Recompression of Hadamard products of tensors in Tucker format

Periša, Lana

Sveučilište u Zagrebu

Faculty of Science
Department of Mathematics

Lana Periša

Recompression of Hadamard products of tensors in Tucker format

DOCTORAL THESIS

Zagreb, 2017

Sveučilište u Zagrebu

Prirodoslovno-matematički fakultet
Matematički odsjek

Lana Periša

Rekompresija Hadamardovog produkta tenzora u Tuckerovom formatu

DOKTORSKI RAD

Zagreb, 2017.

Sveučilište u Zagrebu

Faculty of Science
Department of Mathematics

Lana Periša

Recompression of Hadamard products of tensors in Tucker format

DOCTORAL THESIS

Supervisors:
prof.dr.sc. Ivan Slapničar
prof.dr.sc. Daniel Kressner

Zagreb, 2017

Sveučilište u Zagrebu

Prirodoslovno-matematički fakultet
Matematički odsjek

Lana Periša

Rekompresija Hadamardovog produkta tenzora u Tuckerovom formatu

DOKTORSKI RAD

Mentori:
prof.dr.sc. Ivan Slapničar
prof.dr.sc. Daniel Kressner

Zagreb, 2017.

To my late grandfather Marko, who would have been most proud of my life achievements.

Acknowledgements

I would like to thank my supervisors Ivan Slapničar and Daniel Kressner – Ivan for being there in difficult times and Daniel for opening the world of tensors to me and accepting me in his group at EPFL. Also, thank you to everybody from the group for making my stay in Switzerland a great experience. Special thanks to Ivan, Vinka, Roko, Jelena and Ana for making me feel like home.

I would also like to thank my colleagues from PMF, Zagreb - particularly to Erna, Marija, Zvonimir and Zvonko for many helpful discussions, but mostly for their friendship.

Thank you to all my colleagues from FESB, especially to my female colleagues from ZMF for being very supportive and for spending hours listening and giving advises, I really appreciate it.

Big thanks to Danijela and Sven for providing home for me in Zagreb.

Finally, thank you to all my friends and family, especially Nikola, for all the support over the years.

Abstract

In the last decade low-rank tensors decompositions have been established as a new tool in scientific computing to address large-scale linear and multilinear algebra problems, which would be intractable by classical techniques. Since tensors can be given only as the solution of some algebraic equation, it is important to develop solvers working within the compressed storage scheme. That is what this thesis is concerned with, focusing on Tucker format, one of the most commonly used low-rank representation of tensors, and Hadamard product, which features prominently in tensor-based algorithms in scientific computing and data analysis. Fast algorithms are attained by combining iterative methods, such as Lanczos method and randomized algorithms, with fast matrix-vector products that exploit the structure of Hadamard products. Algorithms are implemented in programming language Julia and a new Julia library for tensors in Tucker format is presented.

Key words

Tensors, Tucker format, Tucker decomposition, higher-order singular value decomposition (HOSVD), Hadamard products, low-rank approximation.

Sažetak

Posljednjih godina tenzorske dekompozicije malog ranga postaju bitan alat u znanstvenom računanju kod rješavanja problema velikih dimenzija linearne i multilinearne algebre, koje ne možemo riješiti klasičnim tehnikama. S obzirom na to da tenzori mogu biti zadani kao rješenja neke algebarske jednadžbe, izuzetno je važno razviti algoritme koji rade direktno s komprimiranim tenzorskim formatima. U ovoj radnji fokusiramo se na Tuckerov format, jednu od najčešće korištenih reprezentacija malog ranga, i Hadamardov produkt, koji ima veliku ulogu u tenzorskim algoritmima za znanstveno računanje i obradu podataka. Brze algoritme dobili smo kombinirajući iterativne metode, poput Lanczosove metode i randomiziranih algoritama, s brzim matrično-vektorskim množenjem koje se temelji na posebnoj strukturi Hadamardovog produkta. Algoritmi su implementirani u novu Julia biblioteku.

Ključne riječi

Tenzori, Tuckerov format, Tuckerov rastav, singularna dekompozicija više reda (HOSVD), Hadamardov produkt, aproksimacija malog ranga.

Contents

Chapter 1

Introduction

A *tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is an N-dimensional array, with the integer I_n, for $n = 1, \ldots, N$, denoting the size of the nth "dimension", which is called *mode*. We say that $\mathcal{X}$ is a tensor of order N. Vectors and matrices are tensors of order 1 and 2, respectively, and for visualization we use tensors of order 3 (see Figure 1.1). Each element of a tensor is defined by its N indices and denoted as $x_{i_1 i_2 \cdots i_N}$, $i_n = 1, 2, \ldots, I_n$, $n = 1, 2, \ldots, N$. The *Hadamard product* $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ for two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is defined as the entrywise product $z_{i_1 i_2 \cdots i_N} = x_{i_1 i_2 \cdots i_N} y_{i_1 i_2 \cdots i_N}$, for $i_n = 1, 2, \ldots, I_n$, $n = 1, 2, \ldots, N$.

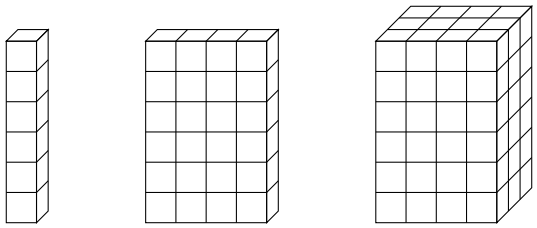


Figure 1.1: Tensors of order one, two and three, with each cube denoting one element.

