathbf{V}_1, \dots, \mathbf{V}_R, \mathbf{W}_1, \dots, \mathbf{W}_R)$, $r \in \{1, \dots, R\}$, in tensor form. In fact, we will always omit the argument of $\mathcal{S}_r^*$ in the sequel to make the notation simpler.

In our problem, this linear system will have to be solved only with factor matrices that are columnwise orthonormal. In that case, $\mathbf{P}_i^{\mathrm{T}} \mathbf{P}_i = \mathbf{I}_{R_1 R_2 R_3}$ for each $i \in \{1, \dots, R\}$. I wrote the Matlab function `CoreTensors` to solve this system.

An important remark The above reasoning can be seen as a generalization of [10, Theorem 4.1] and its proof for the case of the BTD instead of the TKD. The big difference is that here we do not have an explicit analytical expression for the optimal core tensors. We will see in the sequel that this will have two consequences. The first is discussed in the last paragraph of this section and the second, which is much more important, is discussed in the next section.

Problem reformulation using variable projection Letting $\bar{g}_{\mathcal{A}}$ denote the restriction of $g_{\mathcal{A}}$ on $\mathrm{St}(R_1, I_1) \times \mathrm{St}(R_2, I_2) \times \mathrm{St}(R_3, I_3)$, we can say that problem (2.1) is equivalent to the minimization of function $\bar{g}_{\mathcal{A}}$ over its domain. We study this problem in the next section.

A final comment* Even without analytical solution for system (2.2), we can go a step further in the reformulation of our problem. Indeed, we can compute

$$\begin{aligned} f_{\mathcal{A}}(\mathcal{S}_1, \dots, \mathcal{S}_R, \mathbf{U}_1, \dots, \mathbf{U}_R, \mathbf{V}_1, \dots, \mathbf{V}_R, \mathbf{W}_1, \dots, \mathbf{W}_R) &= \left\| \mathbf{a} - \sum_{i=1}^{R} \mathbf{P}_i \mathbf{s}_i \right\|^2 \\ &= \|\mathbf{a}\|^2 - 2\langle \mathbf{a}, \sum_{i=1}^{R} \mathbf{P}_i \mathbf{s}_i \rangle + \left\| \sum_{i=1}^{R} \mathbf{P}_i \mathbf{s}_i \right\|^2 \\ &= \|\mathbf{a}\|^2 - 2 \sum_{i=1}^{R} \langle \mathbf{a}, \mathbf{P}_i \mathbf{s}_i \rangle + \sum_{i=1}^{R} \sum_{j=1}^{R} \langle \mathbf{P}_j \mathbf{s}_j, \mathbf{P}_i \mathbf{s}_i \rangle \\ &= \|\mathbf{a}\|^2 - 2 \sum_{i=1}^{R} \mathbf{s}_i^{\mathrm{T}} \mathbf{P}_i^{\mathrm{T}} \mathbf{a} + \sum_{i=1}^{R} \sum_{j=1}^{R} \mathbf{s}_i^{\mathrm{T}} \mathbf{P}_i^{\mathrm{T}} \mathbf{P}_j \mathbf{s}_j. \end{aligned}$$

The solution $[\mathbf{s}_r^*(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)]_{i,j=1}^{R,1}$ of system (2.2) satisfies

$$\sum_{i=1}^{R}\sum_{j=1}^{R}(\mathbf{s}_i^*)^{\mathrm{T}}\mathbf{P}_i^{\mathrm{T}}\mathbf{P}_j\mathbf{s}_j^* = \sum_{i=1}^{R}(\mathbf{s}_i^*)^{\mathrm{T}}\mathbf{P}_i^{\mathrm{T}}\mathbf{a}.$$

(Recall that we omit the argument of $\mathbf{s}_r^*$ to make the notation simpler.) It follows that

$$\begin{aligned} g_{\mathcal{A}}(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R) &= \|\mathbf{a}\|^2 - \sum_{i=1}^{R}(\mathbf{s}_i^*)^{\mathrm{T}}\mathbf{P}_i^{\mathrm{T}}\mathbf{a} \\ &= \|\mathbf{a}\|^2 - \sum_{i=1}^{R}\langle \mathbf{a}, \mathbf{P}_i\mathbf{s}_i^*\rangle \\ &= \|\mathcal{A}\|^2 - \sum_{i=1}^{R}\langle \mathcal{A}, \mathcal{S}_i^* \cdot_1 \mathbf{U}_i \cdot_2 \mathbf{V}_i \cdot_3 \mathbf{W}_i\rangle \\ &= \|\mathcal{A}\|^2 - \langle \mathcal{A}, \sum_{i=1}^{R}\mathcal{S}_i^* \cdot_1 \mathbf{U}_i \cdot_2 \mathbf{V}_i \cdot_3 \mathbf{W}_i\rangle. \end{aligned}$$

This last equality allows us to reformulate our problem in the same way as in [10, Theorem 4.2]: problem (2.1) is equivalent to the maximization of the function

$$\begin{aligned} \tilde{g}_{\mathcal{A}} : (\mathbb{R}^{I_1\times R_1})^R \times (\mathbb{R}^{I_2\times R_2})^R \times (\mathbb{R}^{I_3\times R_3})^R &\to \mathbb{R} \\ (\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R) &\mapsto \langle \mathcal{A}, \sum_{r=1}^{R}\mathcal{S}_r^* \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r\rangle. \end{aligned}$$

subject to $\mathbf{U}_r \in \mathrm{St}(R_1, I_1)$, $\mathbf{V}_r \in \mathrm{St}(R_2, I_2)$, $\mathbf{W}_r \in \mathrm{St}(R_3, I_3)$ for every $r \in \{1,\dots,R\}$. However, this reformulation is useless since we cannot compute the derivative of $\tilde{g}_{\mathcal{A}}$ as we do not have an analytical expression for $\mathcal{S}_r^*$.

2.4 Gradient of $\bar{g}_{\mathcal{A}}$

Our goal now is to minimize the function $\bar{g}_{\mathcal{A}}$. An important feature of this function is that its domain is a Cartesian product of Stiefel manifolds. This will allow us to apply the "manifold version" of the gradient algorithm, as we will see in the next section. Before that, we need to define what is the gradient of a function defined on a Stiefel manifold. Indeed, it is not defined in the usual sense since the Stiefel manifold contains no interior point.

2.4.1 The Stiefel manifold

In this subsection, we introduce the minimum background needed to define what is the gradient of a function defined on the Stiefel manifold. We refer to [18, chapter 3] for the theory.

The notion of a manifold Informally, a *manifold* is a set $\mathcal{M}$ together with a set of one-to-one correspondences from subsets of $\mathcal{M}$ onto open subsets of $\mathbb{R}^d$ (d is called the *dimension* of the manifold). By convenience, we will simply talk about the manifold $\mathcal{M}$, without refering to the set of one-to-one correspondences. Vector spaces can be endowed with a manifold structure in a natural way and are then called *linear manifolds*. In particular, for positive integers n and p, $\mathbb{R}^{n\times p}$ is a linear manifold.

Embedded submanifolds As a subset of a linear space can be a linear subspace, a subset of a manifold $\mathcal{M}$ can be an *embedded submanifold* of $\mathcal{M}$ if it satisfies some topological conditions. It can be showed [18, subsection 3.3.2] that the set of columnwise orthonormal $n \times p$ matrices, with $1 \leq p \leq n$, is an embedded submanifold of $\mathbb{R}^{n\times p}$, called the *Stiefel manifold* and denoted by $\mathrm{St}(p, n)$. That is,

$$\mathrm{St}(p, n) := \{\mathbf{U} \in \mathbb{R}^{n\times p} : \mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}_p\}.$$

The dimension of $\mathrm{St}(p, n)$ is equal to $np - \frac{1}{2}p(p + 1)$. The Stiefel manifold is a compact set.

Tangent spaces An important feature of manifolds is that, even if they do not have necessarily a vector space structure, they can be locally approximated around each point by a vector space called the *tangent space* and whose elements are called the *tangent vectors*. The tangent space to a manifold $\mathcal{M}$ at $x \in \mathcal{M}$ is denoted by $T_x\mathcal{M}$. Thus, if ξ is a tangent vector to $\mathcal{M}$ at x, we write $\xi \in T_x\mathcal{M}$. Let us point out an important fact: if $\mathcal{M}$ is a linear manifold, then $T_x\mathcal{M}$ can be identified to $\mathcal{M}$ for each $x \in \mathcal{M}$. In particular, if $\mathcal{N}$ is an embedded submanifold of a linear manifold $\mathcal{M}$, $T_x\mathcal{N}$ can be regarded as a linear subspace of $T_x\mathcal{M}$ for any $x \in \mathcal{N}$. For example, it can be showed [18, Example 3.5.2] that the tangent space to $\mathrm{St}(p, n)$ at $\mathbf{X} \in \mathrm{St}(p, n)$ is

$$T_{\mathbf{X}}\mathrm{St}(p, n) = \{\mathbf{Z} \in \mathbb{R}^{n\times p} : \mathbf{X}^{\mathrm{T}}\mathbf{Z} + \mathbf{Z}^{\mathrm{T}}\mathbf{X} = \mathbf{0}\}. \tag{2.3}$$

Riemannian manifolds The last thing to do in order to define the gradient of a function defined on a manifold is to endow the tangent spaces of the manifold with an inner product. A manifold whose tangent spaces are endowed with a smoothly varying inner product is called a *Riemannian manifold*. Once again, things are easier for the case of linear manifolds. Indeed, if a vector space is endowed with an inner product, the tangent spaces to the associated linear manifold inherit this inner product; the linear manifold is then called an *Eucidean space*. In particular, the gradient of a function defined on a linear manifold is the gradient in the usual sense, i.e., with the linear manifold regarded as a vector space.

Riemannian submanifolds If $\mathcal{N}$ is an embedded submanifold of a Riemannian manifold $\mathcal{M}$, $T_x\mathcal{N}$ inherits the inner product in $T_x\mathcal{M}$ for any $x \in \mathcal{N}$ and $\mathcal{N}$ is then called a *Riemannian submanifold* of $\mathcal{M}$. Given $x \in \mathcal{N}$, the orthogonal complement of $T_x\mathcal{N}$ in $T_x\mathcal{M}$ is called the *normal space* to $\mathcal{N}$ at x and is denoted by $(T_x\mathcal{N})^{\perp}$. Any $\xi \in T_x\mathcal{M}$ can be uniquely decomposed into the sum of an element of $T_x\mathcal{N}$ and an element of $(T_x\mathcal{N})^{\perp}$:

$$\xi = \mathrm{P}_x\,\xi + \mathrm{P}_x^{\perp}\,\xi,$$

where we let P_x and $\mathrm{P}_x^{\perp}$ denote the orthogonal projections onto $T_x\mathcal{N}$ and $(T_x\mathcal{N})^{\perp}$, respectively. For example, the tangent spaces to the Stiefel manifold $\mathrm{St}(p, n)$ inherit the standard inner product of $\mathbb{R}^{n\times p}$. Therefore, the Stiefel manifold is a Riemannian submanifold of $\mathbb{R}^{n\times p}$. Moreover, any $\mathbf{Y} \in \mathbb{R}^{n\times p}$ can be projected onto $T_{\mathbf{X}}\mathrm{St}(p, n)$ tanks to the formula [18, equation (3.35)]

$$\mathrm{P}_{\mathbf{X}}\,\mathbf{Y} = (\mathbf{I}_n - \mathbf{X}\mathbf{X}^{\mathrm{T}})\mathbf{Y} + \mathbf{X}\,\mathrm{skew}(\mathbf{X}^{\mathrm{T}}\mathbf{Y}) \tag{2.4}$$

where $\mathrm{skew}(\mathbf{M}) := \frac{1}{2}(\mathbf{M} - \mathbf{M}^{\mathrm{T}})$ is the *skew-symmetric* part of $\mathbf{M}$.

Gradient of a function defined on a Riemannian submanifold Let $\mathcal{M}$ be a Riemannian manifold, $f : \mathcal{M} \to \mathbb{R}$ be defined on $\mathcal{M}$, and $\bar{f}$ be the restriction of f to a Riemannian submanifold $\overline{\mathcal{M}}$ of $\mathcal{M}$. It can be showed [18, subsection 3.6.1] that the gradient of $\bar{f}$ at $x \in \overline{\mathcal{M}}$ is equal to the projection of the gradient of f onto $T_x\overline{\mathcal{M}}$:

$$(\mathrm{grad}\,\bar{f})(x) = \mathrm{P}_x((\mathrm{grad}\,f)(x)).$$

Thus, if $f : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $\bar{f}$ is the restriction of f to $\mathrm{St}(p,n)$, then

$$(\operatorname{grad} \bar{f})(\mathbf{X}) = \mathrm{P}_{\mathbf{X}}((\operatorname{grad} f)(\mathbf{X}))$$

for each $\mathbf{X} \in \mathrm{St}(p,n)$. Moreover, as we said earlier, the gradient of f is the gradient in the usual sense since $\mathbb{R}^{n\times p}$ is a linear manifold.

2.4.2 Gradient of $\bar{g}_{\mathcal{A}}$

We are now ready to compute the gradient of $\bar{g}_{\mathcal{A}}$. This function is not defined on a Stiefel manifold but rather on a Cartesian product of Stiefel manifolds. This is not a problem: the tangent space of a Cartesian product is the Cartesian product of the tangent spaces (each of them being given by equation (2.3) in our case) and so the projection on the tangent space of the Cartesian product can be performed componentwise (using equation (2.4) in our case). The inner product in the Cartesian product is defined as the sum of the componentwise inner products. Indeed, if $(X, \langle\cdot,\cdot\rangle_X)$ and $(Y, \langle\cdot,\cdot\rangle_Y)$ are two pre-Hilbert spaces, it is readily checked that the function

$$((x_1,y_1),(x_2,y_2)) \in (X\times Y)\times(X\times Y) \mapsto \langle x_1,x_2\rangle_X + \langle y_1,y_2\rangle_Y$$

is an inner product on $X \times Y$.

Gradient of $g_{\mathcal{A}}$ Following the notation introduced in subsection 2.2.1, we let

$$\mathrm{D}g_{\mathcal{A}}(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R) \in \mathrm{Lin}((\mathbb{R}^{I_1\times R_1})^R \times (\mathbb{R}^{I_2\times R_2})^R \times (\mathbb{R}^{I_3\times R_3})^R, \mathbb{R})$$

denote the total derivative of $g_{\mathcal{A}}$ at point $(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)$. By the chain rule,

$$\begin{aligned}
&\mathrm{D}g_{\mathcal{A}}(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R) = \\
&\sum_{r=1}^{R} \mathrm{D}_{\mathcal{S}_r} f_{\mathcal{A}}(\mathcal{S}_1,\dots,\mathcal{S}_R,\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)\Big|_{(\mathcal{S}_1,\dots,\mathcal{S}_R)=(\mathcal{S}_r^*,\dots,\mathcal{S}_R^*)} \circ \mathrm{D}\mathcal{S}_r^* \\
&+ \mathrm{D}_{(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)} f_{\mathcal{A}}(\mathcal{S}_1,\dots,\mathcal{S}_R,\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)\Big|_{(\mathcal{S}_1,\dots,\mathcal{S}_R)=(\mathcal{S}_r^*,\dots,\mathcal{S}_R^*)}.
\end{aligned}$$

(Recall that we often write $\mathcal{S}_r^*$ instead of $\mathcal{S}_r^*(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)$.) For the optimal core tensors $\mathcal{S}_r^*$ that satisfy the system (2.2), the first term in the chain rule vanishes since the derivatives of $f_{\mathcal{A}}$ with respect to the core tensors $\mathcal{S}_r$ evaluated at $\mathcal{S}_r^*$ are zero. In conclusion,

$$\begin{aligned}
&\operatorname{grad} g_{\mathcal{A}}(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R) = \\
&\operatorname{grad}_{(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)} f_{\mathcal{A}}(\mathcal{S}_1,\dots,\mathcal{S}_R,\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)\Big|_{(\mathcal{S}_1,\dots,\mathcal{S}_R)=(\mathcal{S}_r^*,\dots,\mathcal{S}_R^*)}.
\end{aligned}$$

Recall that the latter was computed is subsection 2.2.2.

An important remark Thanks to the first-order optimality condition, we were able to compute the gradient of $g_{\mathcal{A}}$, even without any analytical expression for $\mathcal{S}_r^*$. However, it is clearly impossible to compute an analytical expression for the Hessian of $g_{\mathcal{A}}$. Therefore, we will have to restrict ourselves to first-order methods to compute the minimum of $g_{\mathcal{A}}$.

Gradient of $\bar{g}_{\mathcal{A}}$ We can now compute $\operatorname{grad}\bar{g}_{\mathcal{A}}$ by projecting $\operatorname{grad} g_{\mathcal{A}}$ on the tangent space of $\prod_{n=1}^{3}\operatorname{St}(R_n,I_n)^R$. As said earlier, the tangent space of a Cartesian product is the Cartesian product of the tangent spaces:

$$T_{(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)}\prod_{n=1}^{3}\operatorname{St}(R_n,I_n)^R=\prod_{r=1}^{R}T_{\mathbf{U}_r}\operatorname{St}(R_1,I_1)\times\prod_{r=1}^{R}T_{\mathbf{V}_r}\operatorname{St}(R_2,I_2)\times\prod_{r=1}^{R}T_{\mathbf{W}_r}\operatorname{St}(R_3,I_3).$$

Each term in the above product is given by equation (2.3). It follows that the projection on the tangent space can be performed componentwise using equation (2.4):

$$\begin{aligned}&\mathrm{P}_{(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)}(\dot{\mathbf{U}}_1,\dots,\dot{\mathbf{U}}_R,\dot{\mathbf{V}}_1,\dots,\dot{\mathbf{V}}_R,\dot{\mathbf{W}}_1,\dots,\dot{\mathbf{W}}_R)\\&=(\mathrm{P}_{\mathbf{U}_1}\dot{\mathbf{U}}_1,\dots,\mathrm{P}_{\mathbf{U}_R}\dot{\mathbf{U}}_R,\mathrm{P}_{\mathbf{V}_1}\dot{\mathbf{V}}_1,\dots,\mathrm{P}_{\mathbf{V}_R}\dot{\mathbf{V}}_R,\mathrm{P}_{\mathbf{W}_1}\dot{\mathbf{W}}_1,\dots,\mathrm{P}_{\mathbf{W}_R}\dot{\mathbf{W}}_R).\end{aligned}$$

Using the expression for $\operatorname{grad} f_{\mathcal{A}}$ computed is subsection 2.2.2 and the optimality condition written in matrix form, it is easy to check that

$$\begin{aligned}\mathbf{U}_i^{\mathrm{T}}(\operatorname{grad}_{\mathbf{U}_i}g_{\mathcal{A}})(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)&=\mathbf{0},\\\mathbf{V}_i^{\mathrm{T}}(\operatorname{grad}_{\mathbf{V}_i}g_{\mathcal{A}})(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)&=\mathbf{0},\\\mathbf{W}_i^{\mathrm{T}}(\operatorname{grad}_{\mathbf{W}_i}g_{\mathcal{A}})(\mathbf{U}_1,\dots,\mathbf{U}_R,\mathbf{V}_1,\dots,\mathbf{V}_R,\mathbf{W}_1,\dots,\mathbf{W}_R)&=\mathbf{0}\end{aligned}$$

for each $i\in\{1,\dots,R\}$. It follows that

$$\operatorname{grad}\bar{g}_{\mathcal{A}}=\operatorname{grad}g_{\mathcal{A}}.$$

2.5 Algorithms

As explained in the previous section, we have to restrict ourselves to optimization algorithms that use only first-order information. The goal of this section is to present the two algorithms that we will apply on our problem, namely the gradient algorithm and the conjugate gradients algorithm.

2.5.1 Gradient algorithm

Retraction Let us briefly explain what is the "manifold version" of the gradient algorithm. We refer to [18, chapter 4] for the details. Line-search methods in $\mathbb{R}^n$ are based on the update formula

$$x_{k+1}=x_k+t_k\eta_k,$$

where $\eta_k\in\mathbb{R}^n$ is called the *search direction* and $t_k\in\mathbb{R}$ is the *step size* [18]. In other words, we get x_{k+1} by moving from x_k in the direction η_k over a distance given by t_k. On a manifold $\mathcal{M}$, the search direction η_k is selected in the tangent space to $\mathcal{M}$ at x_k, $T_{x_k}\mathcal{M}$, and the notion of moving from x_k in the direction of η_k while staying on the manifold is generalized by the notion of a retraction mapping. Conceptually, a retraction R at $x\in\mathcal{M}$, denoted by R_x, is a mapping from $T_x\mathcal{M}$ to $\mathcal{M}$ with a local rigidity condition that preserves the gradients at x [18]. An illustration of this concept is given in Figure 2.2 where $\xi\in T_x\mathcal{M}$ is the moving direction.

We will not precise here what is the "local rigidity condition". We just explain how to construct retractions on embedded submanifolds with a particular attention for the Stiefel manifold. Let $\mathcal{M}$ be an embedded submanifold of a vector space $\mathcal{E}$. Recall that $T_x\mathcal{M}$ can be viewed as a linear subspace of $T_x\mathcal{E}$ which itself can be identified with $\mathcal{E}$. Therefore, the sum $x+\xi$ of $x\in\mathcal{M}$ and $\xi\in T_x\mathcal{M}$ makes sense and it is tempting to define the retraction $R_x(\xi)$ by "projecting" the point $x+\xi$ on $\mathcal{M}$. We use quotes because this "projection" does not necessarily have to be a projection in the usual sense, i.e., to minimize the distance to $\mathcal{M}$. Actually, defining $R_x(\xi)$ as the usual projection on $\mathcal{M}$ of $x+\xi$ results in a well-defined retraction, called the *projective retraction*

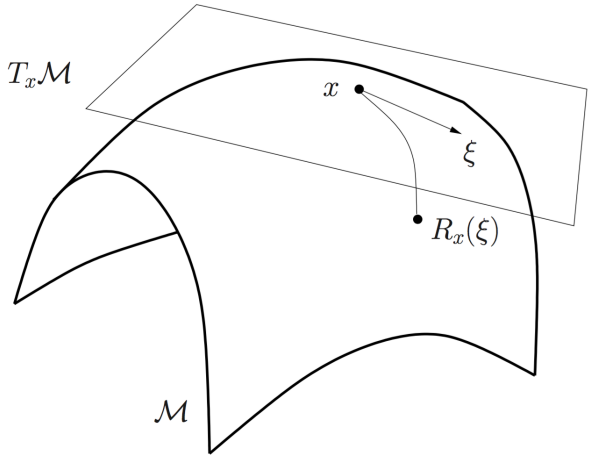


Figure 2.2: Schematic representation of a retraction [18, Figure 4.1].

[19, Proposition 3.2], but other choices are possible. In fact, the only constraint is to define a "projection" that satisfies the local rigidity condition and is computationally efficient.

For instance, a popular choice of retraction on the Stiefel manifold $\mathrm{St}(p,n)$ is the following [18, equation (4.8)]:

$$R_{\mathbf{X}}(\mathbf{Y}) := \mathrm{qf}(\mathbf{X}+\mathbf{Y}), \tag{2.5}$$

where $\mathrm{qf}(\mathbf{A})$ denotes the $\mathbf{Q}$ factor of the decomposition of $\mathbf{A} \in \mathbb{R}^{n\times p}$ with $\mathrm{rank}(\mathbf{A}) = p$ as $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q} \in \mathrm{St}(p,n)$ and $\mathbf{R}$ is an upper triangular $p \times p$ matrix with positive diagonal elements. We will use this retraction in our implementation. However, the manifold in our problem is a Cartesian product of Stiefel manifolds. Again, this is not an issue since the retraction can be performed componentwise:

$$\begin{aligned}&R_{(\mathbf{U}_1,\ldots,\mathbf{U}_R,\mathbf{V}_1,\ldots,\mathbf{V}_R,\mathbf{W}_1,\ldots,\mathbf{W}_R)}(\dot{\mathbf{U}}_1,\ldots,\dot{\mathbf{U}}_R,\dot{\mathbf{V}}_1,\ldots,\dot{\mathbf{V}}_R,\dot{\mathbf{W}}_1,\ldots,\dot{\mathbf{W}}_R) = \\ &(R_{\mathbf{U}_1}(\dot{\mathbf{U}}_1),\ldots,R_{\mathbf{U}_R}(\dot{\mathbf{U}}_R),R_{\mathbf{V}_1}(\dot{\mathbf{V}}_1),\ldots,R_{\mathbf{V}_R}(\dot{\mathbf{V}}_R),R_{\mathbf{W}_1}(\dot{\mathbf{W}}_1),\ldots,R_{\mathbf{W}_R}(\dot{\mathbf{W}}_R)).\end{aligned}$$

Projective retraction on the Stiefel manifold* Before describing the gradient algorithm in detail, it should be noted that the retraction (2.5) is not a projection in the usual sense. That is, the following inequality does *not* necessarily hold:

$$\|\mathbf{X} - \mathrm{qf}(\mathbf{X})\| \leq \|\mathbf{X} - \mathbf{Y}\|$$

for all $\mathbf{X} \in \mathbb{R}^{n\times p}$ with $\mathrm{rank}(\mathbf{X}) = p$ and $\mathbf{Y} \in \mathrm{St}(p,n)$. Let us look at what would be the projective retraction onto $\mathrm{St}(p,n)$.

Proposition 2.5.1 (Projection onto Stiefel manifolds)**.** *Let us write the SVD of* $\mathbf{X} \in \mathbb{R}^{n\times p}$*,* $1 \leq p \leq n$*. There are* $\mathbf{U} \in O_n$*,* $\mathbf{V} \in O_p$ *and*

$$\mathbf{\Sigma} := \begin{bmatrix}\mathrm{diag}(\sigma_1,\ldots,\sigma_p)\\ \mathbf{0}_{(n-p)\times p}\end{bmatrix}, \qquad \sigma_1 \geq \cdots \geq \sigma_p \geq 0,$$

such that $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$*. The projection of* $\mathbf{X}$ *onto* $\mathrm{St}(p,n)$ *is*

$$\mathbf{Y} := \mathbf{U}(:,1:p)\mathbf{V}^{\mathrm{T}}.$$

Proof. It is clear that $\mathbf{Y} \in \mathrm{St}(p,n)$. We need to show that

$$\|\mathbf{X} - \mathbf{Y}\| = \min_{\mathbf{Z} \in \mathrm{St}(p,n)} \|\mathbf{X} - \mathbf{Z}\|.$$

(Note that the minimum is well-defined since the objective function is continuous and $\mathrm{St}(p,n)$ is a compact set.) For any $\mathbf{Z} \in \mathrm{St}(p,n)$,

$$\|\mathbf{X} - \mathbf{Z}\|^2 = \|\mathbf{X}\|^2 - 2\langle \mathbf{X}, \mathbf{Z} \rangle + \|\mathbf{Z}\|^2 = \|\mathbf{X}\|^2 + p - 2\,\mathrm{tr}(\mathbf{Z}^{\mathrm{T}}\mathbf{X}).$$

Therefore,

$$\min_{\mathbf{Z} \in \mathrm{St}(p,n)} \|\mathbf{X} - \mathbf{Z}\| = \|\mathbf{X}\|^2 + p - 2 \max_{\mathbf{Z} \in \mathrm{St}(p,n)} \mathrm{tr}(\mathbf{Z}^{\mathrm{T}}\mathbf{X}).$$

Let us bound the last quantity. For any $\mathbf{Z} \in \mathrm{St}(p,n)$,

$$\mathrm{tr}(\mathbf{Z}^{\mathrm{T}}\mathbf{X}) = \mathrm{tr}(\mathbf{Z}^{\mathrm{T}}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}) = \mathrm{tr}((\mathbf{U}^{\mathrm{T}}\mathbf{Z}\mathbf{V})^{\mathrm{T}}\mathbf{\Sigma}) = \mathrm{tr}(\tilde{\mathbf{Z}}^{\mathrm{T}}\mathbf{\Sigma}) = \sum_{i=1}^{p} \tilde{\mathbf{Z}}(i,i)\sigma_i \leq \sum_{i=1}^{p} \sigma_i$$

since $\tilde{\mathbf{Z}} := \mathbf{U}^{\mathrm{T}}\mathbf{Z}\mathbf{V} \in \mathrm{St}(p,n)$ so that $\tilde{\mathbf{Z}}(i,i) \leq 1$ for each $i \in \{1, \ldots, p\}$. It is easy to check that this bound is reached for $\mathbf{Z} := \mathbf{Y}$. □

A condition ensuring the uniqueness of the projection is given in [19, Proposition 3.4]. (Note that the well-known result that states that the projection onto a closed convex set is unique, see e.g. [20, Theorem 2.2.6], does not apply since the Stiefel manifold is not a convex set.)

The gradient method Line-search methods to minimize a function $f : \mathcal{M} \to \mathbb{R}$ are based on the update formula

$$x_{k+1} = R_{x_k}(t_k \eta_k),$$

where $\eta_k \in T_{x_k}\mathcal{M}$ and $t_k \in \mathbb{R}$. The algorithm is defined by the choice of three ingredients: a retraction R_{x_k}, a search direction η_k and a step size t_k. As mentioned earlier, we shall use the retraction (2.5) in our implementation. On the other hand, the gradient method consists of choosing $\eta_k := -(\mathrm{grad}\, f)(x_k)$. Thus, the only thing to specify is the step size t_k. For that, we will use a backtracking strategy, as presented in [21, Algorithm 3.1].

The backtracking strategy Assume we are at the kth iteration. We want to find $t_k > 0$ such that $f(R_{x_k}(-t_k(\mathrm{grad}\, f)(x_k)))$ is sufficiently small compared to $f(x_k)$. This can be achieved thanks to the Armijo rule: given $\bar{\alpha} > 0$, $\beta, \sigma \in (0,1)$ and $t_k := \bar{\alpha}$, we iterate $t_k := \beta t_k$ until

$$f(R_{x_k}(-t_k(\mathrm{grad}\, f)(x_k))) \leq f(x_k) - \sigma t_k \left\|(\mathrm{grad}\, f)(x_k)\right\|^2.$$

In practice, we set $\bar{\alpha} := 0.2$, $\sigma := 10^{-3}$ and we perform at most 10 iterations in the backtracking loop using three different values for β: $\beta := 0.5$ for the 3 first iterations, $\beta := 0.1$ for the 3 next ones and $\beta := 0.01$ for the other ones. For these parameters, Table 2.1 gives the step length as a function of the number of iterations in the backtracking loop.

Convergence The procedure described in the previous paragraph corresponds to [18, Algorithm 1] with $c := 1$ and equality in [18, equation (4.12)]. In our problem, the domain of our cost function is compact since it is a Cartesian product of Stiefel manifolds. Therefore, [18, Corollary 4.3.2] applies and ensures that

$$\lim_{k \to \infty} \|(\mathrm{grad}\, f)(x_k)\| = 0.$$

In view of this result, it seems natural to stop the algorithm as soon as the norm of the gradient becomes smaller than a given quantity $\epsilon > 0$.

It is worthwhile noting that this result does not say anything about the speed of convergence. In fact, it is possible to estimate the asymptotic speed of convergence but it requires to study the spectrum of the Hessian of the objective function [18, Theorem 4.5.6]. As we do not have an analytical expression for the Hessian, this result is not useful in our case.

Iterations in the backtracking loop	Step length
1	0.2
2	0.1
3	$5 \cdot 10^{-2}$
4	$5 \cdot 10^{-3}$
5	$5 \cdot 10^{-4}$
6	$5 \cdot 10^{-5}$
7	$5 \cdot 10^{-7}$
8	$5 \cdot 10^{-9}$
9	$5 \cdot 10^{-11}$
10	$5 \cdot 10^{-13}$

Table 2.1: Step length as a function of the number of iterations in the backtracking loop.

2.5.2 Conjugate gradients

Vector transport Let us briefly explain what is the "manifold version" of the conjugate gradients algorithm. We refer to [18, section 8.3] for the details. The conjugate gradients method used to minimize a function $f : \mathbb{R}^n \to \mathbb{R}$ is based on the update formula

$$x_{k+1} := x_k + \alpha_k p_k$$

where the search direction p_k is computed as

$$p_k := -(\operatorname{grad} f)(x_k) + \beta_k p_{k-1}.$$

Several choices are possible for the coefficient β_k in the last formula. Here we chose the Fletcher-Reeves formula:

$$\beta_k = \left(\frac{\|(\operatorname{grad} f)(x_k)\|}{\|(\operatorname{grad} f)(x_{k-1})\|} \right)^2.$$

Let us see how to adapt those formulas for the minimization of a function $f : \mathcal{M} \to \mathbb{R}$ where $\mathcal{M}$ is a Riemannian manifold. As for the gradient method, the update formula for the iterate can be generalized tanks to the notion of retraction:

$$x_{k+1} := R_{x_k}(\alpha_k p_k).$$

However, another problem remains: the update formula for the search direction does not make sense anymore since $(\operatorname{grad} f)(x_k)$ and p_{k-1} respectively belong to $T_{x_k}\mathcal{M}$ and $T_{x_{k-1}}\mathcal{M}$ which are two different vector spaces. Roughly speaking, we would need to bring $p_{k-1} \in T_{x_{k-1}}\mathcal{M}$ to $T_{x_k}\mathcal{M}$, where $x_k := R_{x_{k-1}}(\alpha_{k-1}p_{k-1})$. The notion of *vector transport* meets this need. If $x \in \mathcal{M}$ and $\eta, \xi \in T_x\mathcal{M}$, then we can bring ξ to $T_{R_x(\eta)}\mathcal{M}$ using a vector transport $\mathcal{T}$ on $\mathcal{M}$. More precisely, $\mathcal{T}_\eta(\xi) \in T_{R_x(\eta)}\mathcal{M}$ is the vector ξ transported from $T_x\mathcal{M}$ to $T_{R_x(\eta)}\mathcal{M}$. An illustration of this concept is given in Figure 2.3.

We will not define precisely what the vector transport is. We just focus on the construction of a vector transport on Riemannian submanifolds. If $\mathcal{M}$ is an embedded submanifold of a Euclidean space $\mathcal{E}$ and R is a retraction on $\mathcal{M}$, then the vector transport on $\mathcal{M}$ can be defined as

$$\mathcal{T}_\eta(\xi) := \mathrm{P}_{R_x(\eta)}\, \xi$$

for any $\eta, \xi \in T_x\mathcal{M}$, where P_x denotes the orthogonal projection onto $T_x\mathcal{M}$. For instance, if R is a retraction on the Stiefel manifold $\mathrm{St}(p, n)$, then the vector transport can be defined using the formula (2.4) as

$$\mathcal{T}_{\mathbf{U}}(\mathbf{V}) := (\mathbf{I}_n - \mathbf{Y}\mathbf{Y}^{\mathrm{T}})\mathbf{V} + \mathbf{Y}\operatorname{skew}(\mathbf{Y}^{\mathrm{T}}\mathbf{V})$$

for any $\mathbf{U}, \mathbf{V} \in T_{\mathbf{X}}\mathrm{St}(p, n)$, where $\mathbf{Y} := R_{\mathbf{X}}(\mathbf{U})$.

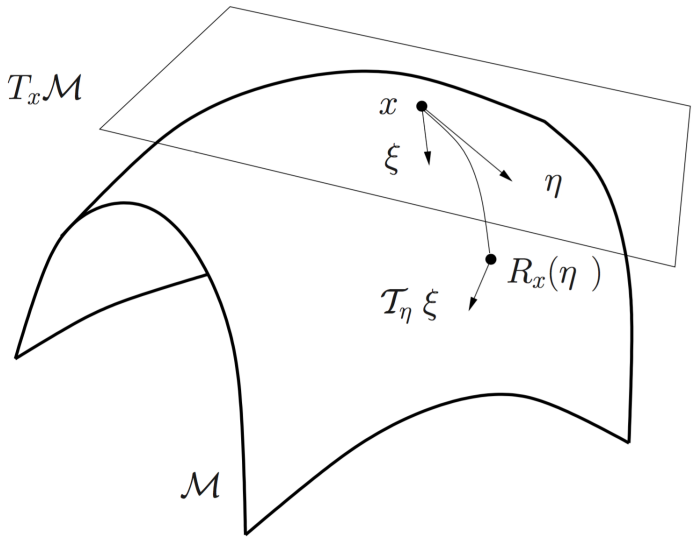


Figure 2.3: Schematic representation of a vector transport [18, Figure 8.1].

The conjugate gradients method Thanks to the notion of vector transport, the update formula for the search direction becomes

$$p_k := -(\operatorname{grad} f)(x_k) + \beta_k \mathcal{T}_{\alpha_{k-1} p_{k-1}}(p_{k-1}).$$

To minimize a general nonlinear function from $\mathbb{R}^n$ to $\mathbb{R}$, it is needed to set $\beta_k := 0$ when n divides k (see e.g. [20, subsection 1.3.2]). In the same way, to minimize a general nonlinear function from $\mathcal{M}$ to $\mathbb{R}$, it is needed to set $\beta_k := 0$ when the dimension of $\mathcal{M}$ divides k. As the dimension of a Cartesian product of vector spaces is the sum of their dimension, it holds that

$$\dim\left(\prod_{n=1}^{3} \operatorname{St}(R_n, I_n)^R\right) = R\sum_{n=1}^{3} \dim \operatorname{St}(R_n, I_n) = R\sum_{n=1}^{3}\left(I_n R_n - \frac{R_n(R_n+1)}{2}\right).$$

The conjugate gradients algorithm to minimize a real-valued differentiable function f defined on a Riemannian manifold $\mathcal{M}$ endowed with a vector transport $\mathcal{T}$ associated with a retraction R is presented in [18, Algorithm 13]. In our implementation, we use the same backtracking strategy as for the gradient method.

2.6 Remarks on the variable projection*

In this section, we study the linear system (2.2). First of all, let us note that this is a linear least-squares system. Therefore, it always admits a solution. The natural question then is about the uniqueness of the solution.

2.6.1 Uniqueness

BTD with two terms Firstly, let us study the simple case $R := 2$. Let $\mathbf{M} := \mathbf{P}_1^{\mathrm{T}}\mathbf{P}_2$. Let us apply elementary transformations on system (2.2):

$$\begin{bmatrix} \mathbf{I} & \mathbf{M} \\ \mathbf{M}^{\mathrm{T}} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1^{\mathrm{T}}\mathbf{a} \\ \mathbf{P}_2^{\mathrm{T}}\mathbf{a} \end{bmatrix} \iff \begin{bmatrix} \mathbf{I} & -\mathbf{M} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{I} & \mathbf{M} \\ \mathbf{M}^{\mathrm{T}} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{M} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{P}_1^{\mathrm{T}}\mathbf{a} \\ \mathbf{P}_2^{\mathrm{T}}\mathbf{a} \end{bmatrix}$$
$$\iff \begin{bmatrix} \mathbf{I} - \mathbf{M}\mathbf{M}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{M}^{\mathrm{T}} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1^{\mathrm{T}}\mathbf{a} - \mathbf{M}\mathbf{P}_2^{\mathrm{T}}\mathbf{a} \\ \mathbf{P}_2^{\mathrm{T}}\mathbf{a} \end{bmatrix}.$$

Thus, solving system (2.2) reduces to solving the following system:

$$(\mathbf{I} - \mathbf{M}\mathbf{M}^{\mathrm{T}})\mathbf{s}_1 = (\mathbf{P}_1^{\mathrm{T}} - \mathbf{M}\mathbf{P}_2^{\mathrm{T}})\mathbf{a}.$$

Solution is unique if and only if $\mathbf{I} - \mathbf{M}\mathbf{M}^{\mathrm{T}}$ is invertible, i.e., if and only if 1 is not singular value of $\mathbf{M}$.

Singular values of M Let us recall that

$$\mathbf{M} = \mathbf{P}_1^{\mathrm{T}}\mathbf{P}_2 = (\mathbf{U}_1^{\mathrm{T}}\mathbf{U}_2) \otimes (\mathbf{V}_1^{\mathrm{T}}\mathbf{V}_2) \otimes (\mathbf{W}_1^{\mathrm{T}}\mathbf{W}_2).$$

Thus, in view of Proposition A.3.2, we just have to study the singular values of $\mathbf{U}^{\mathrm{T}}\mathbf{V}$ for $\mathbf{U}, \mathbf{V} \in \mathrm{St}(R, I)$.

Proposition 2.6.1. *For any $\mathbf{U}, \mathbf{V} \in \mathrm{St}(R, I)$, the singular values of $\mathbf{U}^{\mathrm{T}}\mathbf{V}$ lie in $[0, 1]$. Moreover, a singular value is equal to 1 if and only if $\mathrm{range}(\mathbf{U}) \cap \mathrm{range}(\mathbf{V}) \neq \{0\}$.*

Proof. Let us write the SVD of $\mathbf{U}^{\mathrm{T}}\mathbf{V}$. There are $\tilde{\mathbf{U}}, \tilde{\mathbf{V}} \in O_R$ such that $\mathbf{U}^{\mathrm{T}}\mathbf{V} = \tilde{\mathbf{U}}\mathbf{\Sigma}\tilde{\mathbf{V}}^{\mathrm{T}}$ where $\mathbf{\Sigma} = \mathrm{Diag}(\sigma_1, \ldots, \sigma_R)$ with $\sigma_1 \geq \cdots \geq \sigma_R \geq 0$. Let us define $\hat{\mathbf{U}} := \mathbf{U}\tilde{\mathbf{U}}$ and $\hat{\mathbf{V}} := \mathbf{V}\tilde{\mathbf{V}}$. We have $\mathrm{range}(\mathbf{U}) = \mathrm{range}(\hat{\mathbf{U}})$ and $\hat{\mathbf{U}} \in \mathrm{St}(R, I)$, and also $\mathrm{range}(\mathbf{V}) = \mathrm{range}(\hat{\mathbf{V}})$ and $\hat{\mathbf{V}} \in \mathrm{St}(R, I)$. We see that $\mathbf{\Sigma} = \hat{\mathbf{U}}^{\mathrm{T}}\hat{\mathbf{V}}$. That is, writing $\hat{\mathbf{U}} = [\hat{\mathbf{u}}_1 \ \ldots \ \hat{\mathbf{u}}_R]$ and $\hat{\mathbf{V}} = [\hat{\mathbf{v}}_1 \ \ldots \ \hat{\mathbf{v}}_R]$, $\sigma_i = \hat{\mathbf{u}}_i^{\mathrm{T}}\hat{\mathbf{v}}_i$ for every $i \in \{1, \ldots, R\}$. By the Cauchy-Schwarz inequality, we have $\sigma_i \leq \|\hat{\mathbf{u}}_i\| \, \|\hat{\mathbf{v}}_i\| = 1$. Moreover, we have equality if and only if $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{v}}_i$ are linearly dependent, i.e., if and only if $\mathrm{range}(\mathbf{U}) \cap \mathrm{range}(\mathbf{V}) \neq \{0\}$. □

The argument could be refined to obtain more precise informations about the subspace $\mathrm{range}(\mathbf{U}) \cap \mathrm{range}(\mathbf{V})$ (see, e.g., [1, Theorem 6.4.2]) but this is not needed for our purposes.

Keeping the notation used in the preceding proof, let us recall that the *principal angles* between $\mathrm{range}(\mathbf{U})$ and $\mathrm{range}(\mathbf{V})$ can be defined by

$$\theta_i := \arccos(\sigma_i) \in \left[0, \frac{\pi}{2}\right]$$

for each $i \in \{1, \ldots, R\}$ [1, subsection 6.4.3]. With this definition, we see that 1 is a singular value of $\mathbf{U}^{\mathrm{T}}\mathbf{V}$ if and only if $\mathrm{range}(\mathbf{U})$ and $\mathrm{range}(\mathbf{V})$ have a zero principal angle, i.e., $\mathrm{range}(\mathbf{U})$ and $\mathrm{range}(\mathbf{V})$ have a common vector.

Conclusion The conclusion is that 1 is a singular value of $\mathbf{M}$ if and only if $\mathrm{range}(\mathbf{U}_1) \cap \mathrm{range}(\mathbf{U}_2) \neq \{0\}$, $\mathrm{range}(\mathbf{V}_1) \cap \mathrm{range}(\mathbf{V}_2) \neq \{0\}$ and $\mathrm{range}(\mathbf{W}_1) \cap \mathrm{range}(\mathbf{W}_2) \neq \{0\}$.