1.1 Motivation

Tensor decompositions originated with Hitchcock in 1927 [18, 17], but they received scant attention until the second half on the 20th century, when they became extremely popular, first in the fields of psychometrics and chemometrics, and later expanding to other fields, including signal processing, numerical linear algebra, computer vision, numerical analysis, graph analysis, neuroscience and more [20].

The idea of decomposing a tensor into a core tensor multiplied (or transformed) by a

matrix along each mode was first introduced by Tucker in 1963 [30]. Today best known as the *Tucker decomposition*, during the years it went by many different names – three mode factor analysis (3MFA/Tucker3), three mode principal component analysis (3MPCA), N-mode principal component analysis (N-mode PCA), higher order singular value decomposition (HOSVD) and N-mode singular value decomposition (N-mode SVD).

One of the methods Tucker introduced in his work and referred to as the "Tucker1" method, today is a well-established approach for obtaining the Tucker decomposition and is better known as the higher-order singular value decomposition (HOSVD) from the work of De Lathauwer, De Moor and Vandewalle in 2000 [8], who showed that the HOSVD is a convincing generalization of the matrix SVD.

There are various applications of the Tucker decomposition, in image processing for face recognition and human motion, modeling and transferring facial expressions, image data compression, construction of Kronecker product approximations for preconditioners and rendering texture in two-dimensional images, for watermarking MPEG videos, in data mining for identifying handwritten digits, in multimode statistical analysis and clustering, etc. [20].

The Hadamard product is a fundamental building block of tensor-based algorithms in scientific computing and data analysis. This is partly because Hadamard products of tensors correspond to products of multivariate functions. To see this, consider two functions $u, v : [0,1]^N \to \mathbb{R}$ and discretizations $0 \leq \xi_1^{(n)} < \xi_2^{(n)} < \cdots < \xi_{I_n}^{(n)} \leq 1$ of the interval $[0,1]$. Let the tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ contain the functions u and v evaluated on these grid points, that is, $x_{i_1 \cdots i_N} = u(\xi_{i_1}^{(1)}, \ldots, \xi_{i_N}^{(N)})$ and $y_{i_1 \cdots i_N} = v(\xi_{i_1}^{(1)}, \ldots, \xi_{i_N}^{(N)})$, for $i_n = 1, \ldots, I_n$. Then $\boldsymbol{\mathcal{Z}}$, a tensor containing the function values of the product uv, satisfies $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}$. Applied recursively, Hadamard products allow to deal with other nonlinearities, including polynomials in u, such as $1 + u + u^2 + u^3 + \cdots$, and functions that can be well approximated by a polynomial, such as $\exp(u)$. Other applications of Hadamard products include the computation of level sets [9], reciprocals [22], minima and maxima [11], variances and higher-order moments in uncertainty quantification [6, 10], as well as weighted tensor completion [12].

The N-tuple of ranks of mode-n matricizations of a tensor of order N is referred to as the multilinear rank of a tensor, and the Tucker format compactly represents tensors of low multilinear rank. For an $I_1 \times \cdots \times I_N$ tensor of multilinear rank $(R_1, \ldots, R_N)$ only

$R_1 \cdots R_N + R_1 I_1 + \cdots + R_N I_N$ instead of $I_1 \cdots I_N$ entries need to be stored. This makes the Tucker format particularly suitable for tensors of low order (say, $N = 3, 4, 5$) and it in fact constitutes the basis of the Chebfun3 software for trivariate functions [16].

For function-related tensors, it is known that smoothness of u, v implies that $\mathcal{X}, \mathcal{Y}$ can be well approximated by low-rank tensors [14, 26]. In turn, tensor $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ also admits a good low-rank approximation. This property is not fully reflected on the algebraic side; the Hadamard product generically multiplies, and thus often drastically increases, multilinear ranks. In turn, this makes it difficult to exploit low ranks in the design of computationally efficient algorithms for Hadamard products.

1.2 Contributions of the thesis

The main contribution of this thesis is the development of new algorithms for recompression of Hadamard products of tensors in Tucker format by combining an existing method for Tucker decomposition - the higher-order singular value decomposition, with iterative methods for low-rank matrix approximations, such as Lanczos [27] and randomized [15] algorithms. These algorithms are based on matrix-vector multiplication, so we exploit the well-known structure of the Hadamard products to create new fast matrix-vector multiplication suitable for this particular problem, which results in significant reduction of computational and memory requirements.

We investigate all possibilities for achieving such reductions, resulting in four different algorithms called HOSVD1, HOSVD2, HOSVD3 and HOSVD4. For example, when dealing with tensors of order $N = 3$ with modes of equal size $I_1 = I_2 = I_3 = I$ and all involved multilinear ranks bounded by R, we will see below that the straightforward recompression technique from HOSVD2 algorithm, based on combining the HOSVD with the randomized algorithms requires $\mathcal{O}(IR^4 + R^8)$ operations and $\mathcal{O}(IR^2 + R^6)$ memory. This excessive cost is primarily due to the construction of an intermediate $R^2 \times R^2 \times R^2$ core tensor, limiting such an approach to small values of R. Algorithms HOSVD3 and HOSVD4 avoid this effect by exploiting structure when performing multiplications with the matricizations of $\mathcal{Z}$. The HOSVD4 algorithm requires only $\mathcal{O}(IR^3 + R^6)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory.

More precisely:

- HOSVD1 algorithm implements straightforward approach - first create full tensors out of the given Tucker tensors, multiply them element-wise and then get the Tucker representation of the product by applying HOSVD, which we speed up by combining it with randomized algorithm for calculation of factor matrices.