Generalization For a BTD with more than two terms, the previous reasoning is harder to carry on. In particular, it seems impossible to provide "simple" conditions to ensure the uniqueness of the solution.

2.6.2 Orthogonal transformations

Theorem 2.6.2. *Function $g_{\mathcal{A}}$ is invariant under orthogonal transformations, i.e.,*

$$\begin{aligned} g_{\mathcal{A}}(\mathbf{U}_1\mathbf{Q}_1^{(1)}, \ldots, \mathbf{U}_R\mathbf{Q}_R^{(1)}, \mathbf{V}_1\mathbf{Q}_1^{(2)}, \ldots, \mathbf{V}_R\mathbf{Q}_R^{(2)}, \mathbf{W}_1\mathbf{Q}_1^{(3)}, \ldots, \mathbf{W}_R\mathbf{Q}_R^{(3)}) \\ = g_{\mathcal{A}}(\mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R) \end{aligned}$$

for any $\mathbf{Q}_i^{(j)} \in O_{R_j}$, $i \in \{1, \ldots, R\}$, $j \in \{1, 2, 3\}$.

Proof. Assume matrices $\mathbf{U}_r, \mathbf{V}_r, \mathbf{W}_r$ are replaced respectively by $\mathbf{U}_r\mathbf{Q}_r^{(1)}, \mathbf{V}_r\mathbf{Q}_r^{(2)}, \mathbf{W}_r\mathbf{Q}_r^{(3)}$, $r \in \{1, \ldots, R\}$. The critical points of $f_{\mathcal{A}}$ are then solutions of the system

$$\left(\sum_{r=1}^{R} \mathcal{S}_r \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r\right) \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}} = \mathcal{A} \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}} \quad \text{for each} \quad i \in \{1, \ldots, R\}.$$

Using Proposition 1.2.1, this system becomes

$$\left(\left(\sum_{r=1}^{R} \left(\mathcal{S}_r \cdot_1 \mathbf{Q}_r^{(1)} \cdot_2 \mathbf{Q}_r^{(2)} \cdot_3 \mathbf{Q}_r^{(3)}\right) \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r\right) \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}}\right) \cdot_1 \mathbf{Q}_r^{(1)^{\mathrm{T}}} \cdot_2 \mathbf{Q}_r^{(2)^{\mathrm{T}}} \cdot_3 \mathbf{Q}_r^{(3)^{\mathrm{T}}}$$
$$= \left(\mathcal{A} \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}}\right) \cdot_1 \mathbf{Q}_r^{(1)^{\mathrm{T}}} \cdot_2 \mathbf{Q}_r^{(2)^{\mathrm{T}}} \cdot_3 \mathbf{Q}_r^{(3)^{\mathrm{T}}} \quad \text{for each} \quad i \in \{1, \ldots, R\}$$

or

$$\left(\left(\sum_{r=1}^{R} \left(\mathcal{S}_r \cdot_1 \mathbf{Q}_r^{(1)} \cdot_2 \mathbf{Q}_r^{(2)} \cdot_3 \mathbf{Q}_r^{(3)}\right) \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r\right) \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}}\right) = \left(\mathcal{A} \cdot_1 \mathbf{U}_i^{\mathrm{T}} \cdot_2 \mathbf{V}_i^{\mathrm{T}} \cdot_3 \mathbf{W}_i^{\mathrm{T}}\right)$$

for each $i \in \{1, \ldots, R\}$. Using again Proposition 1.2.1, we also have

$$g_{\mathcal{A}}(\mathbf{U}_1\mathbf{Q}_1^{(1)}, \ldots, \mathbf{U}_R\mathbf{Q}_R^{(1)}, \mathbf{V}_1\mathbf{Q}_1^{(2)}, \ldots, \mathbf{V}_R\mathbf{Q}_R^{(2)}, \mathbf{W}_1\mathbf{Q}_1^{(3)}, \ldots, \mathbf{W}_R\mathbf{Q}_R^{(3)})$$
$$= \|A\|^2 - \langle \mathcal{A}, \sum_{r=1}^{R} \left(\mathcal{S}_r \cdot_1 \mathbf{Q}_r^{(1)} \cdot_2 \mathbf{Q}_r^{(2)} \cdot_3 \mathbf{Q}_r^{(3)}\right) \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r \rangle.$$

Replacing $\mathcal{S}_r \cdot_1 \mathbf{Q}_r^{(1)} \cdot_2 \mathbf{Q}_r^{(2)} \cdot_3 \mathbf{Q}_r^{(3)}$ by $\tilde{\mathcal{S}}_r$, we see that we recover the same equations as before the orthogonal transformations. □

Chapter 3

Numerical results

In this chapter, we perfom numerical tests to evaluate the two methods that were presented in the previous chapter, namely the gradient method and the conjugate gradients method. In the first section, we study the influence of different parameters on the behaviour of the methods. In the second section, we compare our two methods with the already available ones. All our Matlab implementations are listed in Appendix B.

Before getting to that, let us discuss the kind of numerical tests we are going to perform. When we try to approximate a mathematical object by another one, both objects must be close enough to each other. In other words, when you try to replace a possibly complex object with a model, the latter must be well chosen. For instance, trying to approximate a damped sine signal with a sine signal will not work because these two signals are too different. In the same way, if we try to approximate a randomly generated tensor by a particular BTD, there is a priori no reason for that BTD to be a good model for the tensor. It follows that the approximation may not be satisfactory. In brief, looking for a structure that is not present will probably not work.

3.1 Recovering a known BTD

Our first test will be to construct a BTD by ourselves and then try to recover it. This test has two advantages. First, we try to recover a structure that is really present. Secondly, recovering a known BTD is a problem whose we know exactly the solution, which is a big advantage to evalute our methods as precisely as possible. Let us describe the test in detail. We select

- an integer $R \geq 2$,
- $(I_1, I_2, I_3), (R_1, R_2, R_3) \in \mathbb{N}_*^3$ such that $R_n \leq I_n$ for each $n \in \{1, 2, 3\}$,
- $\mathcal{S}_r \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ for each $r \in \{1, \ldots, R\}$ according to the standard normal distribution, i.e., $\mathcal{S}_r :=$ `randn(`R_1`,`R_2`,`R_3`)` in Matlab,
- $\mathbf{U}_r \in \mathrm{St}(R_1, I_1)$, $\mathbf{V}_r \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W}_r \in \mathrm{St}(R_3, I_3)$ for each $r \in \{1, \ldots, R\}$ according to the standard normal distribution, i.e., $\mathbf{U}_r :=$ `qf(randn(`I_1`,`R_1`))` in Matlab,

and we define

$$\mathcal{A} := \sum_{r=1}^{R} \mathcal{S}_r \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r.$$

After that, we select a starting iterate and we run our two methods. Unless otherwise specified,

- the starting iterate is chosen randomly according to the standard normal distribution,
- our methods stop as soon as the norm of the gradient becomes smaller than $5 \cdot 10^{-14}$.

In the four first subsections, we study the effect of different parameters on the behavior of the methods. The fifth subsection discusses the convergence of the methods. The two next subsections are given for further information. The sixth subsection deals with the choice of the retraction. In the seventh subsection, we highlight the benefit of variable projection. Finally, the last subsection draws a conclusion on the results.

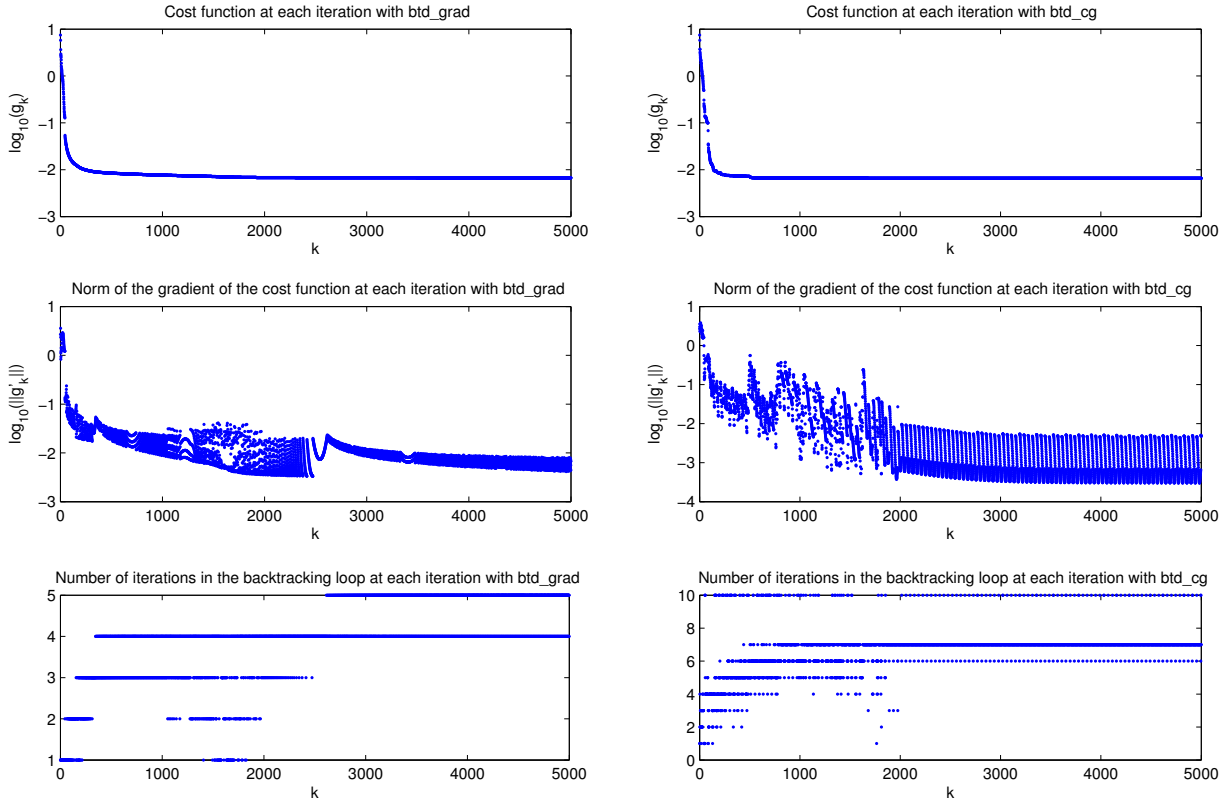
3.1.1 Influence of the starting iterate

Behavior of the methods with different randomly selected starting iterates

The following observation is probably the first to be pointed out: the behavior of both methods strongly depends on the starting point. Let us illustrate this from a simple example with $R := 2$, $I_n := 5$ and $R_n := 2$ for each $n \in \{1, 2, 3\}$. We stop the process as soon as the norm of the gradient becomes smaller than $5 \cdot 10^{-14}$ or 5000 iterations have been performed. The cost function and the norm of its gradient are plotted for four different randomly selected starting points in Figure 3.1. Each subfigure corresponds to a different starting point.

Figure 3.1a We observe that none of the two algorithms converges. For both of them, the cost function decreases to about 10^{-2} and stagnates near this value. The norm of the gradient of the cost function falls just below 10^{-2} for the gradient algorithm and just below 10^{-3} for the conjugate gradients algorithm.

We also see that the number of iterations performed in the backtracking loop increases with the number of iterations in the main loop, denoted k on the plots. Roughly speaking, the gradient algorithm performs 4 or 5 iterations in the backtracking loop from the 3000th iteration while the conjugate gradients algorithm performs 6, 7 or even 10 iterations in the backtracking loop from the 2500th iteration. This is because the sufficient decrease condition is difficult to satisfy. In view of those numbers, recalling Table 2.1, it is not surprising that the algorithms stagnate since the step lengths are really small.



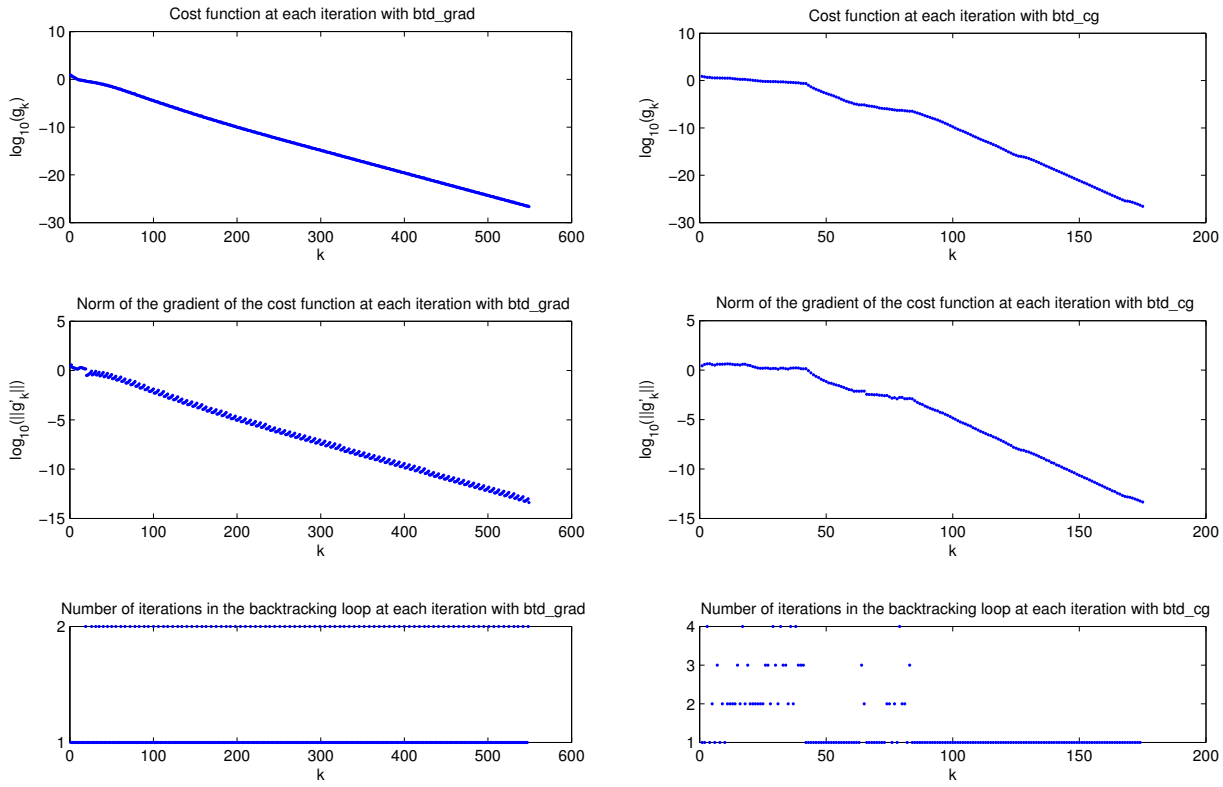
(a) None of the two algorithms converges.

Figure 3.1: Runs of `btd_grad` and `btd_cg` with 4 different starting iterates.

Figure 3.1b On the contrary, we observe here that both algorithms converge, which means, as a reminder, that the norm of the gradient of the cost function falls down $5 \cdot 10^{-14}$. We also see that the objective function falls below 10^{-25}.

In contrast with Figure 3.1a, we see that the number of iterations in the backtracking loop is almost always equal to 1 for both algorithms. This shows that the algorithms advance easily.

We should also mention that all the principal angles between the ranges of the exact factor matrices (i.e., used to construct the BTD) and the factor matrices computed by both algorithms are equal to 0 which means that the ranges are the same. For instance, if we let $\mathbf{U}_1^{\text{grad}}$ denote the approximation of $\mathbf{U}_1$ computed by the gradient algorithm, then the R_1 singular values of $\mathbf{U}_1^{\text{T}}\mathbf{U}_1^{\text{grad}}$ are equal to 1 which means that $\text{range}(\mathbf{U}_1) = \text{range}(\mathbf{U}_1^{\text{grad}})$.

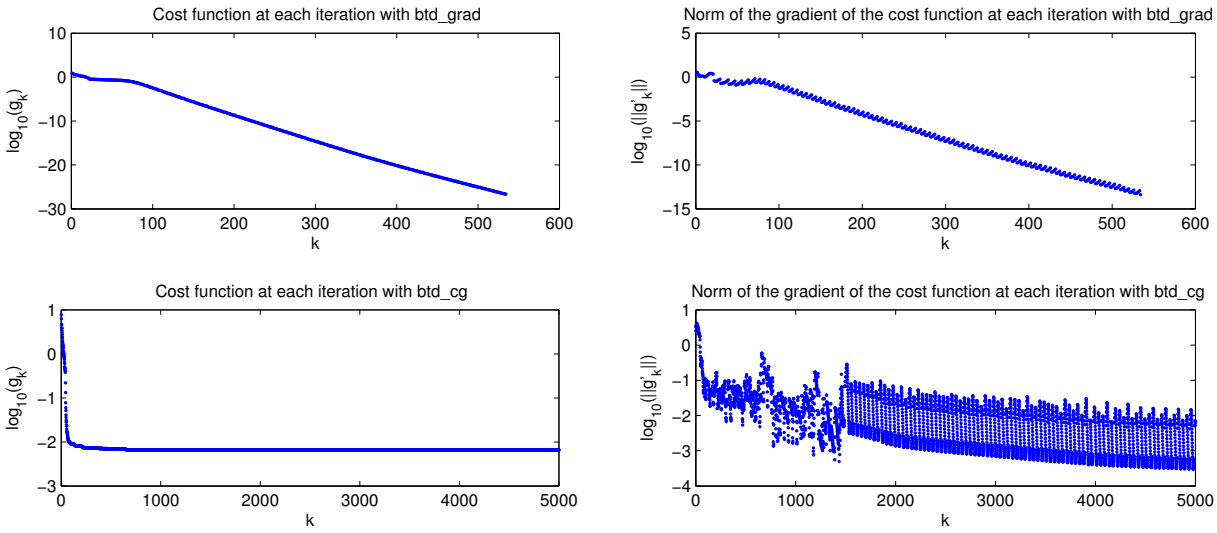


(b) Both algorithms converge.

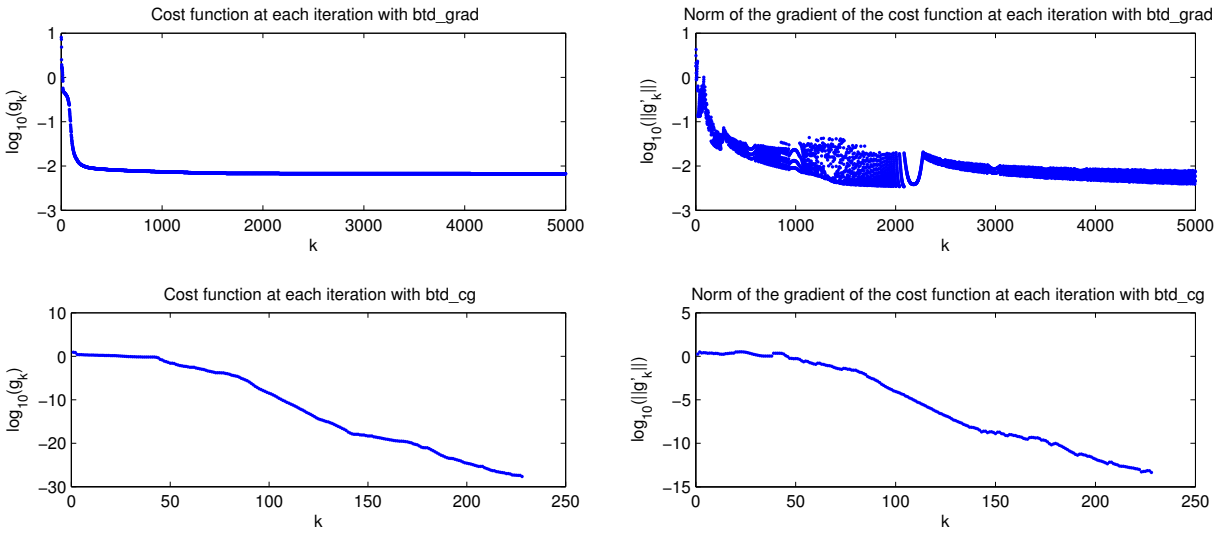
Figure 3.1: Runs of `btd_grad` and `btd_cg` with 4 different starting iterates.

Figure 3.1(c)-(d) We see that only one of the two algorithms converges. This means that a starting iterate may be good for the gradient algorithm but not for the conjugate gradients algorithm, and vice versa, which may seem surprising at first sight. Otherwise, we observe the same behaviors as in Figure 3.1(a)-(b).

Conclusion We have seen that the behavior of our two methods starting with random iterates is hard to predict. This suggests that the problem may be difficult to solve if we have no idea of what does the optimal solution look like.



(c) The gradient algorithm converges but the conjugate gradients does not.



(d) The conjugate gradients algorithm converges but the gradient does not.

Figure 3.1: Runs of `btd_grad` and `btd_cg` with 4 different starting iterates.

Influence of the distance between the starting iterate and the optimal solution

In the first part of this subsection, we studied the ability of our methods to recover a given BTD starting from 4 different starting points. In this part, we use our knowledge of the optimal solution to study the effect of the distance between the starting iterate and the optimal solution on the behavior of the methods. To that end, we proceed as follows.

1. We construct a BTD as explained at the begining of the subsection. We let x denote $(\mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R)$.

2. We pick up $\dot{x} \in \prod_{n=1}^{3} \left(\mathbb{R}^{I_n \times R_n}\right)^R$ at random according to the standard normal distribution.

3. We normalize the norm of $\dot{x}$ to 1: $\dot{x} := \dot{x} / \|\dot{x}\|$.

4. We select a noise level $\sigma > 0$ and we define $x_0 := x + \sigma \dot{x}$.

5. We let $\bar{x}_0$ denote the projection of x_0 onto $\prod_{n=1}^{3} \mathrm{St}(R_n, I_n)^R$.

6. We use $\bar{x}_0$ as starting point for our algorithms.

We use the following parameters: $(I_1, I_2, I_3) := (6, 6, 6)$, $R \in \{2, 3, 4\}$, $(R_1, R_2, R_3) := (2, 3, 2)$ and $\sigma \in \{0.01, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4\}$. As usual, we stop the process as soon as the norm of the gradient becomes smaller than $5 \cdot 10^{-14}$ or a certain number of iterations is reached. Before presenting the results, we should make three remarks.

1. We control only the distance σ between x and x_0, not the distance between x and $\bar{x}_0$. In general, these two distances are not equal. However, there are often quite close to each other. For instance, in all our numerical tests, the distance between x and $\bar{x}_0$ is always between 0.76σ and 0.91σ.

2. By "distance between x and $\bar{x}_0$", we mean the usual distance in the space $\prod_{n=1}^{3} \left(\mathbb{R}^{I_n \times R_n}\right)^R$, not the Riemannian distance in $\prod_{n=1}^{3} \mathrm{St}(R_n, I_n)^R$ defined by [18, equation (3.30)].

3. The distance between any two points of $\prod_{n=1}^{3} \mathrm{St}(R_n, I_n)^R$ is bounded.

   Indeed, for any $\mathbf{U} \in \mathrm{St}(p, n)$,

   $$\|\mathbf{U}\| = \sqrt{\mathrm{tr}(\mathbf{U}^{\mathrm{T}}\mathbf{U})} = \sqrt{\mathrm{tr}(\mathbf{I}_p)} = \sqrt{p}.$$

   Therefore, for any $\mathbf{X}, \mathbf{Y} \in \mathrm{St}(p, n)$, Minkowski's inequality yields

   $$\|\mathbf{X} - \mathbf{Y}\| \leq \|\mathbf{X}\| + \|\mathbf{Y}\| = 2\sqrt{p}.$$

   Let

   $$x := (\mathbf{U}_1, \ldots, \mathbf{U}_R, \mathbf{V}_1, \ldots, \mathbf{V}_R, \mathbf{W}_1, \ldots, \mathbf{W}_R),$$
   $$\bar{x} := (\bar{\mathbf{U}}_1, \ldots, \bar{\mathbf{U}}_R, \bar{\mathbf{V}}_1, \ldots, \bar{\mathbf{V}}_R, \bar{\mathbf{W}}_1, \ldots, \bar{\mathbf{W}}_R) \in \prod_{n=1}^{3} \mathrm{St}(R_n, I_n)^R.$$

   Then,

   $$\begin{aligned}
   \|x - \bar{x}\|^2 &= \left\|(\mathbf{U}_1 - \bar{\mathbf{U}}_1, \ldots, \mathbf{U}_R - \bar{\mathbf{U}}_R, \mathbf{V}_1 - \bar{\mathbf{V}}_1, \ldots, \mathbf{V}_R - \bar{\mathbf{V}}_R, \mathbf{W}_1 - \bar{\mathbf{W}}_1, \ldots, \mathbf{W}_R - \bar{\mathbf{W}}_R)\right\|^2 \\
   &= \sum_{r=1}^{R} \left(\left\|\mathbf{U}_r - \bar{\mathbf{U}}_r\right\|^2 + \left\|\mathbf{V}_r - \bar{\mathbf{V}}_r\right\|^2 + \left\|\mathbf{W}_r - \bar{\mathbf{W}}_r\right\|^2 \right) \\
   &\leq \sum_{r=1}^{R} (4R_1 + 4R_2 + 4R_3) \\
   &= 4R(R_1 + R_2 + R_3).
   \end{aligned}$$

   In conclusion,

   $$\|x - \bar{x}\| \leq 2\sqrt{R}\sqrt{R_1 + R_2 + R_3}.$$

   It is clear that this bound can be reached so it is called the diameter of $\prod_{n=1}^{3} \mathrm{St}(R_n, I_n)^R$.

   For our parameters, the diameter is between 7.4 and 10.6, which shows that our values for σ are reasonable.

For each value of σ, 20 different starting iterates were used with both methods. A success means that the norm of the gradient falls below $5 \cdot 10^{-14}$. The number of successes for the 3 values of R are given in Table 3.1. We imposed an upper limit on the number of iterations varying from $3 \cdot 10^3$ for $R = 2$ to $3 \cdot 10^4$ for $R = 4$.

We see that the number of successes decreases as σ and R increase. In particular, for $R \in \{3, 4\}$ both methods failed each time they were run starting with a randomly selected point. Besides that, both methods seem to be reliable when σ is not too large.

σ	$R := 2$		$R := 3$		$R := 4$	
0.01	20	20	20	20	20	20
0.5	20	20	20	20	20	20
0.1	20	20	20	20	20	20
1.5	20	20	20	20	20	20
2	20	20	20	20	20	20
2.5	20	20	20	19	20	18
3	20	20	20	19	20	14
3.5	20	20	18	15	17	14
4	18	18	17	17	14	11
random	13	14	0	0	0	0

Table 3.1: Number of successes over 20 attempts as a function of σ for 3 values of R. Left and right subcolumns correspond respectively to `btd_grad` and `btd_cg`.

3.1.2 Influence of the number of terms in the BTD

Another important observation is that the more terms the BTD contains, the more difficult is the problem. We already made this observation in Table 3.1. In this subsection, we investigate this fact in more detail.

Our first test consists of recovering a known BTD of R terms starting from 20 randomly selected points for $R \in \{2, 3, 4, 5\}$. We are going to focus on the number of successes of each method: a success means that the method is able to bring the norm of the gradient below a certain threshold using a given number of iterations.

The results for $(I_1, I_2, I_3) := (6, 6, 6)$, $(R_1, R_2, R_3) := (2, 3, 2)$ and $R \in \{2, 3, 4\}$ are presented in Table 3.2. The tolerance on the gradient norm is $5 \cdot 10^{-14}$. The maximum number of iterations is $4 \cdot 10^4$ for $R \in \{2, 3\}$ and $5 \cdot 10^4$ for $R = 4$. We observe that both methods work well for $R = 2$. For $R = 3$, they fail in more than half the cases. For $R = 4$, both methods fail most of the time. Moreover, we see that the gradient algorithm needs almost $5 \cdot 10^4$ iterations to converge.

R	2		3		4	
number of successes	19	20	7	7	1	2
minimum number of iterations	806	438	1063	436	47 167	7454
mean number of iterations	1668	1083	4165	959	47 167	7859
maximum number of iterations	3944	6997	12 105	1824	47 167	8284
minimum running time	2.02	1.57	3.82	2.15	348.00	78.86
mean running time	3.57	5.31	26.05	6.70	348.00	80.74
maximum running time	15.11	42.49	86.58	14.64	348.00	82.61

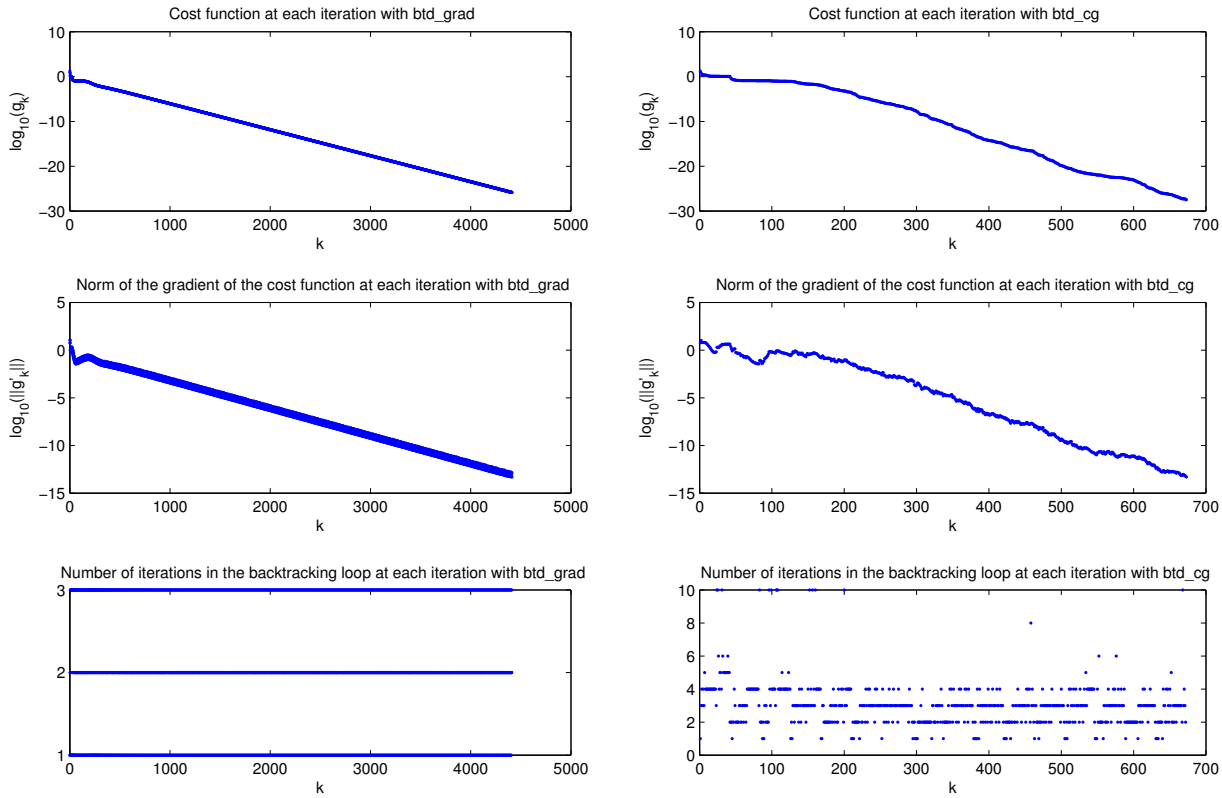
Table 3.2: Runs of `btd_grad` (left subcolumns) and `btd_cg` (right subcolumns) with 20 randomly selected starting iterates for 3 values of R. The extrema and means are computed only over the successful runs. Running times are given in seconds.

We also ran the methods for $R := 5$ with $(I_1, I_2, I_3) := (10, 10, 10)$ and $(R_1, R_2, R_3) := (2, 3, 2)$. We did not include the results in Table 3.2 because they are too simple: none of the two methods has succeeded over 20 attempts. Although the upper limit on the number of iterations was 10^5 for `btd_grad` and $5 \cdot 10^4$ for `btd_cg`, the norm of the gradient never fell below 10^{-7}. The only performance that can be highlighted is that `btd_grad` brought the norm of the gradient below 10^{-16} 5 times while `btd_cg` did it 15 times.

The conclusion is clear: the success rate of both methods decreases very quickly with R. In fact, the methods can be said to be reliable only for BTDs of two terms. Let us end this subsection with some representative plots. Figure 3.2 describes the runs of both methods on three BTDs with $(I_1, I_2, I_3) := (5, 5, 5)$, $(R_1, R_2, R_3) := (2, 2, 2)$ and $R \in \{2, 3, 4\}$. The starting iterates are generated randomly. The tolerance on the norm of the gradient is set to $5 \cdot 10^{-14}$.

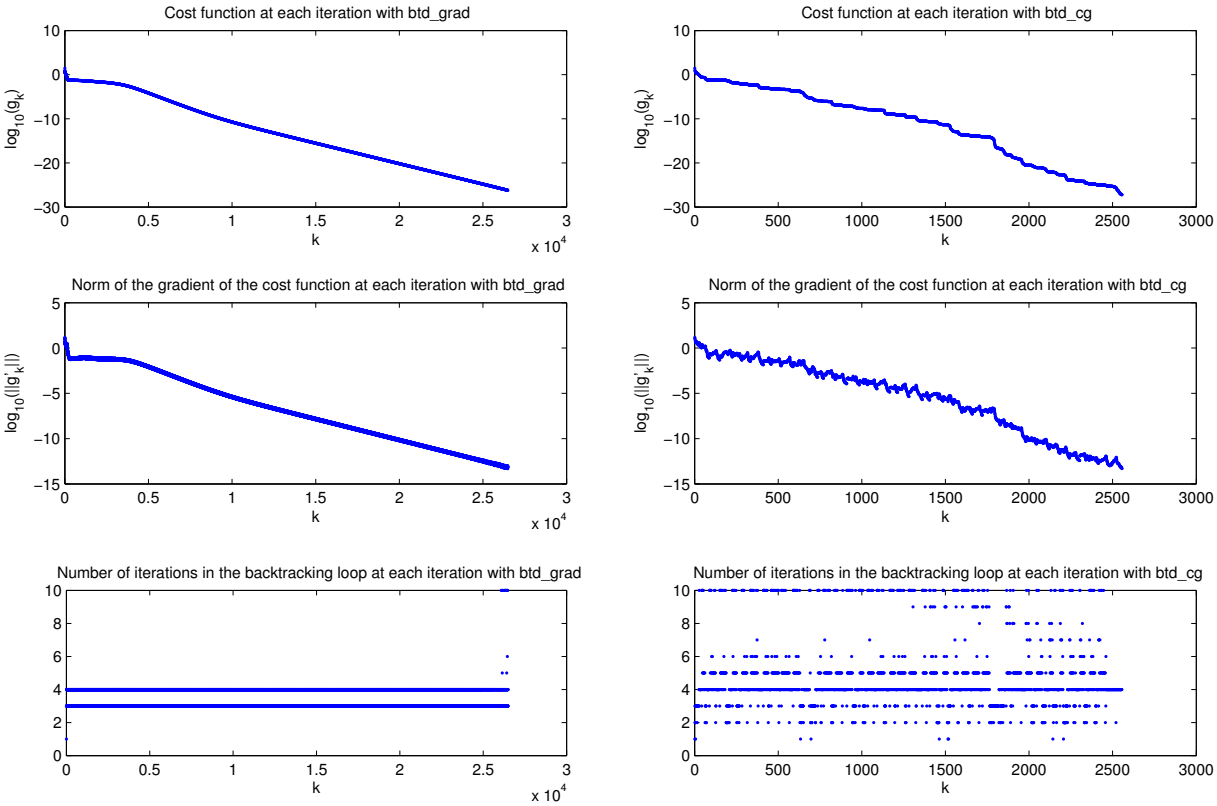
As it is often the case (recall Table 3.2), the gradient algorithm needs more iterations to converge than the conjugate gradients algorithm. However, the latter often makes more iterations in the backtracking loop than the first. Although it does not appear completely clearly in Table 3.2, the number of iterations needed to converge increases with the number of terms in the BTD; this can be observed by comparing subfigures (a) and (b).

Finally, let us comment the failure observed in Figure 3.2c. The conjugate gradients algorithm seems to completely stagnate from iteration number $2 \cdot 10^4$; especially many iterations are performed in each backtracking loop. The gradient method does not seem to stagnate but the convergence is really slow.

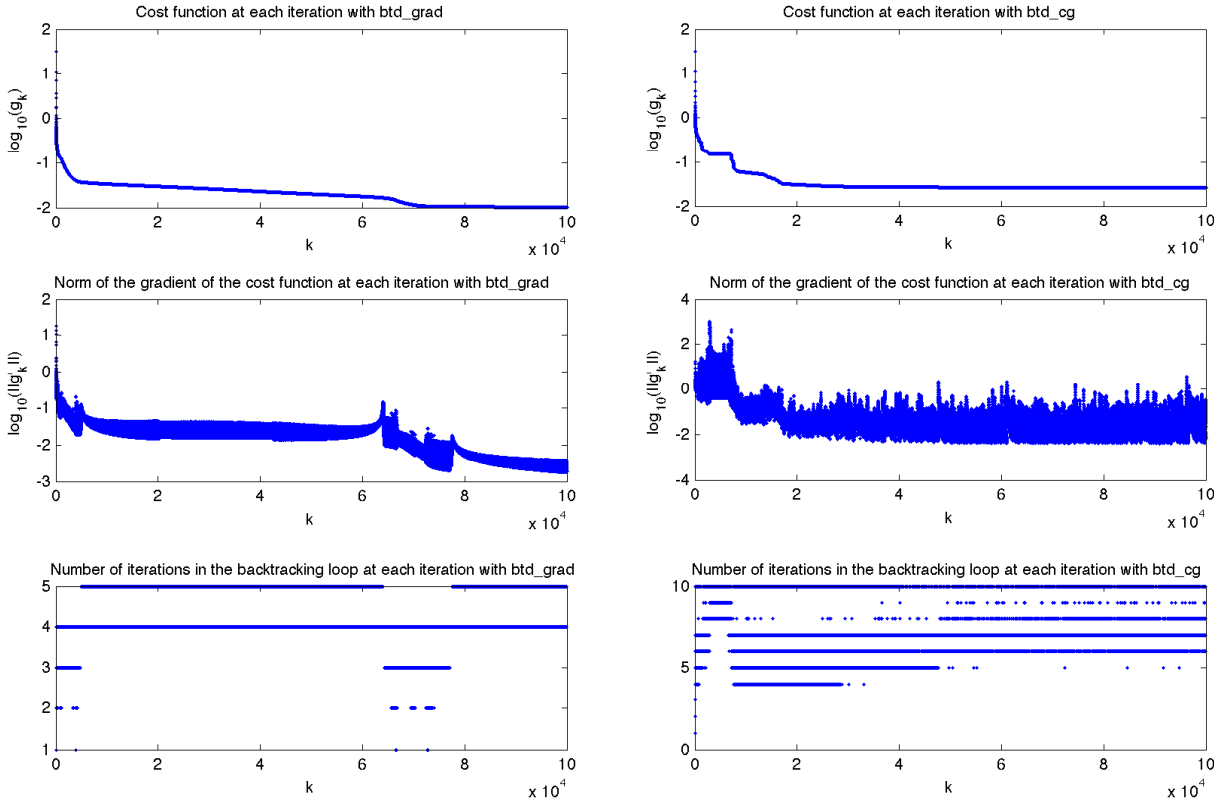


(a) $R := 2$. Both algorithms converge.

Figure 3.2: Runs of `btd_grad` and `btd_cg` for BTDs with 2, 3 and 4 terms.



(b) $R := 3$. Both algorithms converge.



(c) $R := 4$. None of the two algorithms converges.

Figure 3.2: Runs of `btd_grad` and `btd_cg` for BTDs with 2, 3 and 4 terms.

3.1.3 Influence of the size of the tensor

The numerical tests suggest that the size of the tensor to be decomposed does not influence so much the behavior of the methods. The number of iterations needed to converge does not vary in a perceptible way with the size, even if the computational cost of one iteration obviously increases with it.

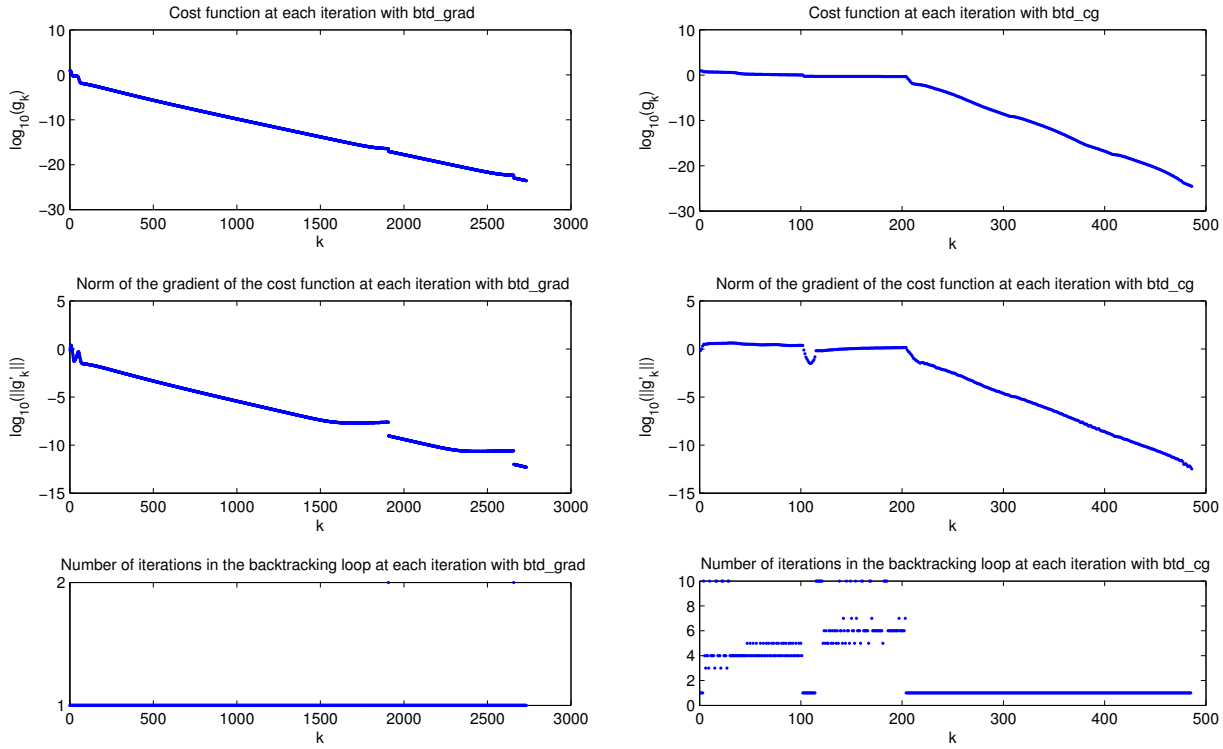
Figure 3.3 describes the runs of our two methods on three BTDs of different sizes with $(R_1, R_2, R_3) := (2, 2, 2)$ and $R := 2$. The tolerance on the norm of the gradient is set to $5 \cdot 10^{-13}$; this value was chosen because both methods do not seem to be able to bring the norm of the gradient below the usual threshold of $5 \cdot 10^{-14}$ for the considered sizes.

Looking at the plots, the size does not seem to influence either the number of iterations needed to converge, or the number of iterations in the backtracking loop. In addition, we should highlight the strange behavior of the gradient method in Figure 3.3c: the norm of the gradient increases during about a thousand of iterations before falling down to the threshold.

Finally, let us show that the computational cost of one iteration increases with the size. The running times per iteration for each of the three runs of Figure 3.3 are given in Table 3.3. We observe two facts. First, as announced, the running time of one iteration increases with the size. Secondly, the running time of one iteration is greater for `btd_cg` than for `btd_grad`, which is normal since `btd_cg` performs more iterations in the backtracking loop.