- HOSVD2 algorithm exploits the structure by getting the QR factorization of each of the factor matrices from the known representation, uses orthonormal matrices as the resulting factor matrices and multiplies the core tensor by upper triangular matrices. If needed, it performs additional HOSVD on the resulting core tensor to achieve requested multilinear rank, using randomized algorithm. This algorithm gives best results when applied on tenors with small multilinear ranks.

- HOSVD3 algorithm is the one that fully exploits the structure. It calculates factor matrices by applying Lanczos algorithm combined with structure-exploiting matrix-vector multiplication and also exploits the structure for core tensor calculation. This algorithm works good for large tensors with reasonably large multilinear ranks.

- HOSVD4 algorithm calculates factor matrices as orthonormal basis for approximations of range of the matricized tensor by combining randomized algorithm with structure-exploiting matrix-vector multiplication with rank-one vectors. The structure is again exploited to get core tensor and additional HOSVD is performed if necessary, to attain requested multilinear rank. This combination is the most promising one and we will show that it can work with extremely large tensors with reasonably large multilinear ranks.

All algorithms presented in this thesis have been implemented in the Julia programming language [5], in order to attain reasonable speed in our numerical experiments at the convenience of a MATLAB implementation. As a by-product of this work, we provide a novel Julia library called *TensorToolbox*, which is meant to be a general-purpose library for tensors and tensors in Tucker format. It is available on `https://github.com/lanaperisa/TensorToolbox.jl` and includes all functionality of the `ttensor` class from the MATLAB Tensor toolbox [3].

The paper following this research is published in SIAM Journal on Scientific Computing [21].

1.3 Chapter-by-chapter overview

As stated in the definition of a tensor at the beginning of this chapter, we will only be interested in tensors with real entries. Even though most of the theory can be generalized to tensors over field $\mathbb{C}$, this will not be of our interest.

Throughout the text, when needed, we use Julia code to explain how certain operations are computed. The Julia syntax is very similar to MATLAB, though easily understandable.

Chapter 2 is the preliminaries chapter in which we present matrix theory and algorithms needed for our work with tensors. Particularly important are the properties of different matrix products from Section 2.2 and the details of Lanczos and randomized algorithms presented in Section 2.3.

In Chapter 3 we introduce basic tensors operations - mode-n matricization and n-mode product, together with their properties (Section 3.1), define tensors in Tucker format and explain its importance in Section 3.2, while in Sections 3.3 and 3.4 we present algorithms for obtaining the Tucker representation of a given tensor – the standard HOSVD algorithm and its modification HOSVD-AR, together with most important result involving them. In Section 3.5 we explain how some of the presented operations and algorithms can be efficiently computed.

In Chapter 4 we introduce the problem of recompression of Hadamard products of tensors in Tucker format and offer and analyze ideas on how to solve it, introducing new operations for fast matrix-vector multiplication and calculation of core tensor, which are then studied in details in Chapter 5.

Chapter 6 includes complexity analysis of the algorithms in Section 6.1, and numerical experiments in Section 6.2, where we compare all four algorithms and provide the recommendation for which algorithm to use depending on the sizes and multilinear ranks of the involved tensors.

The functionality of the Julia package is presented in Appendix A.

1.4 Computational environment

All algorithms from this thesis are implemented and tested in Julia version 0.5.2., on a PC with an Intel(R) Core(TM) i5-3470 quadcore 3.20GHz CPU, 256KB L2 cache, 6MB L3 cache, and 4GB of RAM. Multithreading is turned on and all four cores are utilized.

1.5 Notation

Throughout this thesis we follow notation introduced in [20]. Real numbers or scalars are denoted as lowercase letters $x, y \in \mathbb{R}$, vectors as boldface lowercase letters $\mathbf{x} = (x_i) \in \mathbb{R}^n$ and matrices as boldface capital letters $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$. Tensors are denoted by boldface Euler script letters $\boldsymbol{\mathcal{X}} = (x_{i_1 i_2 \cdots i_N}) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ (for LaTeX formatting see [19, Appendix A]).

The columns and rows of a matrix are written, respectively, as

$$\mathbf{A} = [\, \mathbf{a}_1 \, \cdots \, \mathbf{a}_n \,] \quad \text{and} \quad \mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix}.$$

We use Julia notation to denote subset of matrix columns and rows. For example, we can partition matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ as

$$\mathbf{A} = \Big[\mathbf{A}[:, 1:k] \quad \mathbf{A}[:, k+1:m] \Big], \quad \text{for any } 1 \leq k \leq n,$$

or

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}[1:l, :] \\ \mathbf{A}[l+1:n, :] \end{bmatrix}, \quad \text{for any } 1 \leq l \leq m.$$

Sometimes we will also use short notation $\mathbf{A}_{:k} = \mathbf{A}[:, k]$ and $\mathbf{x}_{1:k} = \mathbf{x}[1:k]$ for $\mathbf{x} \in \mathbb{R}^n$, $1 \leq k \leq n$.

Chapter 2

Preliminaries

In this chapter we provide preliminaries for our work with tensors - Section 2.1 introduces basic results of linear algebra, vector and matrix norms, inner product and matrix factorizations; different matrix products and their properties are presented in Section 2.2, while Section 2.3 contains theory of iterative methods for low-rank matrix approximation. We are interested only in matrices with real entries, so we restrict our discussion to field $\mathbb{R}$ whenever possible.

2.1 Matrix analysis

Throughout this section we follow [13] and additionally [29], omitting the proofs that are not important for our work.