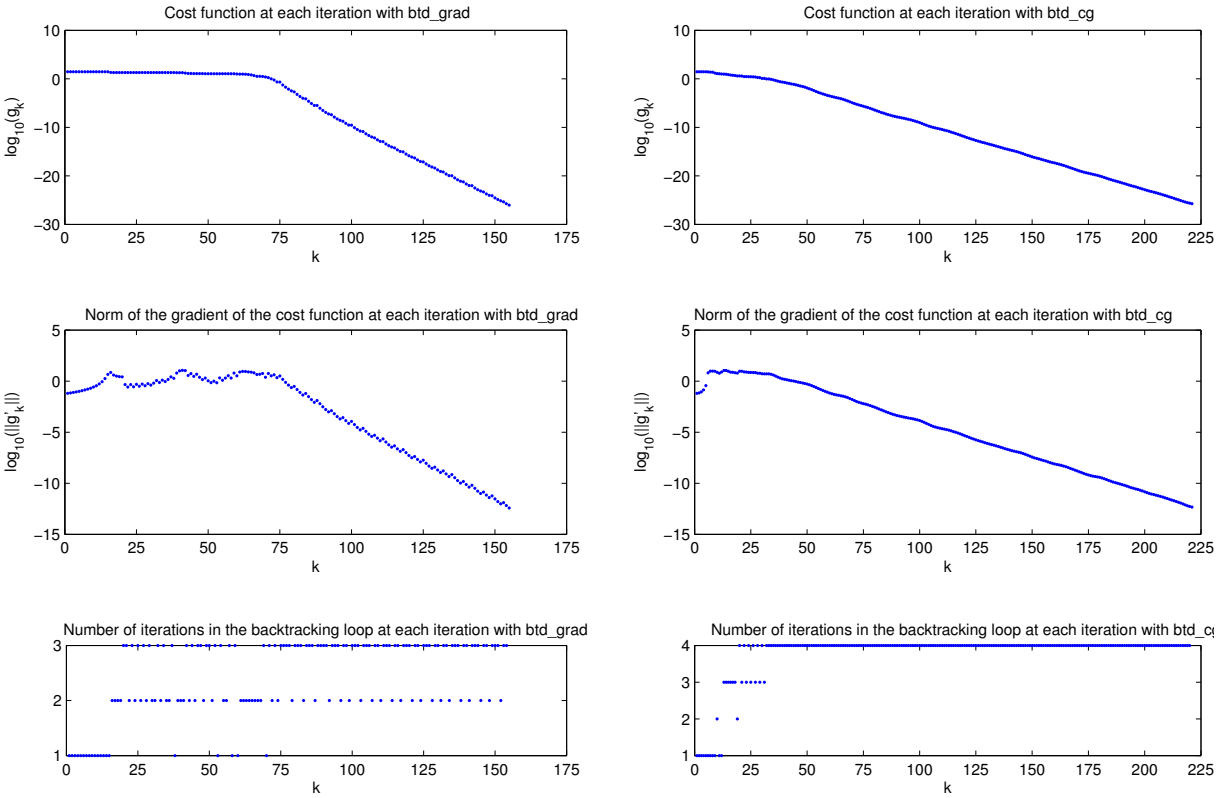
	$(10, 10, 10)$	$(20, 20, 20)$	$(40, 40, 40)$
`btd_grad`	$3.74 \cdot 10^{-3}$	$8.64 \cdot 10^{-3}$	$2.20 \cdot 10^{-2}$
`btd_cg`	$7.15 \cdot 10^{-3}$	$1.28 \cdot 10^{-2}$	$9.04 \cdot 10^{-2}$

Table 3.3: Running time (in seconds) per iteration for 3 different sizes.

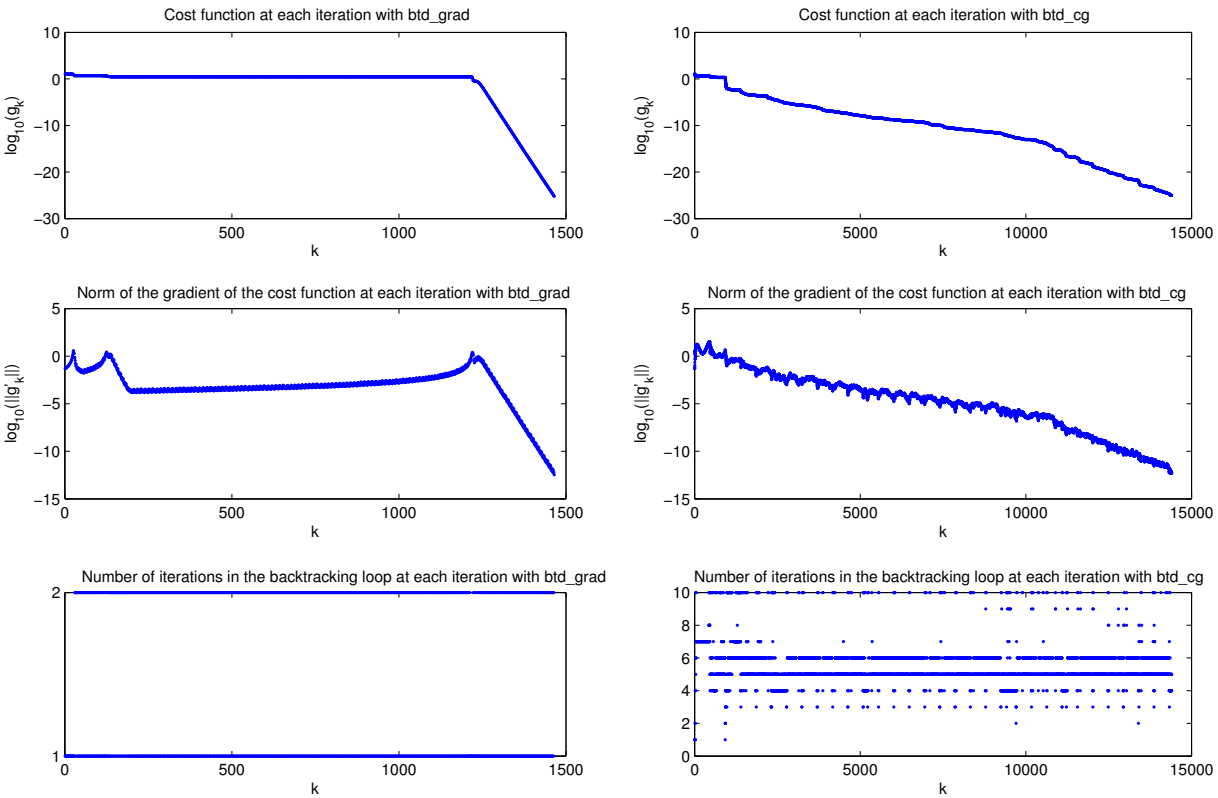


(a) $(I_1, I_2, I_3) := (10, 10, 10)$

Figure 3.3: Runs of `btd_grad` and `btd_cg` for BTDs of different sizes.



(b) $(I_1, I_2, I_3) := (20, 20, 20)$



(c) $(I_1, I_2, I_3) := (40, 40, 40)$

Figure 3.3: Runs of `btd_grad` and `btd_cg` for BTDs of different sizes.

3.1.4 Influence of the multilinear rank of the terms in the BTD

The influence of the multilinear rank of the terms in the BTD is similar to the one of the size. Consequently, this subsection follows closely the same structure as the previous one.

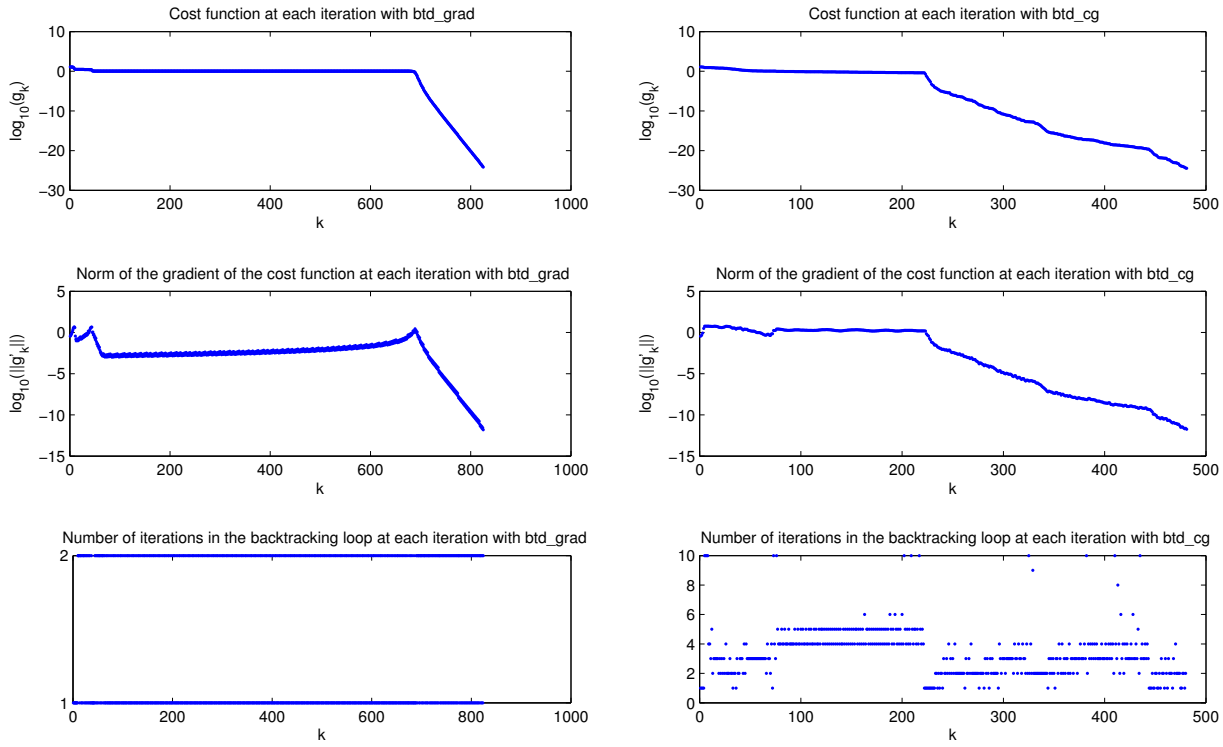
Figure 3.4 describes the runs of our two methods on three BTDs of different multilinear ranks with $(I_1, I_2, I_3) := (20, 20, 20)$ and $R := 2$. The tolerance on the norm of the gradient is set to $2 \cdot 10^{-12}$ because both methods do not seem to be able to bring the norm of the gradient below the usual threshold of $5 \cdot 10^{-14}$ for the considered multilinear ranks.

Looking at the plots, the multilinear rank does not seem to influence either the number of iterations needed to converge, or the number of iterations in the backtracking loop. In addition, as we did for Figure 3.3c, let us point out the strange behavior of the gradient method in Figure 3.4a: the norm of the gradient decreases only over the last hundred iterations.

Finally, let us show that the computational cost of one iteration increases with the multilinear rank. The running times per iteration for each of the three runs of Figure 3.4 are given in Table 3.4. We can make exactly the same observations as for Table 3.3. However, we should note that the running time grows more quickly with the multilinear rank than with the size. This is not surprising since the multilinear rank controls directly the size of the linear system (2.2) that has to be solved to perform the variable projection.

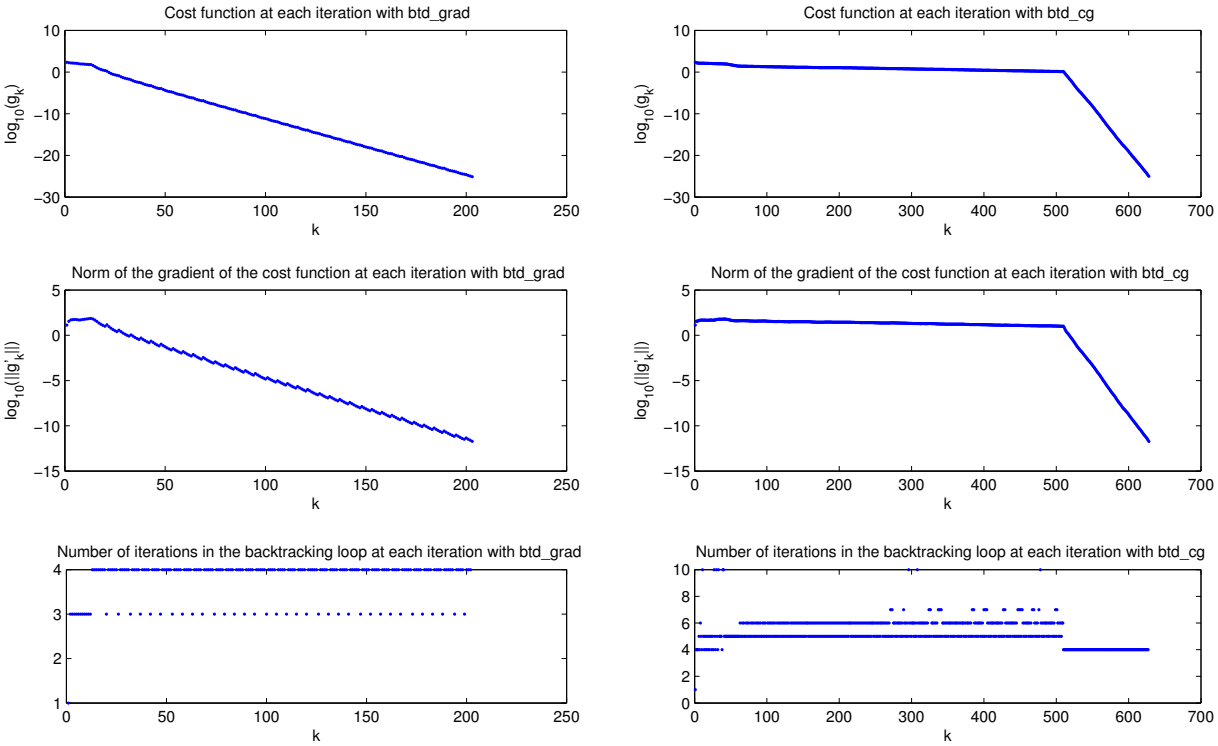
	$(2, 2, 2)$	$(5, 5, 5)$	$(10, 10, 10)$
`btd_grad`	$3.56 \cdot 10^{-3}$	$1.08 \cdot 10^{-1}$	2.18
`btd_cg`	$7.73 \cdot 10^{-3}$	$1.51 \cdot 10^{-1}$	3.60

Table 3.4: Running time (in seconds) per iteration for 3 different multilinear ranks.

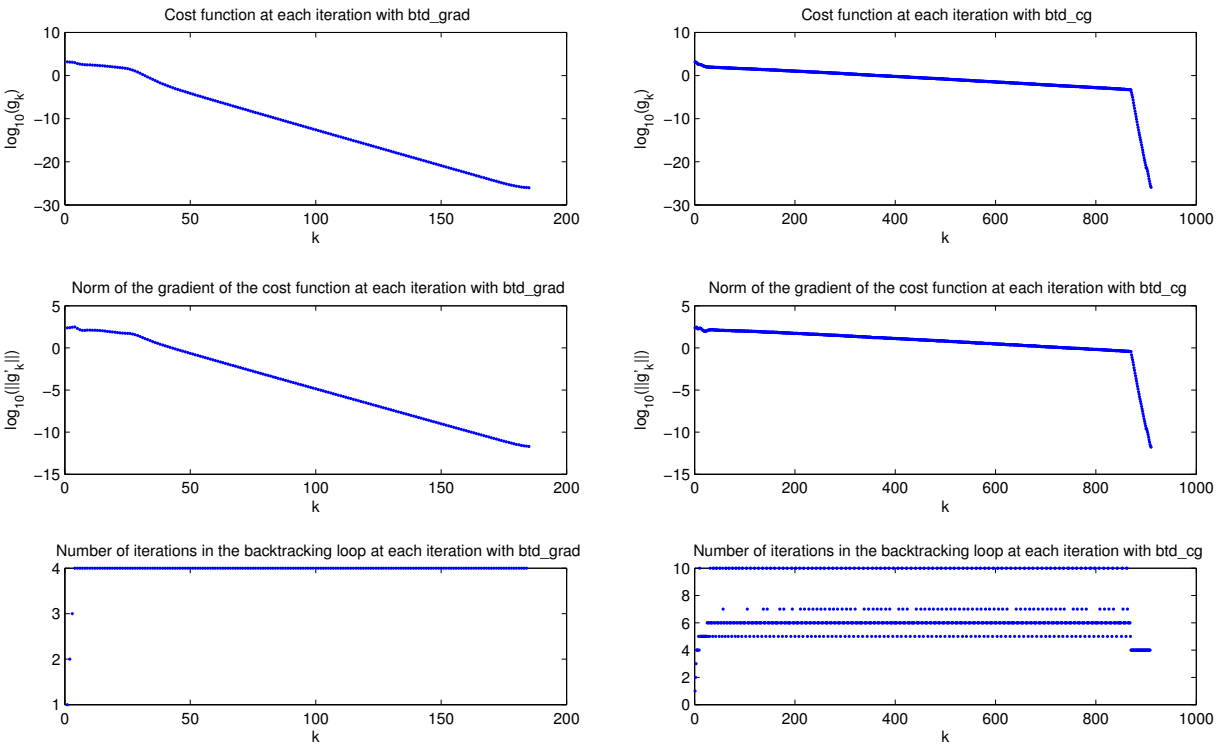


(a) $(R_1, R_2, R_3) := (2, 2, 2)$

Figure 3.4: Runs of `btd_grad` and `btd_cg` for BTDs of different sizes.



(b) $(R_1, R_2, R_3) := (5, 5, 5)$



(c) $(R_1, R_2, R_3) := (10, 10, 10)$

Figure 3.4: Runs of `btd_grad` and `btd_cg` for BTDs of different multilinear ranks.

3.1.5 Why convergence is not necessarily observed

In the preceding tests, we frequently observed that the algorithms stopped because the limit of iterations was reached, while the norm of the gradient was still quite large (see, e.g., Figure 3.1a or Figure 3.2c). But the theory ensures that the gradient algorithm must converge, i.e., it must hold that

$$\lim_{k\to\infty} \|(\operatorname{grad} \bar{g}_{\mathcal{A}})(x_k)\| = 0.$$

In this subsection, we try to understand why the convergence of the norm of the gradient to zero in not necessarily observed in practice by looking at a simple example with only two variables.

Let us define the function

$$f_{a,b} : \mathbb{R}^2 \to \mathbb{R} : (x, y) \mapsto f_{a,b}(x, y) := g_1(ax) + g_2(by)$$

with $a, b \in \mathbb{R}$,

$$g_1(x) := \frac{x^2}{1+x^2} = 1 - \frac{1}{1+x^2} \qquad \text{and} \qquad g_2(x) := x^2.$$

This function is infinitely differentiable and we compute

$$f'_{a,b}(x, y) = \begin{bmatrix} a g'_1(ax) \\ 2b^2 y \end{bmatrix} \qquad \text{and} \qquad f''_{a,b}(x, y) = \begin{bmatrix} a^2 g''_1(ax) & 0 \\ 0 & 2b^2 \end{bmatrix}$$

with

$$g'_1(x) := \frac{2x}{(1+x^2)^2} \qquad \text{and} \qquad g''_1(x) := 2\frac{1-3x^2}{(1+x^2)^3}.$$

Using [20, Lemma 1.2.2], it is easy to see that $f'_{a,b}$ is Lipschitz continuous with constant $L_{a,b} := 2\max(a^2, b^2)$. However, it should be noted that $f_{a,b}$ is not a convex function.

Let us try to minimize $f_{a,b}$ using the gradient algorithm. We wrote the following Matlab functions which implement the gradient algorithm:

- `grad_bktg` uses a backtracking strategy to select a step length at each iteration,
- `grad_lip` uses a constant step length equal to $1/L_{a,b}$.

First, let us check that the sequence generated by the gradient algorithm will converge to the unique minimizer of $f_{a,b}$, namely $(0, 0)$. For simplicity, we assume that the initial iterate (x_0, y_0) has positive components. In our problem, the two components of the iterates evolve independently according to the following update formulas:

$$x_{k+1} = \left(1 - \frac{2a^2 t_k}{(1+(ax_k)^2)^2}\right) x_k, \qquad y_{k+1} = (1 - 2b^2 t) y_k.$$

If $0 < t_k \leq 1/L_{a,b}$, then the sequences $(x_k)_{k\in\mathbb{N}}$ and $(y_k)_{k\in\mathbb{N}}$ are both decreasing and positive and so converge. Taking the limit $k \to \infty$ in the update formulas yields

$$\lim_{k\to\infty} x_k = \lim_{k\to\infty} y_k = 0,$$

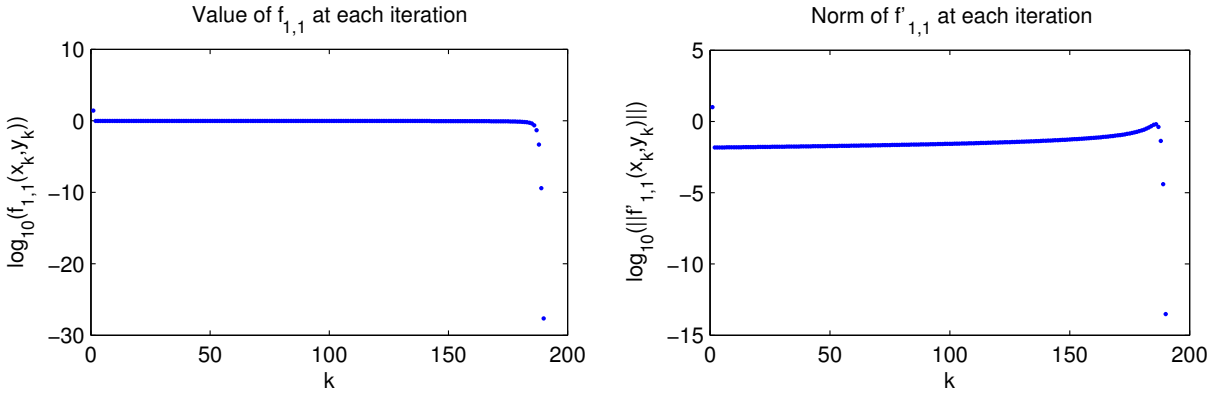
as announced.