2.1.1 Linear algebra basics

There are two important subspaces associated with a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. The *range* (*column space*) and the *null space* of $\mathbf{A}$ are defined, respectively, as

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{y} \in \mathbb{R}^m \mid \mathbf{y} = \mathbf{A}\mathbf{x}, \text{ for some } \mathbf{x} \in \mathbb{R}^n\},$$
$$\mathcal{N}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} = \mathbf{0}\}.$$

Furthermore, if $\mathbf{A} = [\,\mathbf{a}_1 \cdots \mathbf{a}_n\,]$, then

$$\mathcal{R}(\mathbf{A}) = \text{span}\,\{\mathbf{a}_1, \cdots, \mathbf{a}_n\}.$$

The *rank* of a matrix $\mathbf{A}$ is defined as

$$rank(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A})).$$

Rank is unchanged by left or right multiplication by a non-singular matrix and it holds $rank(\mathbf{A}) = rank(\mathbf{A}^T)$, from which it follows $rank(\mathbf{A}) \leq \min\{m, n\}$. If $rank(\mathbf{A}) = \min\{m, n\}$, we say that $\mathbf{A}$ has *full rank*.

2.1.2 Norms and inner product

For a vector $\mathbf{x} = (x_i) \in \mathbb{R}^n$ we use the 2-*norm*

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}} \quad \left(= \sqrt{\mathbf{x}^*\mathbf{x}}, \text{ if } \mathbf{x} \in \mathbb{C}^n\right),$$

while for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m\times n}$ we use two different norms - the *(induced)* 2-*norm*

$$\|\mathbf{A}\|_2 = \sup_{\mathbf{x}\in\mathbb{R}^n} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sup_{\substack{\mathbf{x}\in\mathbb{R}^n \\ \|\mathbf{x}\|_2=1}} \|\mathbf{A}\mathbf{x}\|_2,$$

and the *Frobenius norm*

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^T\mathbf{A})} = \sqrt{\mathrm{tr}(\mathbf{A}\mathbf{A}^T)}. \tag{2.1}$$

Here, $\mathrm{tr}(\mathbf{A})$ denotes the *trace* of $\mathbf{A}$, the sum of its diagonal elements, for which

$$\mathrm{tr}(\mathbf{A}\mathbf{B}) = \mathrm{tr}(\mathbf{B}\mathbf{A})$$

holds. For any $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, matrix norms satisfy

- $\|\mathbf{A}\|_2 = \|\mathbf{A}^T\|_2, \quad \|\mathbf{A}\|_F = \|\mathbf{A}^T\|_F,$
- $\|\mathbf{A}\mathbf{B}\|_2 \leq \|\mathbf{A}\|_2\|\mathbf{B}\|_2, \quad \|\mathbf{A}\mathbf{B}\|_F \leq \|\mathbf{A}\|_F\|\mathbf{B}\|_F,$
- $\|\mathbf{A}\mathbf{B}\|_F \leq \|\mathbf{A}\|_F\|\mathbf{B}\|_2, \quad \|\mathbf{A}\mathbf{B}\|_F \leq \|\mathbf{A}\|_2\|\mathbf{B}\|_F.$ (2.2)

Furthermore, the *inner product* of matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$,

$$\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{tr}\left(\mathbf{A}^T\mathbf{B}\right), \tag{2.3}$$

corresponds to Frobenius norm, $\|\mathbf{A}\|_F = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle}$. The following property holds,

$$\|\mathbf{A} - \mathbf{B}\|_F^2 = \|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2 - 2\langle \mathbf{A}, \mathbf{B} \rangle. \tag{2.4}$$

2.1.3 Orthogonality and the SVD

A set of $\mathbb{R}^n$ vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ is *orthogonal* if $\mathbf{x}_i^T \mathbf{x}_j = 0$, whenever $i \neq j$, and *orthonormal* if it is orthogonal and $\|\mathbf{x}_j\|_2 = 1$, for every $j = 1, \ldots, k$.

A collection of subspaces $\mathcal{S}_1, \ldots, \mathcal{S}_k$ in $\mathbb{R}^n$ is *mutually orthogonal* if $\mathbf{x}^T \mathbf{y} = 0$ whenever $\mathbf{x} \in \mathcal{S}_i$ and $\mathbf{y} \in \mathcal{S}_j$, for $i \neq j$. The *orthogonal complement* of a subspace $\mathcal{S} \subset \mathbb{R}^n$ is defined by

$$\mathcal{S}^{\perp} = \left\{ \mathbf{y} \in \mathbb{R}^n \mid \mathbf{y}^T \mathbf{x} = 0, \text{ for all } \mathbf{x} \in \mathcal{S} \right\},$$

and it is easy to show that $\mathcal{R}(\mathbf{A})^{\perp} = \mathcal{N}(\mathbf{A}^T)$, for any matrix $\mathbf{A}$.

A matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is said to be *orthogonal* if $\mathbf{Q}^T \mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$, where $\mathbf{I}$ is the $n \times n$ identity matrix. Columns $\mathbf{q}_j$ of the orthogonal $\mathbf{Q}$ form an orthonormal basis for $\mathbb{R}^n$. A rectangular matrix with orthonormal columns, i.e. $\mathbf{Q} \in \mathbb{R}^{m \times n}$ such that $\mathbf{Q}^T \mathbf{Q} = \mathbf{I}$, is called *orthonormal*.