Now, let us show that the convergence may be very slow so that it is not always observable in practice. We always take $(x_0, y_0) := (5, 5)$ as starting iterate and we stop the process as soon as the norm of $f'_{a,b}$ becomes smaller than $5 \cdot 10^{-14}$.

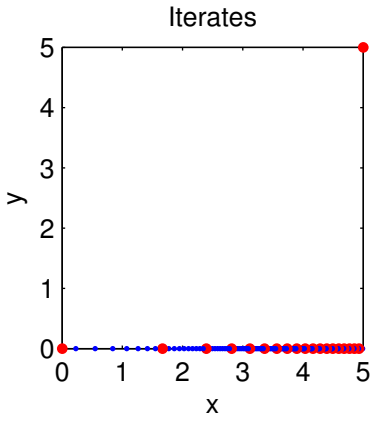
The run of both `grad_bktg` and `grad_lip` on $f_{1,1}$ is described in Figure 3.5. The first step length tried in the backtracking loop is $1/2$, which is equal to $1/L_{1,1}$, and it turns out that the

sufficient decrease condition is met with that step length. It follows that the sequences of iterates generated by the two methods are the same.

We observe in Figure 3.5a that during most of the iterations, the objective stagnates near 1 and the norm of the gradient slowly increases. This is also illustrated in Figure 3.5b: the distance between two consecutive iterates increases with the iteration number (ignoring the initial iterate). More precisely, we see that about 100 iterations are needed to travel from $(x_1, y_1) = (\frac{3375}{676}, 0) \approx (4.99, 0)$ to $(4, 0)$. We also see that more than 50 iterations are needed to travel from $(4, 0)$ to $(3, 0)$. In fact, Figure 3.5 shows that the convergence of the algorithm occurs during the 5 last iterations.



(a) The backtracking loop performs only one iteration.



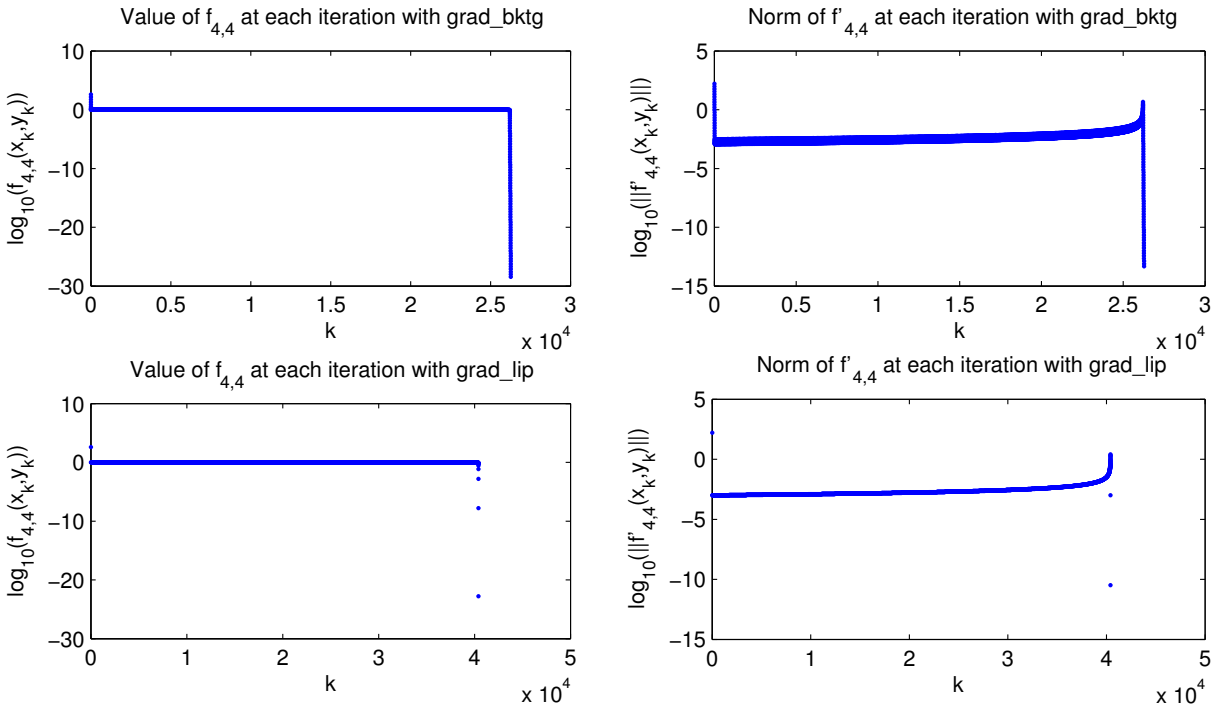
(b) Each of the 190 iterates is represented by a point. The points (x_k, y_k) such that k is a multiple of 10 are drawn in red.

Figure 3.5: Runs of both `grad_bktg` and `grad_lip` on $f_{1,1}$.

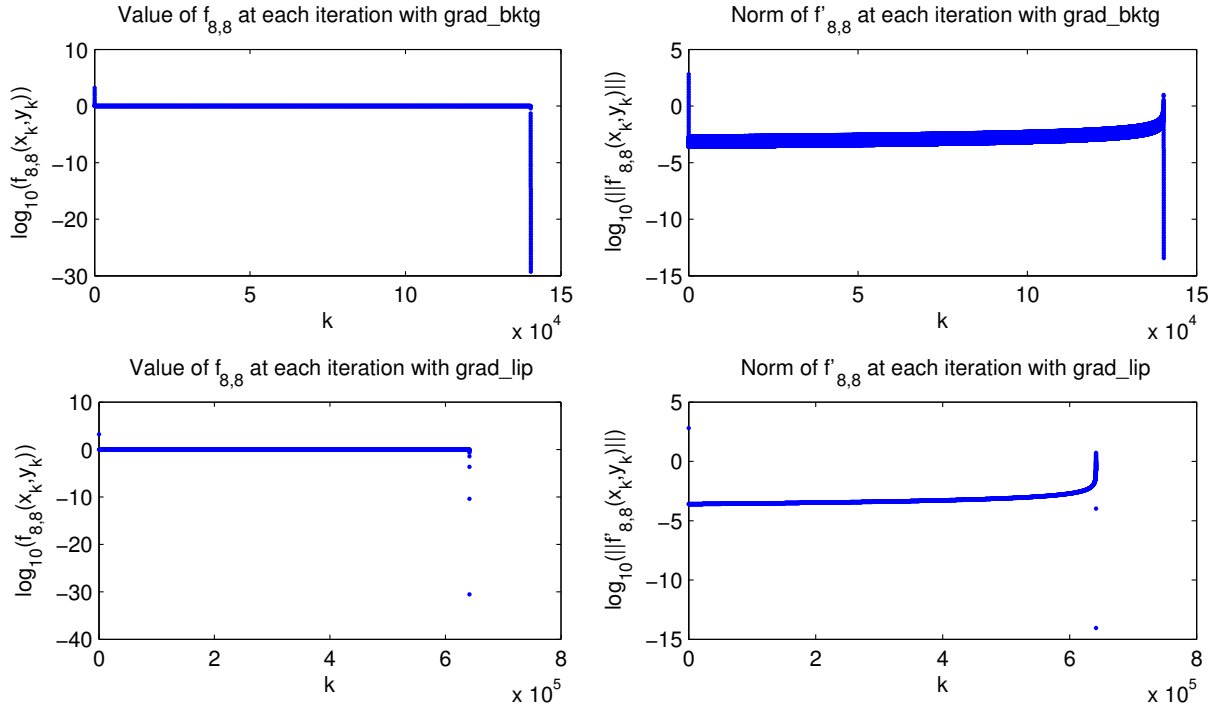
One can argue that the stagnation observed in the BTD computation is far longer than the one observed in Figure 3.5. This is true but we are going to see that the stagnation of the gradient algorithm in the minimization of $f_{a,b}$ can be far longer if we increase a and b.

The runs of `grad_bktg` and `grad_lip` on $f_{a,b}$ for 2 different couples (a, b) are described in Figure 3.6. Qualitatively, the behavior of the two methods is the same as in Figure 3.5a. However, we observe that the stagnation of `grad_bktg` lasts about $25 \cdot 10^3$ iterations for $a = b = 4$ and more than 10^5 iterations for $a = b = 8$. We can also see that the backtracking strategy is often more performant than the constant step length strategy.

Let us now try to understand what is going on when applying the gradient algorithm to minimize $f_{a,b}$. The graph of $f_{1,1}$ is plotted on the right of Figure 3.7. We could say that this graph looks like a valley: the function increases rapidly with $|y|$ but is almost flat when $|y|$ is



(a) The backtracking loop performs 1 or 4 iterations.



(b) The backtracking loop performs 2, 3 or 4 iterations.

Figure 3.6: Runs of `grad_bktg` and `grad_lip` on $f_{a,b}$ for 2 different couples (a, b).

small, except if $|x|$ is also small, as can be seen on the left plot. Moreover, the more a and b increase the more those characteristics are significant.

Let us now try to explain the path followed by the iterates (Figure 3.5b) using the "valley picture". The first step from $(5, 5)$ to $(4.99, 0)$ is quite big because the norm of the gradient is quite big near $(5, 5)$. The point $(4.99, 0)$ is in the flat part of the valley, i.e., the norm of $f'_{1,1}$ is

small near this point. Travelling through the valley to $(0, 0)$ takes a lot of iterations because the norm of the gradient is quite small which implies small distances between consecutive iterates.

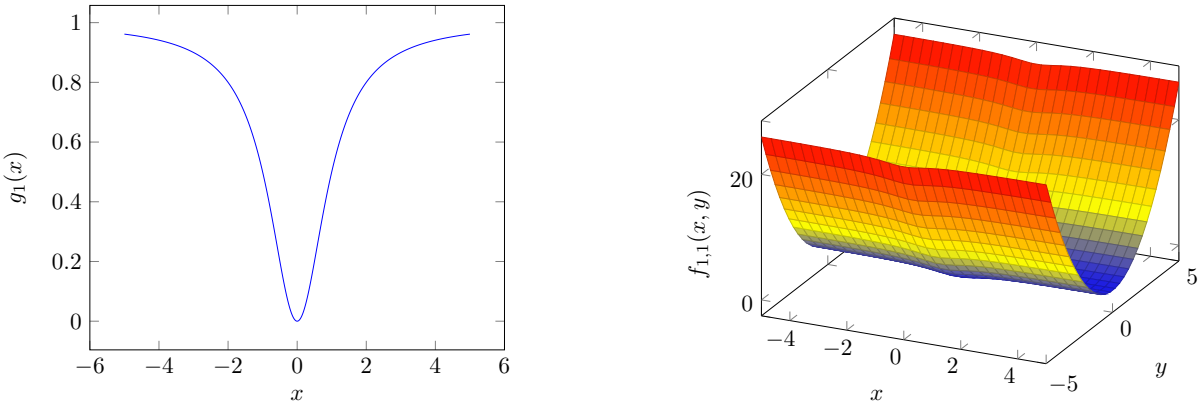


Figure 3.7: Respective graphs of g_1 (on the left) and $f_{1,1}$ (on the right).

The conclusion is that the gradient algorithm can be very slow to minimize a function that is "almost flat" with respect to one or more of its variables. In our simple example, the function is almost flat with respect to the first variable. It follows that the sequence $((x_k, y_k))_{k\in\mathbb{N}}$ generated by the gradient algorithm does converge to a minimum but the convergence of the sequence $(x_k)_{k\in\mathbb{N}}$ takes more than 10^5 iterations which is quite slow. In a more complex problem, such as the BTD computation, it is not always possible to wait for hundreds of thousands of iterations.

3.1.6 Gradient algorithm with projective retraction*

Up to now, we used only the qf retraction (2.5) in our gradient method. One could argue that the projective retraction introduced in Proposition 2.5.1 should work better because it is a true projection, i.e., it minimizes the distance to the feasible set. In fact, this is not necessarily the case in practice and when there is a difference, it is often quite minor.

Figure 3.8 describes the run of the gradient method with each of the two retractions to recover a BTD composed of 3 terms of size $(6, 6, 6)$ and of multilinear rank $(2, 2, 2)$ starting from a randomly selected point. The process was stopped as soon as the norm of the gradient became smaller than 10^{-12}. Qualitatively, we see that the behavior of the method is the same with both retractions.

The number of iterations and the running time with each of the two retractions are given in Table 3.5. Although the run using the projective retraction needs a bit less iterations than the one using the qf retraction, it is more than a minute slower, which is not surprising since its computational cost is higher.

	number of iterations	running time (in seconds)
qf retraction	137 808	1 655.0
projective retraction	137 780	1 741.5

Table 3.5: Runs of the gradient algorithm with the qf retraction (2.5) and with the projective retraction.

For smaller BTDs, i.e., with smaller sizes or less terms, the number of iterations and the running time are the same or the difference is negligible. In conclusion, the projective retraction does not seem to work better than the qf retraction. Furthermore, the qf retraction is cheaper.

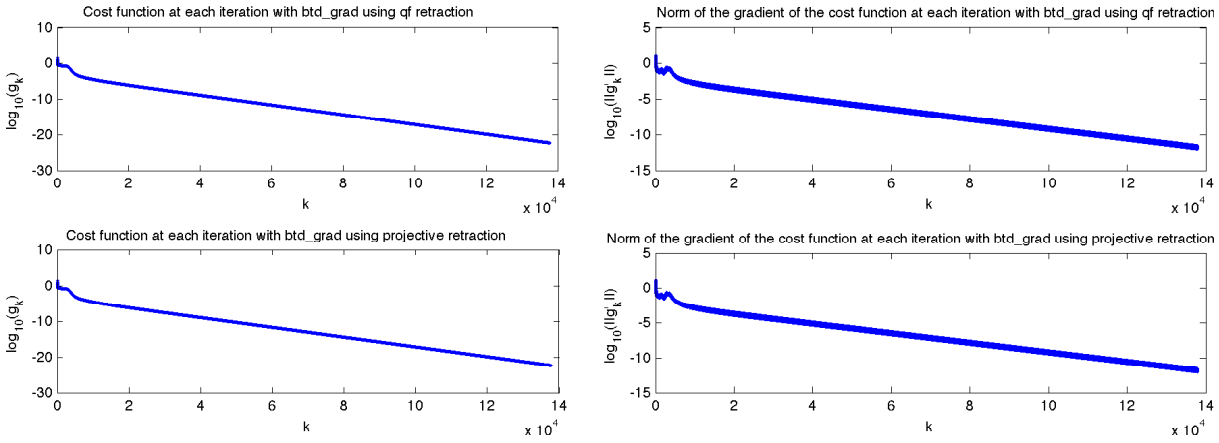


Figure 3.8: Runs of the gradient algorithm with the qf retraction (2.5) and with the projective retraction.

3.1.7 Gradient algorithm without variable projection*

Up to now, we tried to solve the problem (2.1) using variable projection. We saw that, since this variable projection can be performed only numerically, we are not able to compute the Hessian of the cost function $f_{\mathcal{A}}$. As a result, we have to use only first-order methods to solve the problem. It is so legitimate to check that the gradient method is more performant with variable projection than without.

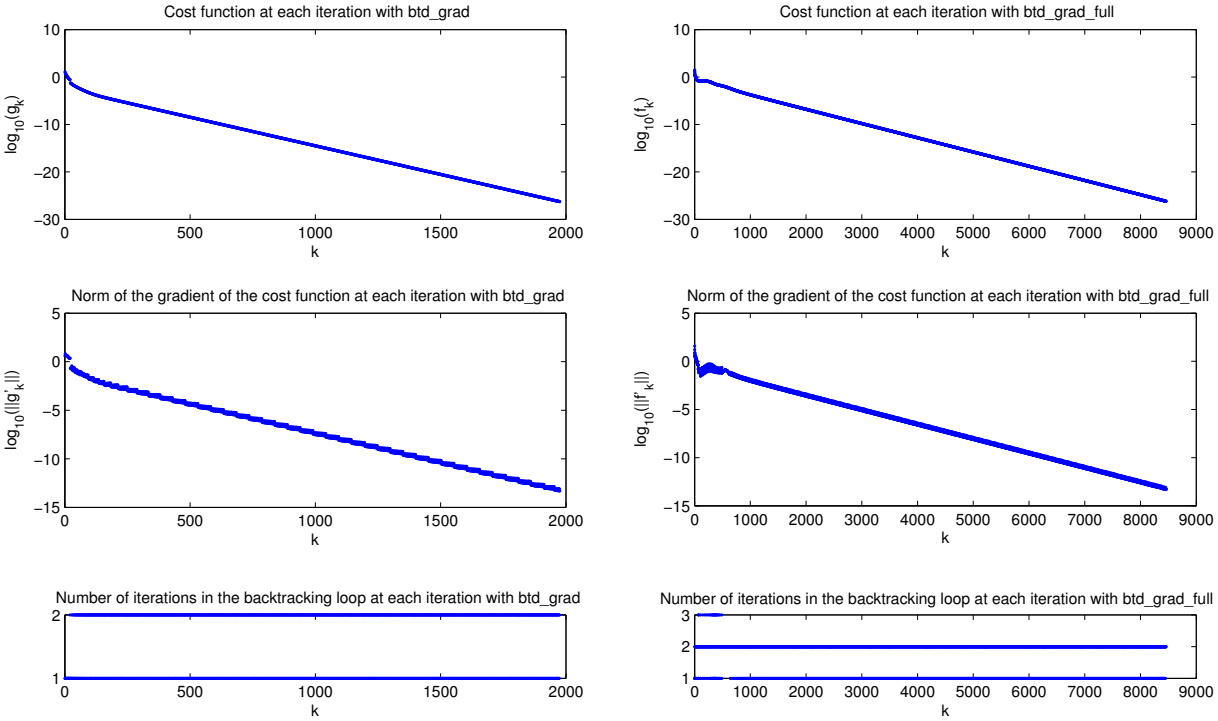
Figure 3.9 presents two runs of `btd_grad` and `btd_grad_full` on two different BTDs starting from a randomly selected point. The parameters are $(I_1, I_2, I_3) := (5, 5, 5,)$, $(R_1, R_2, R_3) := (2, 2, 2)$ and $R := 2$. It is clear that `btd_grad` outperforms `btd_grad_full` in both runs, especially the second.

3.1.8 Conclusion

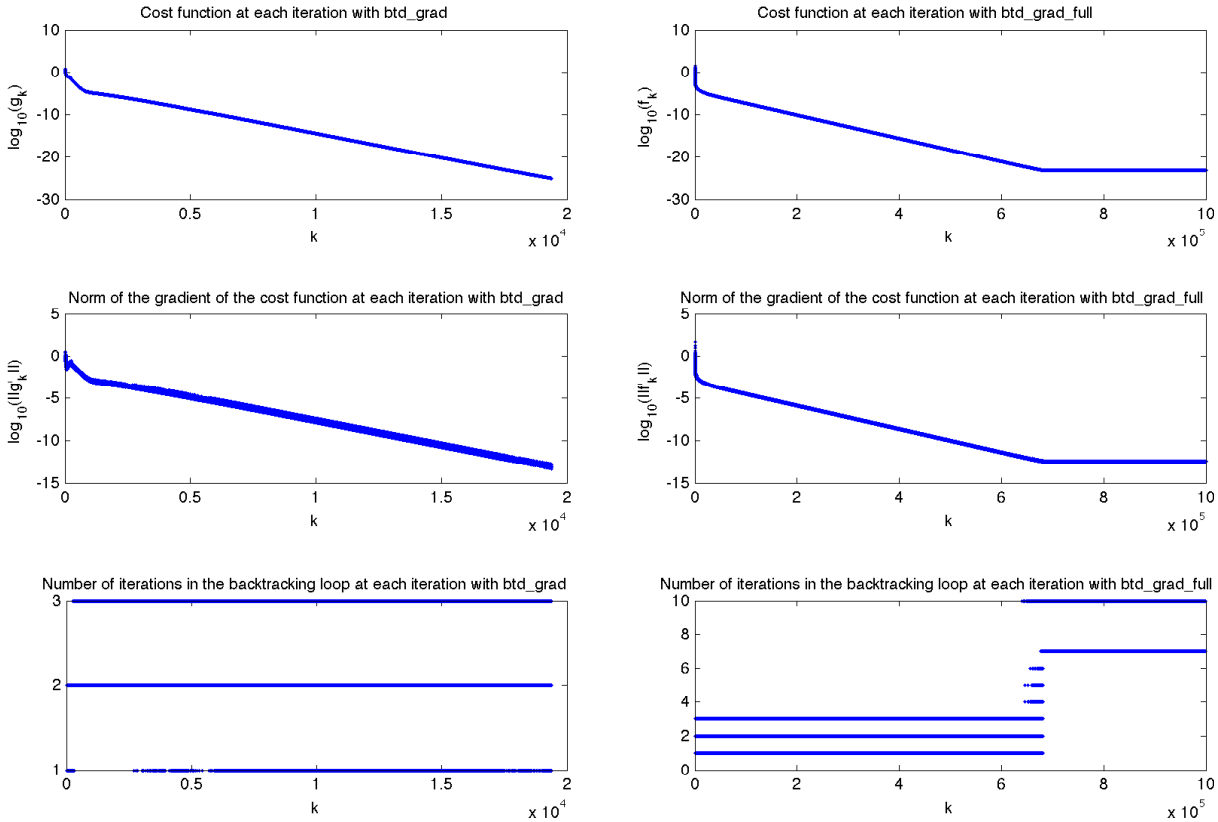
The important parameters in the BTD problem are the starting iterate and the number of terms. We saw in subsection 3.1.1 that for a given BTD, a starting iterate may lead to convergence while another one does not (or the convergence is too slow to be observed). Subsection 3.1.2 showed that our two methods are reliable only for BTDs containing two terms. They may work occasionally for BTDs containing more than two terms but the success ratios are small.

When both methods converge, the conjugate gradients algorithm is often more performant than the gradient algorithm even if exceptions exist.