The vector 2-norm is invariant under orthogonal transformations,

$$\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2, \quad \text{whenever } \mathbf{Q}^T \mathbf{Q} = \mathbf{I},$$

and for any $m \times n$ orthonormal $\mathbf{Q}$,

$$\|\mathbf{Q}\|_2 = 1, \quad \|\mathbf{Q}\|_F = \sqrt{n}. \tag{2.5}$$

The matrix 2-norm and the Frobenius norm are also invariant with respect to orthogonal transformations - for $\mathbf{U}$ and $\mathbf{V}$ orthonormal,

$$\|\mathbf{U}\mathbf{A}\mathbf{V}^T\|_2 = \|\mathbf{A}\|_2, \quad \|\mathbf{U}\mathbf{A}\mathbf{V}^T\|_F = \|\mathbf{A}\|_F. \tag{2.6}$$

Theorem 2.1.1. *[13, Theorem 2.5.2] For every $\mathbf{A} \in \mathbb{R}^{m \times n}$ there exist orthogonal matrices*

$$\mathbf{U} = [\, \mathbf{u}_1 \cdots \mathbf{u}_m \,] \in \mathbb{R}^{m \times m} \quad \textit{and} \quad \mathbf{V} = [\, \mathbf{v}_1 \cdots \mathbf{v}_n \,] \in \mathbb{R}^{n \times n}$$

such that

$$\mathbf{U}^T \mathbf{A} \mathbf{V} = \mathbf{\Sigma},$$

where $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix, whose diagonal entries σ_j are nonnegative and in nonincreasing order, that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$, where $p = \min\{m, n\}$.

This theorem defines the decomposition known as the *singular value decomposition* (SVD) of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{2.7}$$

Entries $\sigma_1, \ldots, \sigma_p$ are uniquely determined and called the *singular values* of $\mathbf{A}$, while orthonormal vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ represent the *left singular vectors* and the *right singular vectors* of $\mathbf{A}$, respectively.

The SVD reveals a great deal about the structure of a matrix. If the SVD of $\mathbf{A}$ is given by (2.7), and we define r by $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$, then $rank(\mathbf{A}) = r$ and

$$\mathcal{R}(\mathbf{A}) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}, \tag{2.8}$$

$$\mathcal{N}(\mathbf{A}) = \text{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}, \tag{2.9}$$

and we have the SVD expansion

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T.$$

The matrix 2-norm and the Frobenius norm are connected to the SVD,

$$\|\mathbf{A}\|_2 = \sigma_1, \quad \|\mathbf{A}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}, \tag{2.10}$$

therefore the equivalence

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r}\|\mathbf{A}\|_2. \tag{2.11}$$

Eliminating $\sigma_{r+1} = \cdots = \sigma_p = 0$ from $\mathbf{\Sigma}$ and appropriate columns of $\mathbf{U}$ and $\mathbf{V}$ in (2.7) leads to the *truncated SVD*

$$\mathbf{A} = \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T, \tag{2.12}$$

with $\mathbf{U}_r \in \mathbb{R}^{m \times r}$, $\mathbf{V}_r \in \mathbb{R}^{n \times r}$ orthonormal and $\mathbf{\Sigma}_r = diag(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r \times r}$.

SVD also defines the best low-rank approximation to a given matrix.

Theorem 2.1.2. *[13, Theorem 2.5.3] Let the SVD of $\mathbf{A} \in \mathbb{R}^{m \times n}$ be given by Theorem 2.1.1. If $k < r = rank(\mathbf{A})$ and*

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T,$$

then

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \min_{rank(B) \leq k} \|\mathbf{A} - \mathbf{B}\|_2 = \sigma_{k+1}$$

and

$$\|\mathbf{A} - \mathbf{A}_k\|_F = \min_{rank(B) \leq k} \|\mathbf{A} - \mathbf{B}\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}.$$

The SVD can be computed in $\mathcal{O}(\min\{mn^2, m^2n\})$ operations, after which it can be used as a tool for different problems - determining the rank of a matrix can be done by counting the number of singular values greater than a judiciously chosen tolerance and finding an orthonormal basis of a range or a null space via (2.8) and (2.9). Also, the standard method for computing the 2-norm and the Frobenius norm of a matrix is by its singular values (2.10).

2.1.4 Orthogonal projectors

Let $\mathcal{S} \subset \mathbb{R}^n$ be a subspace. Matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ is the *orthogonal projector* onto $\mathcal{S}$ if

$$\mathbf{P}^T = \mathbf{P}, \tag{2.13}$$

$$\mathbf{P}^2 = \mathbf{P}, \tag{2.14}$$

$$\mathcal{R}(\mathbf{P}) = \mathcal{S}. \tag{2.15}$$

For any $\mathbf{x} \in \mathcal{S}$, $\mathbf{P}\mathbf{x} = \mathbf{x}$. If $\mathbf{P}$ is an orthogonal projector, $\mathbf{I} - \mathbf{P}$ is also an orthogonal projector, called the *complementary projector* to $\mathbf{P}$, and it projects onto $\mathcal{S}^\perp$, or the nullspace of $\mathbf{P}$. It is easy to show that (2.13) and (2.14) hold for $\mathbf{I} - \mathbf{P}$. We can see that (2.15) also holds by taking $\mathbf{x} \in \mathcal{N}(\mathbf{P})$. Then

$$\mathbf{P}\mathbf{x} = \mathbf{0} \quad \Rightarrow \quad (\mathbf{I} - \mathbf{P})\mathbf{x} = \mathbf{x},$$

which gives $\mathcal{N}(\mathbf{P}) \subset \mathcal{R}(\mathbf{I} - \mathbf{P})$. Conversely, for any $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{P}(\mathbf{I} - \mathbf{P})\mathbf{x} = (\mathbf{P} - \mathbf{P}^2)\mathbf{x} = \mathbf{0},$$

giving $\mathcal{R}(\mathbf{I} - \mathbf{P}) \subset \mathcal{N}(\mathbf{P})$. We conclude $\mathcal{R}(\mathbf{I} - \mathbf{P}) = \mathcal{S}^\perp$.

If $\mathbf{P}_1$ and $\mathbf{P}_2$ are each orthogonal projectors onto $\mathcal{S}$, then for any $\mathbf{x} \in \mathbb{R}^n$ we have

$$\begin{aligned} \|(\mathbf{P}_1 - \mathbf{P}_2)\mathbf{x}\|_2^2 &= \mathbf{x}^T(\mathbf{P}_1 - \mathbf{P}_2)^T(\mathbf{P}_1 - \mathbf{P}_2)\mathbf{x} \\ &= \mathbf{x}^T(\mathbf{P}_1 - \mathbf{P}_1^T\mathbf{P}_2 - \mathbf{P}_2^T\mathbf{P}_1 + \mathbf{P}_2)\mathbf{x} \\ &= (\mathbf{P}_1\mathbf{x})^T(\mathbf{I} - \mathbf{P}_2)\mathbf{x} + (\mathbf{P}_2\mathbf{x})^T(\mathbf{I} - \mathbf{P}_1)\mathbf{x}. \end{aligned} \tag{2.16}$$

From $\mathcal{R}(\mathbf{P}_1) = \mathcal{R}(\mathbf{P}_2) = \mathcal{S}$ follows $\mathbf{P}_1\mathbf{x} = \mathbf{P}_2\mathbf{x} = \mathbf{x}$, therefore the right-hand side of (2.16) is zero, showing that the orthogonal projector for a subspace is unique.

If the columns of $\mathbf{Q} = [\,\mathbf{q}_1 \cdots \mathbf{q}_k\,]$ form an orthonormal basis for a subspace $\mathcal{S} \subset \mathbb{R}^n$, then $\mathbf{P} = \mathbf{Q}\mathbf{Q}^T$ is the unique orthogonal projector onto $\mathcal{S}$. Obviously, (2.13) and (2.14)

hold. And since every $\mathbf{x} \in \mathcal{S}$ can be written as

$$\mathbf{x} = \sum_{j=1}^{k} (\mathbf{q}_j^T \mathbf{x}) \mathbf{q}_j = \sum_{j=1}^{k} (\mathbf{q}_j \mathbf{q}_j^T) \mathbf{x},$$

we have $\mathbf{x} \in \mathcal{R}(\mathbf{Q}\mathbf{Q}^T)$, and vise versa, if $\mathbf{x} \in \mathcal{R}(\mathbf{Q}\mathbf{Q}^T)$, there exists $\mathbf{y} \in \mathbb{R}^n$ such that

$$\mathbf{x} = \sum_{j=1}^{k} (\mathbf{q}_j \mathbf{q}_j^T) \mathbf{y} = \sum_{j=1}^{k} (\mathbf{q}_j^T \mathbf{y}) \mathbf{q}_j,$$

following $\mathbf{x} \in \mathcal{S}$. This gives $\mathcal{R}(\mathbf{Q}\mathbf{Q}^T) = \mathcal{S}$.

Moreover, if $\mathbf{Q}$ is any orthonormal matrix, then $\mathbf{P} = \mathbf{Q}\mathbf{Q}^T$ is a unique projector onto $\mathcal{R}(\mathbf{Q})$.

There are several important orthogonal projectors associated with the singular value decomposition. Suppose $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the SVD of $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $r = rank(\mathbf{A})$. If we have the $\mathbf{U}$ and $\mathbf{V}$ partitionings

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_r & \tilde{\mathbf{U}}_r \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} \mathbf{V}_r & \tilde{\mathbf{V}}_r \end{bmatrix},$$

then, from (2.8) and (2.9),

$$\begin{aligned}
\mathbf{U}_r \mathbf{U}_r^T &= \text{ projector onto } \mathcal{R}(\mathbf{A}), \\
\tilde{\mathbf{U}}_r \tilde{\mathbf{U}}_r^T &= \text{ projector onto } \mathcal{R}(\mathbf{A})^\perp = \mathcal{N}(\mathbf{A}^T), \\
\mathbf{V}_r \mathbf{V}_r^T &= \text{ projector onto } \mathcal{N}(\mathbf{A})^\perp = \mathcal{R}(\mathbf{A}^T), \\
\tilde{\mathbf{V}}_r \tilde{\mathbf{V}}_r^T &= \text{ projector onto } \mathcal{N}(\mathbf{A}).
\end{aligned}$$

Furthermore, for an orthogonal projector $\mathbf{P}$,

$$\|\mathbf{P}\mathbf{x}\|_2 \leq \|\mathbf{x}\|_2, \text{ for any vector } \mathbf{x} \text{ and} \tag{2.17}$$

$$\|\mathbf{P}\mathbf{A}\|_F \leq \|\mathbf{A}\|_F, \text{ for any matrix } \mathbf{A}. \tag{2.18}$$