Finally, we could have studied the complexity in detail. It is indeed possible to evaluate the computational cost of one iteration for both methods. Therefore, if we can estimate the global rate of convergence of our methods, we can get an upper bound on the number of iterations needed to obtain a given accuracy. As a result, we can compute an upper for the computational cost of the methods. This kind of analysis is possible in the framework of convex optimization; see, e.g., [20, chapter 2]. However, in the framework of optimization on matrix manifolds, one can only estimate the asymptotic rate of convergence; see [18, Theorem 4.5.6]. Furthermore, as we already noticed at the end of subsection 2.5.1, even this asymptotic result is not useful in our case since we do not know the Hessian of the cost function.



(a) First example: `btd_grad` and `btd_grad_full` bring respectively $\|\mathrm{grad}\,\bar{g}_{\mathcal{A}}\|$ and $\|\mathrm{grad}\,f_{\mathcal{A}}\|$ below $5 \cdot 10^{-14}$.



(b) Second example: `btd_grad` brings $\|\mathrm{grad}\,\bar{g}_{\mathcal{A}}\|$ below $5 \cdot 10^{-14}$ but `btd_grad_full` cannot bring $\|\mathrm{grad}\,f_{\mathcal{A}}\|$ below $3.78 \cdot 10^{-13}$ and so stagnates from about iteration number 680 000.

Figure 3.9: Comparison of `btd_grad` and `btd_grad_full`.

3.2 Comparison with the already available methods

3.2.1 State of the art

As far as we know, not so many algorithms have been designed to compute a general BTD. Tensorlab proposes the two following functions:[1]

- `btd_minf` uses L-BFGS with dogleg trust region (it is a quasi-Newton method),
- `btd_nls` uses nonlinear least squares by Gauss-Newton with dogleg trust region.

Another available algorithm is the alternating least squares algorithm introduced in [6]. This algorithm is not included in Tensorlab and does not work better than `btd_nls` in general. In the next subsection we compare our two implementations, `btd_grad` and `btd_cg`, with `btd_minf` and `btd_nls`.

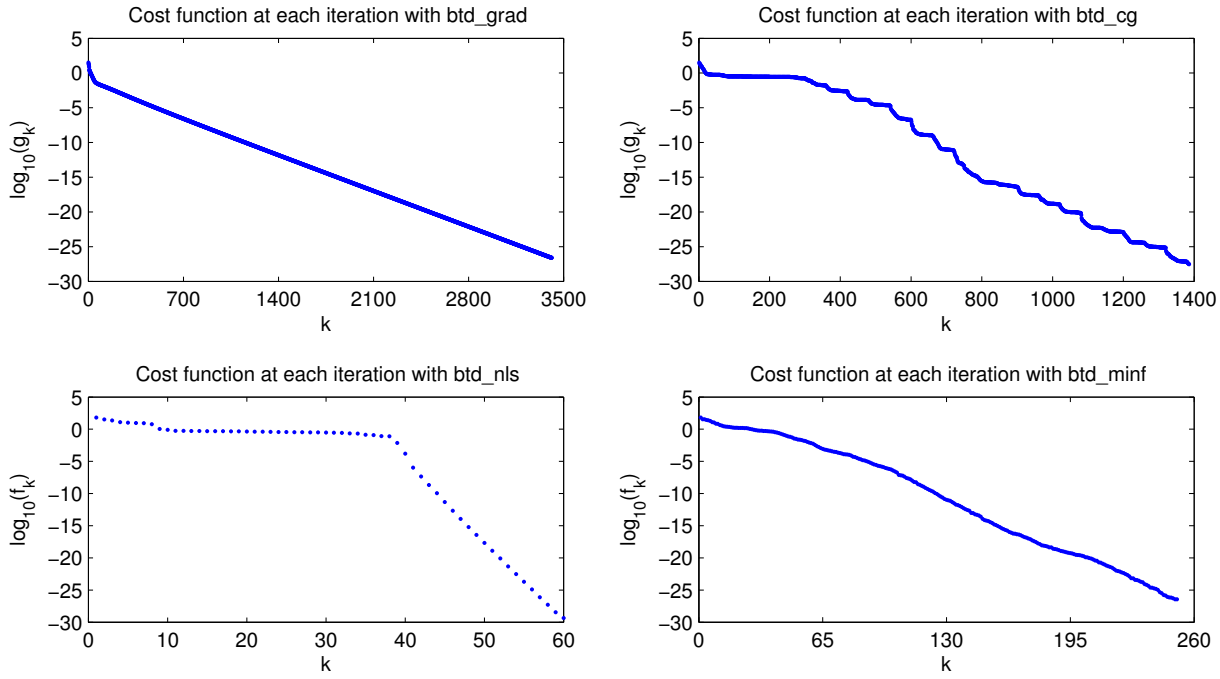
3.2.2 Comparison with `btd_minf` and `btd_nls`

Recovering a known BTD

We start with the test described at the beginning of section 3.1. The functions `btd_grad` and `btd_cg` stop as soon as the norm of the gradient falls below $5 \cdot 10^{-14}$. The functions `btd_nls` and `btd_minf` were used with the options `TolFun` and `TolX` respectively set to 10^{-30} and $3 \cdot 10^{-16}$.

The evolutions of the cost function with the four methods on a given BTD with $(I_1, I_2, I_3) := (6, 6, 6)$, $R := 2$, and $(R_1, R_2, R_3) := (2, 3, 2)$ are plotted in Figure 3.10 for four different starting iterates. Each subfigure corresponds to a different starting point.

In Figure 3.10a, all methods converge. As often in this case, the two methods from Tensorlab, especially `btd_nls`, need less iterations to converge than the two methods developed in this project. As measured by the principal angles, $\text{range}(\mathbf{U}_r) = \text{range}(\mathbf{U}_r^{\text{grad}}) = \text{range}(\mathbf{U}_r^{\text{cg}}) = \text{range}(\mathbf{U}_r^{\text{nls}}) = \text{range}(\mathbf{U}_r^{\text{minf}})$ and so on for $\mathbf{V}_r$ and $\mathbf{W}_r$, for any $r \in \{1, \ldots, R\}$.

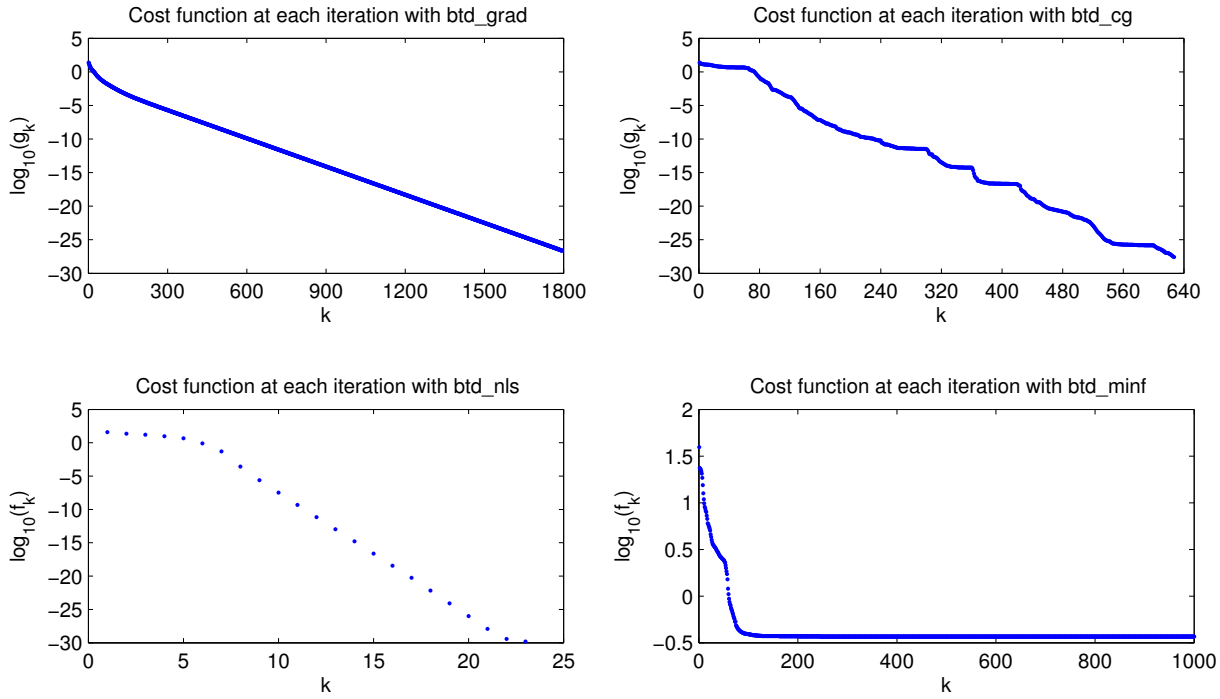


(a) The four methods converge.

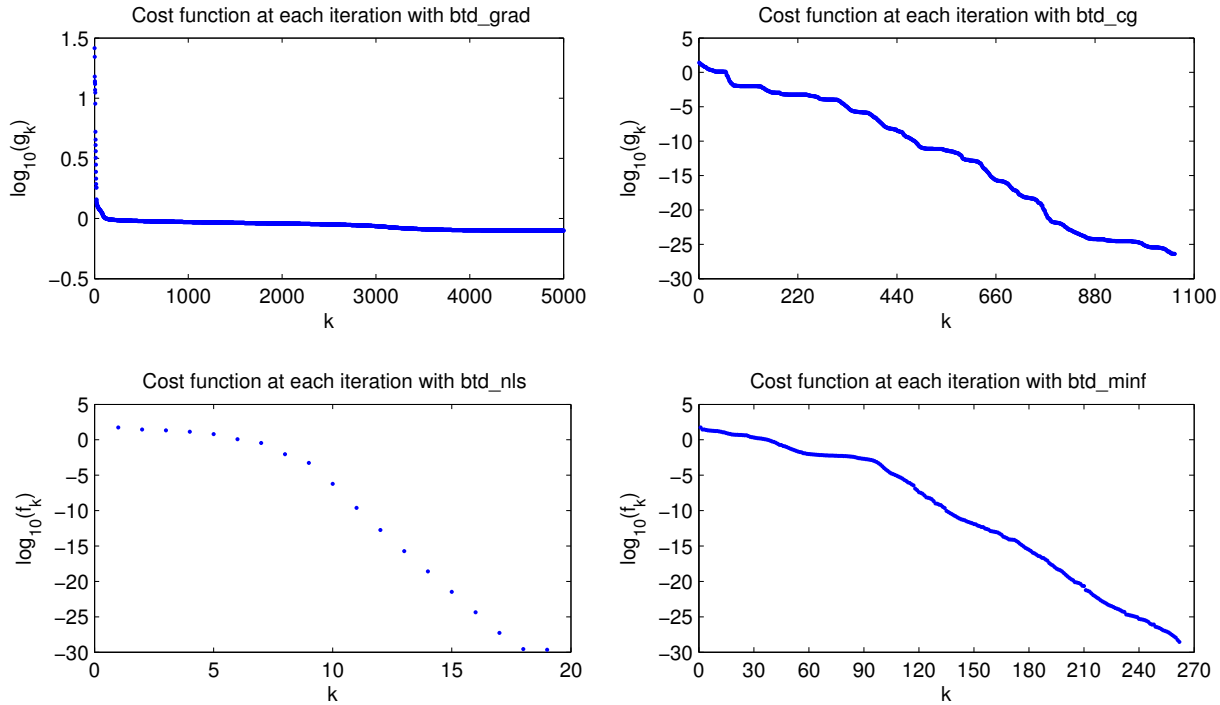
Figure 3.10: Comparison of `btd_grad`, `btd_cg`, `btd_nls` and `btd_minf` for a BTD with 2 terms.

[1] See `http://tensorlab.net/doc/btd.html`.

In Figure 3.10(b)-(d), we see that some of the four methods may converge while the others do not. This shows that all the methods are sensitive to the starting iterate.

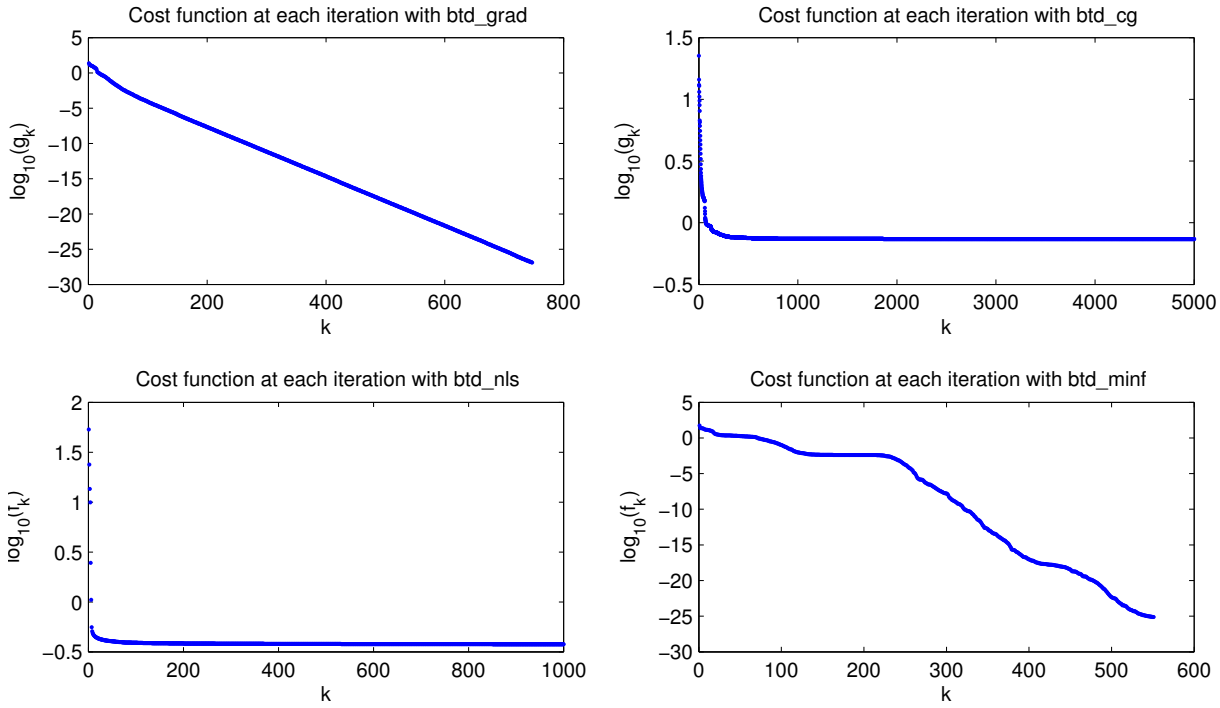


(b) `btd_minf` does not converge.



(c) `btd_grad` does not converge.

Figure 3.10: Comparison of `btd_grad`, `btd_cg`, `btd_nls` and `btd_minf` for a BTD with 2 terms.



(d) `btd_cg` and `btd_nls` do not converge.

Figure 3.10: Comparison of `btd_grad`, `btd_cg`, `btd_nls` and `btd_minf` for a BTD with 2 terms.

The evolutions of the cost function for a BTD with $(I_1, I_2, I_3) := (6, 6, 6)$, $R := 3$ and $(R_1, R_2, R_3) := (2, 3, 2)$ are plotted in Figure 3.11 for the four methods starting from the same point.

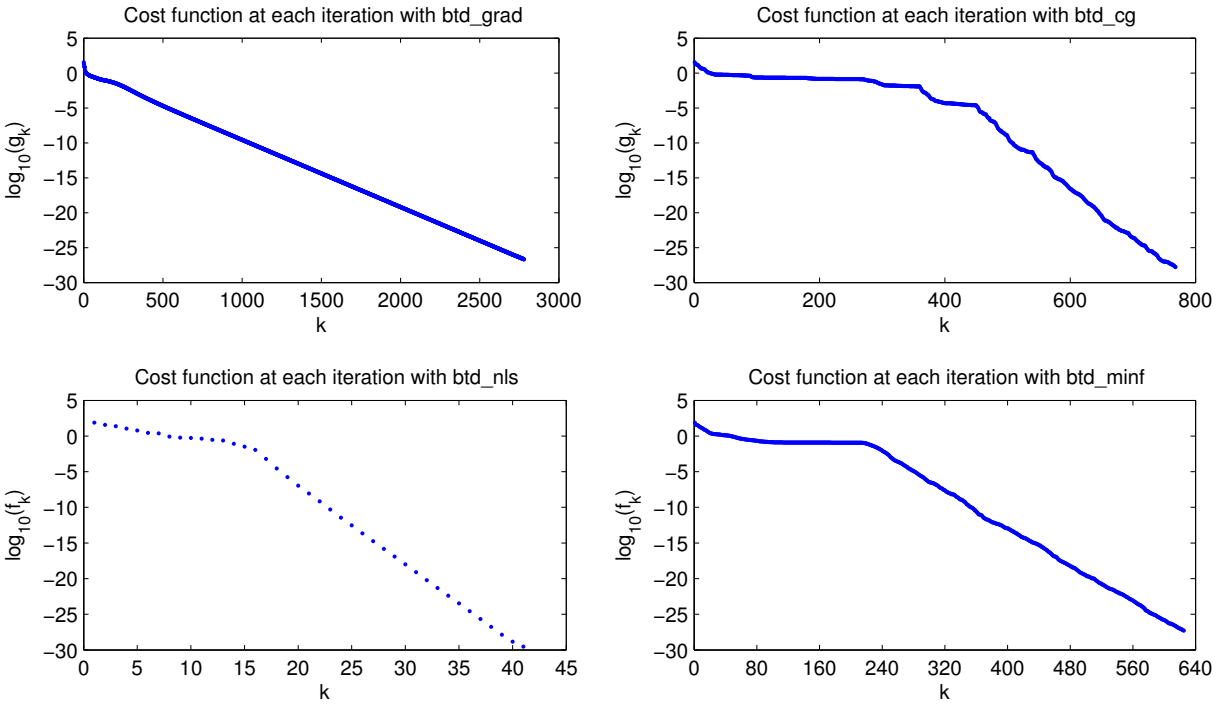


Figure 3.11: Comparison of `btd_grad`, `btd_cg`, `btd_nls` and `btd_minf` for a BTD with 3 terms.

To end this first part of the subsection, let us compare the reliability of the four methods. We ran each of them on three BTDs with $(I_1, I_2, I_3) := (5, 5, 5)$, $R \in \{2, 3, 4\}$ and $(R_1, R_2, R_3) := (2, 2, 2)$ starting from 20 randomly selected points. Table 3.6 presents the numbers of successes for the four methods and $R \in \{2, 3\}$. A success means that the cost function falls below 10^{-25}. To achieve this with `btd_nls` and `btd_minf`, the options `TolFun` and `TolX` were set respectively to 10^{-30} and $3 \cdot 10^{-16}$. With `btd_grad` and `btd_cg`, the tolerance on the norm of the gradient was set to $5 \cdot 10^{-14}$, except for `btd_cg` with $R = 3$ where it was set to $9 \cdot 10^{-14}$.

	`btd_grad`	`btd_cg`	`btd_nls`	`btd_minf`
number of successes	20	20	20	20
minimum number of iterations	379	220	12	98
mean number of iterations	446	307	22	148
maximum number of iterations	569	381	57	215
minimum running time	2.01	1.56	2.44	0.3865
mean running time	2.48	2.29	4.49	0.59
maximum running time	3.24	3.17	11.38	0.88
mean running time per iteration	0.0056	0.0075	0.20	0.0040

(a) $R := 2$

	`btd_grad`	`btd_cg`	`btd_nls`	`btd_minf`
number of successes	4	6	3	2
minimum number of iterations	1851	426	31	353
mean number of iterations	1993	512	57	399
maximum number of iterations	2094	603	33	445
minimum running time	14.19	4.09	13.28	1.74
mean running time	16.21	5.61	25.30	2.08
maximum running time	18.14	6.82	48.81	2.42
mean running time per iteration	0.0081	0.011	0.44	0.0052

(b) $R := 3$

Table 3.6: Runs of `btd_grad`, `btd_cg`, `btd_nls` and `btd_minf` with 20 randomly selected starting iterates for 2 values of R. The extrema and means are computed only over the successful runs. Running times are given in seconds.

For $R = 2$, all the methods converged for each of the 20 starting iterates. As noticed previously, the functions from Tensorlab need less iterations to converge. However, in view of the running times, the computational cost of one iteration seems to be higher for `btd_nls` than for the three other methods. In conclusion, for this test, the four methods perform in a similar way.

As already observed in subsection 3.1.2, the success rates are much lower for $R = 3$ than for $R = 2$. The function `btd_cg` has the highest success rate. However, using more than 20 different starting iterates would be necessary to conclude that the conjugate gradients algorithm is more reliable than the three other methods. The mean running time per iteration obviously increases from $R = 2$ to $R = 3$, especially for `btd_nls`.

Finally, let us discuss the results for $R = 4$. They were not reported in Table 3.6 because `btd_grad`, `btd_cg` and `btd_minf` never succeeded over the 20 attempts. Only `btd_nls` succeeded once, using 526 iterations for a running time of about 369 seconds.

Denoising a known BTD

Let us now test our methods on a BTD on which different noise levels have been added. We set $(I_1, I_2, I_3) := (5, 5, 5)$, $R := 2$, $(R_1, R_2, R_3) := (2, 2, 2)$ and select

- $\mathcal{S}_r \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ for each $r \in \{1, \ldots, R\}$ according to the standard normal distribution, i.e., $\mathcal{S}_r :=$ `randn(`R_1`,`R_2`,`R_3`)` in Matlab,
- $\mathbf{U}_r \in \mathrm{St}(R_1, I_1)$, $\mathbf{V}_r \in \mathrm{St}(R_2, I_2)$ and $\mathbf{W}_r \in \mathrm{St}(R_3, I_3)$ for each $r \in \{1, \ldots, R\}$ according to the standard normal distribution, i.e., $\mathbf{U}_r :=$ `qf(randn(`I_1`,`R_1`))` in Matlab.

Then, we construct the BTD

$$\mathcal{A} := \sum_{r=1}^{R} \mathcal{S}_r \cdot_1 \mathbf{U}_r \cdot_2 \mathbf{V}_r \cdot_3 \mathbf{W}_r$$

and normalize it to 1:

$$\mathcal{A} := \frac{\mathcal{A}}{\|\mathcal{A}\|}.$$

Finally, we select $\mathcal{N} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ according to the standard normal distribution, i.e., $\mathcal{N} :=$ `randn(`I_1`,`I_2`,`I_3`)` in Matlab, and we define

$$A_\sigma := \mathcal{A} + \sigma\mathcal{N}$$

for any real value of the parameter σ which controls the noise level on $\mathcal{A}$.