The first inequality follows from the fact that $\mathbf{x}$ can be written as a sum of two orthogonal components $\mathbf{x} = \mathbf{P}\mathbf{x} + (\mathbf{I} - \mathbf{P})\mathbf{x}$, therefore $\|\mathbf{x}\|_2 = \|\mathbf{P}\mathbf{x}\|_2 + \|(\mathbf{I} - \mathbf{P})\mathbf{x}\|_2$. The second one follows from the Frobenius norm characterization (2.1),

$$\begin{aligned}
\|\mathbf{A}\|_F^2 - \|\mathbf{P}\mathbf{A}\|_F^2 &= \text{tr}(\mathbf{A}^T\mathbf{A}) - \text{tr}(\mathbf{A}^T\mathbf{P}^T\mathbf{P}\mathbf{A}) \\
&= \text{tr}(\mathbf{A}^T\mathbf{A}) - \text{tr}(\mathbf{A}^T\mathbf{P}\mathbf{A}) = \text{tr}(\mathbf{A}^T (\mathbf{I} - \mathbf{P}) \mathbf{A}) \\
&= \text{tr}\Big[\mathbf{A}^T (\mathbf{I} - \mathbf{P})^T (\mathbf{I} - \mathbf{P}) \mathbf{A}\Big]
\end{aligned}$$

$$= \| (\mathbf{I} - \mathbf{P})\, \mathbf{A}\|_F^2 \geq 0.$$

From (2.17) follows $\|\mathbf{P}\|_2 \leq 1$, while on the other hand, property (2.14) gives $\|\mathbf{P}\|_2 = \|\mathbf{P}^2\|_2 \leq \|\mathbf{P}\|_2^2$, from which it follows $\|\mathbf{P}\|_2 \geq 1$. Therefore, every orthogonal projector $\mathbf{P} \neq 0$ satisfies

$$\|\mathbf{P}\|_2 = 1. \tag{2.19}$$

2.1.5 Spectral decomposition of a symmetric matrix

The *eigenvalues* of a general quadratic matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ are the n roots of its *characteristic polynomial* $p(\mathbf{x}) = \det(\mathbf{xI} - \mathbf{A})$, where $\det(\mathbf{A})$ denotes the determinant of a matrix. The set of these roots is called the *spectrum* and is denoted by $\boldsymbol{\lambda}(\mathbf{A})$. Since polynomials with real coefficients can have complex roots, $\boldsymbol{\lambda}(\mathbf{A}) \subset \mathbb{C}$. If $\lambda \in \boldsymbol{\lambda}(\mathbf{A})$, then the nonzero vectors $\mathbf{x}$ that satisfy

$$\mathbf{Ax} = \lambda\mathbf{x} \tag{2.20}$$

are referred to as the *eigenvectors*. Obviously, $\mathbf{x}$ can be complex, too, so in general we have $\mathbf{x} \in \mathbb{C}^n$.

If $\boldsymbol{\lambda}(\mathbf{A}) = \{\lambda_1, \ldots, \lambda_n\}$, then

$$\det(\mathbf{A}) = \lambda_1\lambda_2\cdots\lambda_n \quad \text{and} \quad \operatorname{tr}(\mathbf{A}) = \lambda_1 + \cdots + \lambda_n.$$

Also, we can characterize the matrix 2-norm by its largest eigenvalue

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})}. \tag{2.21}$$

Setting $\boldsymbol{\Lambda} = diag(\lambda_1, \ldots, \lambda_n)$ and $\mathbf{X} = [\,\mathbf{x}_1 \cdots \mathbf{x}_n\,]$, with $\mathbf{x}_j$ being the eigenvector associated with each eigenvalue λ_j, (2.20) can be written as $\mathbf{AX} = \mathbf{X\Lambda}$. Moreover, if each eigenvalue has the same number of linearly independent eigenvectors associated with it as is its multiplicity as the root of the characteristic polynomial, matrix $\mathbf{X}$ is non-singular, so we have

$$\mathbf{A} = \mathbf{X\Lambda X}^{-1}. \tag{2.22}$$

Decomposition (2.22) is called the *spectral decomposition* or the *eigendecomposition* of $\mathbf{A}$.

Theorem 2.1.3. *[13, Theorem 7.4.1] For every $\mathbf{A} \in \mathbb{R}^{n\times n}$, there exists an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$ such that*

$$\mathbf{Q}^T\mathbf{A}\mathbf{Q} = \mathbf{T} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} & \cdots & \mathbf{T}_{1k} \\ & \mathbf{T}_{22} & \cdots & \mathbf{T}_{2k} \\ & & \ddots & \vdots \\ & & & \mathbf{T}_{kk}, \end{bmatrix} \tag{2.23}$$

where $\mathbf{T}$ is a block upper triangular with same eigenvalues as $\mathbf{A}$. Furthermore, its eigenvalues are the union of eigenvalues of its diagonal blocks $\mathbf{T}_{ii}$, which are either a 1×1 matrix containing a real eigenvalue or a 2×2 matrix with complex conjugate eigenvalues.

Decomposition (2.23) is known as the *real Schur decomposition.*

If we are, however, dealing with a symmetric matrix $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n\times n}$, then its spectrum will be a subset of $\mathbb{R}$. We can see this by taking an eigenvalue λ of $\mathbf{A}$ and its normalized eigenvector $\mathbf{x}$, $\|\mathbf{x}\|_2 = 1$. Now

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad \Rightarrow \quad \lambda = \mathbf{x}^*\mathbf{A}\mathbf{x} = \mathbf{x}^*\mathbf{A}^T\mathbf{x} = \overline{\mathbf{x}^*\mathbf{A}\mathbf{x}} = \overline{\lambda}.$$

It follows that the matrix $\mathbf{T}$ from the Schur decomposition (2.23) is upper triangular with eigenvalues on the diagonal and symmetric, since

$$\mathbf{T} = \mathbf{Q}^T\mathbf{A}\mathbf{Q} = \mathbf{Q}^T\mathbf{A}^T\mathbf{Q} = \mathbf{T}^T.$$

So the Schur decomposition of a symmetric matrix $\mathbf{A}$ is exactly its spectral decomposition (2.22) with $\mathbf{Q} = \mathbf{X}$ and $\mathbf{T} = diag(\lambda_1, \ldots, \lambda_n)$.