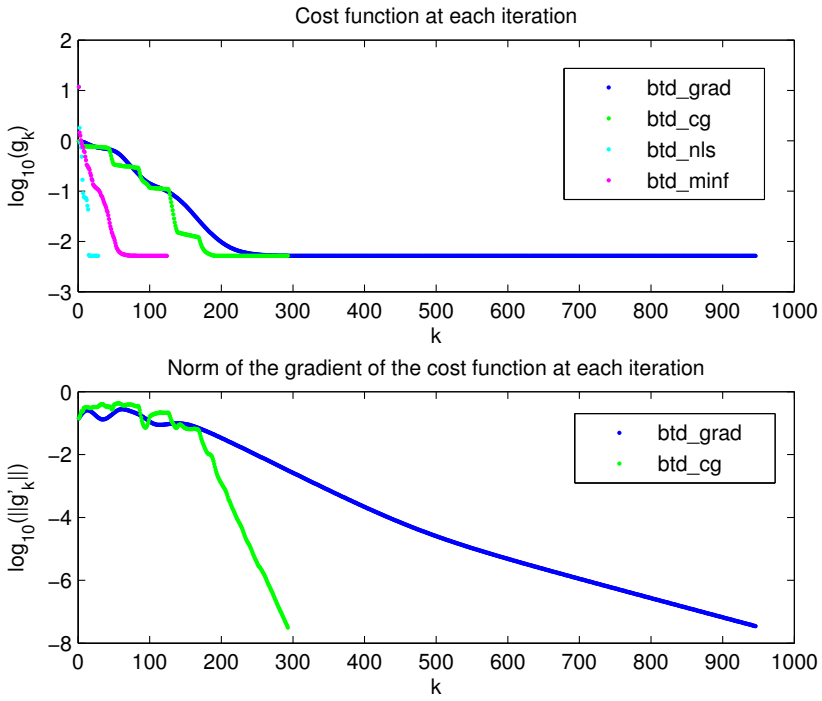
The four methods were run on $\mathcal{A}_\sigma$ for each $\sigma \in \{0.1, 0.5, 1\}$. The functions `btd_nls` and `btd_minf` were used with option `TolFun` set to 10^{-30}. The functions `btd_grad` and `btd_cg` were not able to bring the norm of the gradient below the usual threshold of $5 \cdot 10^{-14}$. Moreover, it appeared that the threshold under which the norm of the gradient cannot fall depends on σ. Therefore, the stopping criterion for `btd_grad` and `btd_cg` based on the norm of the gradient was adapted according to the value of σ as follows: the methods stop as soon as the norm of the gradient falls below $3.5 \cdot 10^{-8}$, $1.9 \cdot 10^{-7}$ and $2.6 \cdot 10^{-7}$ for $\sigma = 0.1$, $\sigma = 0.5$ and $\sigma = 1$, respectively.

The evolutions of the cost function with each of the four methods, as well as the norm of the gradient for `btd_grad` and `btd_cg`, are plotted in Figure 3.12 for three different values of σ. Each subfigure corresponds to a different noise level. Once again, `btd_nls` needs many less iterations to converge than the three other methods. We also observe that `btd_grad` needs many more iterations to bring the norm of the gradient under the threshold than to bring the cost function at its minimum.

Table 3.7 contains the values of $\|\mathcal{A}_\sigma - \hat{\mathcal{A}}_\sigma\|$ and $\|\mathcal{A} - \hat{\mathcal{A}}_\sigma\|$ for each of the three runs, where $\hat{\mathcal{A}}_\sigma$ denotes the estimate of $\mathcal{A}_\sigma$ returned by the different methods.

σ	0.1		0.5		1	
`btd_grad`	0.0716	0.0693	0.3583	0.3664	0.6132	0.8604
`btd_cg`	0.0716	0.0693	0.3583	0.3664	0.6132	0.8604
`btd_nls`	0.0716	0.0693	0.3583	0.3664	0.6132	0.8604
`btd_minf`	0.0716	0.0693	0.3583	0.3664	0.6290	0.8538

Table 3.7: Values of $\|\mathcal{A}_\sigma - \hat{\mathcal{A}}_\sigma\|$ (left subcolumns) and $\|\mathcal{A} - \hat{\mathcal{A}}_\sigma\|$ (right subcolumns) for three values of σ.



(a) $\sigma := 0.1$

Figure 3.12: Runs for three different noise levels.

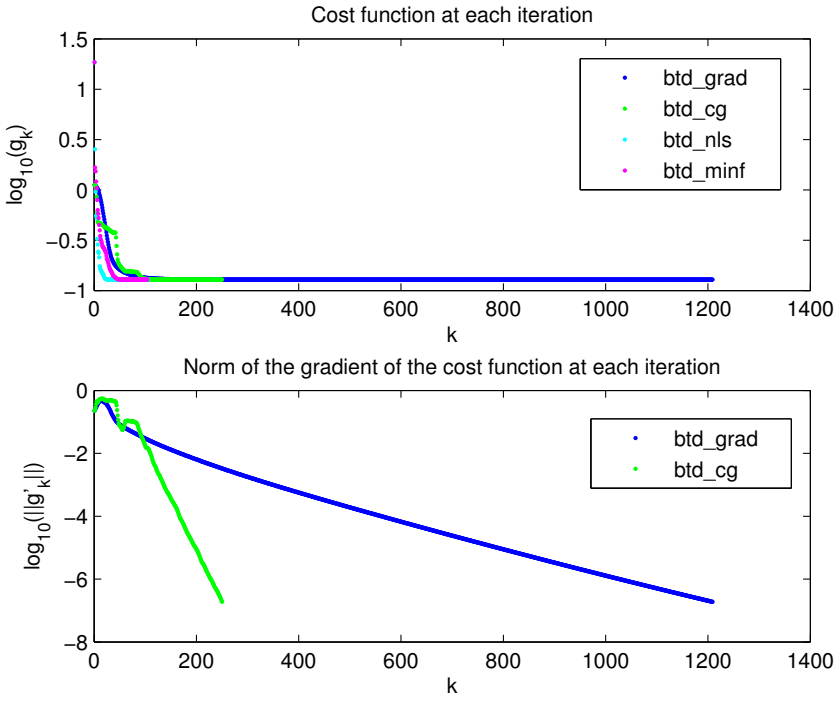
In all cases, $\|\mathcal{A}_\sigma - \hat{\mathcal{A}}_\sigma\| < \sigma$, which shows that the algorithms do not recover exactly the tensor $\mathcal{A}$ from $\mathcal{A}_\sigma$. Nevertheless, it is interesting to note that $\|\mathcal{A}_\sigma - \hat{\mathcal{A}}_\sigma\| > \|\mathcal{A} - \hat{\mathcal{A}}_\sigma\|$ for $\sigma = 0.1$. In other words, the BTD output by the different algorithms is closer to $\mathcal{A}$ than to $\mathcal{A}_\sigma$. For $\sigma \in \{0.5, 1\}$, however, $\|\mathcal{A}_\sigma - \hat{\mathcal{A}}_\sigma\| < \|\mathcal{A} - \hat{\mathcal{A}}_\sigma\|$. This makes sense since, as σ increases, the contribution of $\mathcal{A}$ to the structure of $\mathcal{A}_\sigma$ decreases.

For $\sigma = 0.1$, `btd_minf` does not converge to the same points as the other methods. This shows that the problem admits several local minima. This was already obsvered for the approximation of a third-order tensor by a TKD, i.e., the case $R := 1$, in the paper [13].

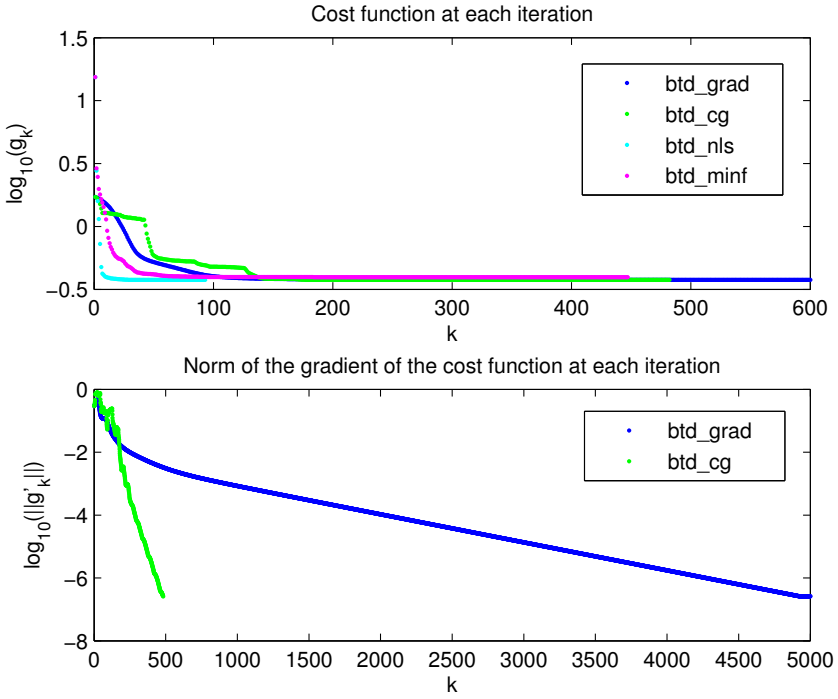
3.2.3 Conclusion

The four methods can be said to be reliable only for BTDs containing two terms, in which case they perform in similar ways. They may work occasionally for BTDs of three terms but the success ratios are small. Finally, the success rates for BTDs with more than three terms are close to zero.

The four methods work well for BTDs of two terms corrupted with different noise levels. However, the higher is the noise level, the higher is the threshold under which `btd_grad` and `btd_cg` cannot bring the norm of the gradient.



(b) $\sigma := 0.5$



(c) $\sigma := 1$. The top plot stops at iteration 600 to improve readibility.

Figure 3.12: Runs for three different noise levels.

Conclusion

Higher-order tensors are more and more used in different areas of applied mathematics. For instance, it is now frequent to use *tensorization*, i.e., to create a data tensor from lower-dimensional original data [2]. We saw that multilinear algebra is structurally more complex than linear algebra, as illustrated by the concept of rank. This results in a greater diversity in the different tensor decompositions. Among them, the BTD is particularly important because of its great flexibility.

In this work, I focused on the computation of the best approximation in the least-squares sense of a third-order tensor by a BTD. Following the work [7] of M. Ishteva, I investigated the opportunity of solving the optimization problem using variable projection. I demonstrated that variable projection allows the reduction of the initial problem to a minimization problem whose feasible set is a Cartesian product of Stiefel manifolds. The main drawback of the variable projection is that it can be performed only numerically which prevents us to compute the Hessian of the cost function and, consequently, to use second-order algorithms.

I implemented the gradient algorithm and the conjugate gradients algorithm, both with a backtracking strategy to select the step size. These two new methods perform much like the already available ones in Tensorlab. Two kinds of tests were performed to reach this conclusion. First, I tried to recover a known BTD starting from a randomly selected point. It was shown that all the methods are quite reliable for BTDs of two terms. However, for BTDs containing more than two terms, they fail in more than half the cases. Secondly, I tried to recover known BTDs of two terms on which a certain noise level had been added. It was shown that the methods still work well in presence of noise. This test also showed that the cost function associated to the BTD approximation problem often has many local minima. This was already noticed for the TKD in [13] and thus it is not surprising that it also holds for the BTD.

The results may seem disappointing, especially when compared with those in [7] for the low multilinear rank approximation. However, it should be recalled that all the runs in [7] were performed starting from the truncated HOSVD, which is in general much better than a randomly generated point. Therefore, designing procedures able to generate cheap and good enough starting iterates for BTDs seems to be a priority. This being said, let us now propose some other methods that could be used to solve the BTD problem. A first possibility is the Riemannian Barzilai-Borwein method with nonmonotone line search developed in [22]. In a different style, it could be interesting to test simulated annealing methods on the BTD problem. Indeed, these methods are designed to find only global extrema of the objective function. They may be useful to solve the BTD problem or at least to provide a good enough starting iterate to the already available methods. Finally, it would be interesting to try to solve the BTD problem using the Newton method, without variable projection of course.

Appendix A

Short reminders

A.1 Cauchy-Schwarz inequality

Proposition A.1.1 (Cauchy-Schwarz inequality)**.** *Let $(X, \langle\cdot,\cdot\rangle)$ be a pre-Hilbert space. For any $u, v \in X$,*

$$|\langle u, v\rangle| \leq \|u\| \, \|v\|$$

and equality is reached if and only if u and v are linearly dependent.

Proof. We assume $v \neq 0$ and define

$$w := u - \frac{\langle u, v\rangle}{\langle v, v\rangle} v.$$

We have $v \perp w$ and so by the Pythagorean identity,

$$\|u\|^2 = \left\| w + \frac{\langle u, v\rangle}{\langle v, v\rangle} v \right\|^2 = \|w\|^2 + \frac{|\langle u, v\rangle|^2}{\|v\|^2} \geq \frac{|\langle u, v\rangle|^2}{\|v\|^2}.$$

We have equality if and only if $w = 0$, i.e., if and only if u and v are linearly dependent. □

A.2 Trace

Let F be an arbitrary field. The *trace* of $\mathbf{A} \in F^{n\times n}$ is

$$\operatorname{tr}(\mathbf{A}) := \sum_{i=1}^{n} \mathbf{A}(i, i).$$

It is easy to check that for any $\mathbf{A}, \mathbf{B} \in F^{m\times n}$,

$$\operatorname{tr}(\mathbf{B}^{\mathrm{T}}\mathbf{A}) = \operatorname{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{B}) = \operatorname{tr}(\mathbf{A}\mathbf{B}^{\mathrm{T}}).$$

A.3 Kronecker product

Definition Let F be an arbitrary field. The *Kronecker product* of $\mathbf{A} \in F^{m\times n}$ and $\mathbf{B} \in F^{p\times q}$ is

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} \mathbf{A}(1,1)\mathbf{B} & \dots & \mathbf{A}(1,n)\mathbf{B} \\ \vdots & & \vdots \\ \mathbf{A}(m,1)\mathbf{B} & \dots & \mathbf{A}(m,n)\mathbf{B} \end{bmatrix} \in F^{mp\times nq}.$$

In other words,

$$(\mathbf{A} \otimes \mathbf{B})(p(i-1)+k, q(j-1)+l) = \mathbf{A}(i,j)\mathbf{B}(k,l)$$

for each $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$ and $(k, l) \in \{1, \dots, p\} \times \{1, \dots, q\}$.

Associativity It is easy to check that the Kronecker product is associative. In the same way as the summation and product symbols are defined, we define

$$\begin{cases} \bigotimes_{n=1}^{0} \mathbf{A}_n := 1, \\ \bigotimes_{n=1}^{N} \mathbf{A}_n := \left(\bigotimes_{n=1}^{N-1} \mathbf{A}_n \right) \otimes \mathbf{A}_N \text{ for each } N \in \mathbb{N}_*. \end{cases}$$

Kronecker product and matrix product The next result is a very important property of the Kronecker product; its proof is a simple manipulation of indices.

Proposition A.3.1. *Let* $\mathbf{A}$*,* $\mathbf{B}$*,* $\mathbf{C}$ *and* $\mathbf{D}$ *be matrices defined over a field. If the products* $\mathbf{AC}$ *and* $\mathbf{BD}$ *are defined, then*

$$(\mathbf{A} \otimes \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}, \qquad (\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}).$$

Proposition A.3.2. *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{p \times q}$ *be written by SVD:*

$$\mathbf{A} = \mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^{\mathrm{T}}, \qquad \mathbf{B} = \mathbf{U}_2 \mathbf{\Sigma}_2 \mathbf{V}_2^{\mathrm{T}}$$

with singular values $\sigma_1, \ldots, \sigma_{R_1}$ *and* $\mu_1, \ldots, \mu_{R_2}$*, respectively. Then the SVD of* $\mathbf{A} \otimes \mathbf{B}$ *is*

$$(\mathbf{U}_1 \otimes \mathbf{U}_2)(\mathbf{\Sigma}_1 \otimes \mathbf{\Sigma}_2)(\mathbf{V}_1^{\mathrm{T}} \otimes \mathbf{V}_2^{\mathrm{T}}).$$

In particular, the singular values of $\mathbf{A} \otimes \mathbf{B}$ *are* $\sigma_i \mu_j$ *for all* $(i, j) \in \{1, \ldots, R_1\} \times \{1, \ldots, R_2\}$.

Proof. This follows from Proposition A.3.1. □

Proposition A.3.3. *If* $\mathbf{Q}_i \in \mathrm{St}(R_i, I_i)$ *for each* $i \in \{1, 2\}$*, then* $\mathbf{Q}_1 \otimes \mathbf{Q}_2 \in \mathrm{St}(R_1 R_2, I_1 I_2)$.

Proof. Observe that $(\mathbf{Q}_1 \otimes \mathbf{Q}_2)^{\mathrm{T}}(\mathbf{Q}_1 \otimes \mathbf{Q}_2) = \mathbf{I}_{R_1 R_2}$. □

Appendix B

Matlab implementations

All the numerical tests were performed using Matlab with the Tensorlab toolbox. Unless otherwise specified, the functions listed below were all written by myself.

B.1 BTD computation

Main functions The following functions try to compute the best approximation in the least-squares sense of a third-order tensor by a BTD.

- `btd_grad` implements the gradient algorithm with backtracking described in subsection 2.5.1.
- `btd_cg` implements the conjugate gradients algorithm with backtracking described in subsection 2.5.2.
- `btd_grad_full` implements the gradient algorithm with backtracking and without variable projection (i.e., it tries to minimize $f_{\mathcal{A}}$, not $g_{\mathcal{A}}$).

Auxiliary functions

- `qf` returns the Q factor of the thin QR decomposition of a given matrix.
  (Author: Pierre-Antoine Absil)
- `ProjStiefel` returns the projection of a given matrix onto the Stiefel manifold.
- `ProjTanStiefel` computes the projection of a given matrix on the tangent space of the Stiefel manifold at another given matrix.
- `f` evaluates the cost function $f_{\mathcal{A}}$ at a given point.
- `grad_f` evaluates the gradient of the cost function $f_{\mathcal{A}}$.
- `grad_g` evaluates the gradient of the cost function $g_{\mathcal{A}}$.
- `CoreTensors` returns the optimal core tensors for given factor matrices.

B.2 General gradient algorithm

- `grad_bktg` implements the gradient algorithm with a backtracking strategy to select the step length.
- `grad_lip` implements the gradient algorithm with a constant step length equal to the inverse of the Lipschitz constant of the gradient of the objective function.

Bibliography

[1] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, USA, 4th edition, 2013.

[2] A. Cichocki, D. Mandic, A.H. Phan, C. Caiafa, G. Zhou, Q. Zhao, and L. De Lathauwer. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, March 2015.

[3] N. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. Papalexakis, and C. Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2016.

[4] L. De Lathauwer. Decompositions of a higher-order tensor in block terms—Part I: Lemmas for partitioned matrices. *SIAM J. Matrix Anal. Appl.*, 30(3):1022–1032, 2008.

[5] L. De Lathauwer. Decompositions of a higher-order tensor in block terms—Part II: Definitions and uniqueness. *SIAM J. Matrix Anal. Appl.*, 30(3):1033–1066, 2008.

[6] L. De Lathauwer and D. Nion. Decompositions of a higher-order tensor in block terms—Part III: Alternating least squares algorithms. *SIAM J. Matrix Anal. Appl.*, 30(3):1067–1083, 2008.

[7] M. Ishteva, P.-A. Absil, S. Van Huffel, and L. De Lathauwer. Best low multilinear rank approximation of higher-order tensors, based on the Riemannian trust-region scheme. *SIAM J. Matrix Anal. Appl.*, 32(1):115–135, 2011.

[8] L. Velho, A.C. Frery, and J. Gomes. *Image Processing for Computer Graphics and Vision*. Texts in Computer Science. Springer, 2nd edition, 2009.

[9] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[10] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.*, 21(4):1324–1342, 2000.

[11] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, New-York, NY, USA, 2nd edition, 2012.

[12] C.J. Hillar and L.-H. Lim. Most tensor problems are NP-hard. *Journal of the ACM*, 60(6):45, 2013.

[13] M. Ishteva, P.-A. Absil, S. Van Huffel, and L. De Lathauwer. Tucker compression and local optima. *Chemometrics and Intelligent Laboratory Systems*, 2010.

[14] O. Debals, M. Van Barel, and L. De Lathauwer. Löwner-based blind signal separation of rational functions with applications. *IEEE Transactions in Signal Processing*, 64(3):1909–1918, Apr. 2016.

[15] B. Hunyadi, D. Camps, L. Sorber, W. Van Paesschen, M. De Vos, S. Van Huffel, and L. De Lathauwer. Block term decomposition for modelling epileptic seizures. *EURASIP Journal on Advances in Signal Processing, Special Issue on Recent Advances in Tensor Based Signal and Image Processing*, 2014(139):1909–1918, Sep. 2014.

[16] L. De Lathauwer. Block component analysis, a new concept for blind source separation. In *Proc. of the 10th International Conference on Latent Variable Analysis and Signal Separation (LVA ICA 2012)*, volume 5, pages 1–8, Mar. 2006.

[17] L. De Lathauwer. Blind separation of exponential polynomials and the decomposition of a tensor in rank-$(L_r, L_r, 1)$ terms. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1451–1474, Dec. 2011.

[18] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, USA, 2008.

[19] P.-A. Absil and Jérôme Malick. Projection-like retractions on matrix manifolds. *SIAM J. Matrix Anal. Appl.*, 22(1):135–158, 2012.

[20] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer, 2004.

[21] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition, 2006.

[22] B. Iannazzo and M. Porcelli. The Riemannian Barzilai-Borwein method with nonmonotone line search and the matrix geometric mean computation. *IMA Journal of Numerical Analysis*, April 2017. Version of February 14, 2017.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**