From this it follows that the eigenvectors of a symmetric matrix can be chosen to be real and mutually orthogonal, so the spectral decomposition of a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ can be written as

$$\mathbf{A} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^T, \tag{2.24}$$

where $\mathbf{X} = [\,\mathbf{x}_1 \cdots \mathbf{x}_n\,] \in \mathbb{R}^{n\times n}$ is orthogonal with eigenvectors of $\mathbf{A}$ as columns, and $\mathbf{\Lambda} = diag(\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$ has eigenvalues of $\mathbf{A}$ on the diagonal. This decomposition can be obtained in $\mathcal{O}(n^3)$ operations.

If $rank(\mathbf{A}) = r < n$, then $\mathbf{A}$ has exactly r nonzero eigenvalues, which follows from

$$rank(\mathbf{A}) = rank(\mathbf{X}\mathbf{\Lambda}\mathbf{X}^T) = rank(\mathbf{\Lambda}),$$

so we can eliminate the zero eigenvalues from $\mathbf{\Lambda}$ and the corresponding eigenvectors from $\mathbf{X}$, and have decomposition

$$\mathbf{A} = \mathbf{X}_r \mathbf{\Lambda}_r \mathbf{X}_r^T, \tag{2.25}$$

with $\mathbf{X}_r$ $n \times r$ orthonormal matrix and $\mathbf{\Lambda}_r$ $r \times r$ diagonal with nonzero diagonal elements.

Symmetric matrices $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ play a significant role because of their connection to the SVD (see Theorem 2.1.5) and it can easily be shown that they have non-negative eigenvalues and the following relation holds

$$\|\mathbf{A}^T\mathbf{A}\|_2 = \|\mathbf{A}\mathbf{A}^T\|_2 = \|\mathbf{A}\|_2^2, \tag{2.26}$$

which follows from the 2-norm characterization (2.21) and the fact that if λ is an eigenvalue of a matrix $\mathbf{A}$, then λ^2 is an eigenvalue of $\mathbf{A}^2$.

In the next two theorems we explain the connection between singular values and eigenvalues of a symmetric matrix.

Theorem 2.1.4. *[29, Theorem 5.5] If $\mathbf{A} = \mathbf{A}^T$, then the singular values of $\mathbf{A}$ are the absolute values of the eigenvalues of $\mathbf{A}$.*

Proof. As stated earlier, when dealing with a symmetric matrix, the spectral decomposition has the form (2.24) with $\mathbf{X}$ orthogonal. Now, we rewrite it as

$$\mathbf{A} = \mathbf{X}|\mathbf{\Lambda}|\text{sgn}(\mathbf{\Lambda})\mathbf{X}^T, \tag{2.27}$$

where $|\mathbf{\Lambda}|$ and $\text{sgn}(\mathbf{\Lambda})$ denote diagonal matrices whose entries are numbers $|\lambda_j|$ and $\text{sgn}(\lambda_j)$, respectively. Since $\text{sgn}(\mathbf{\Lambda})\mathbf{X}^T$ is also orthogonal, (2.27) is the SVD of $\mathbf{A}$, with singular values $|\lambda_j|$. □

Theorem 2.1.5. *[29, Theorem 5.4] The nonzero singular values of $\mathbf{A} \in \mathbb{R}^{m\times n}$ are the square roots of the nonzero eigenvalues of $\mathbf{A}^T\mathbf{A}$ or $\mathbf{A}\mathbf{A}^T$.*

Proof. Let $\mathbf{A}$ have SVD (2.7). Then

$$\mathbf{A}^T\mathbf{A} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T,$$
$$\mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T,$$

which are spectral decompositions of symmetric matrices $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ (see (2.24)) with eigenvalues on the diagonal of matrix $\mathbf{\Sigma}^2$. Since the nonzero diagonal elements of $\mathbf{\Sigma}$ are $\sigma_1 \geq \cdots \geq \sigma_r > 0$, with $r = rank(\mathbf{A})$, the nonzero diagonal elements of $\mathbf{\Sigma}^2$ are $\sigma_1^2, \ldots, \sigma_r^2$, from which it follows that the eigenvalues of matrices $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are exactly squares of singular values of $\mathbf{A}$. □

2.1.6 QR factorization

The *QR factorization* of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, with $m \geq n$, is given by

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q} \in \mathbb{R}^{m\times m}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{m\times n}$ is upper triangular.

Theorem 2.1.6. *[13, Theorem 5.2.1] If* $\mathbf{A} = \mathbf{QR}$ *is a QR factorization of a full rank matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$*, with* $m \geq n$*, and* $\mathbf{A} = [\,\mathbf{a}_1 \cdots \mathbf{a}_n\,]$ *and* $\mathbf{Q} = [\,\mathbf{q}_1 \cdots \mathbf{q}_m\,]$*, then*

$$span\{\mathbf{a}_1, \ldots, \mathbf{a}_k\} = span\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}, \quad for\ k = 1, 2 \ldots, n.$$

In particular, if we partition $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_n & \tilde{\mathbf{Q}}_n \end{bmatrix}$*, where* $\mathbf{Q}_n = \mathbf{Q}[:, 1:n\,]$*, then*

$$\mathcal{R}(\mathbf{A}) = \mathcal{R}(\mathbf{Q}_n)$$
$$\mathcal{R}(\mathbf{A})^{\perp} = \mathcal{R}(\tilde{\mathbf{Q}}_n)$$

and $\mathbf{A} = \mathbf{Q}_n\mathbf{R}_n$*, with* $\mathbf{R}_n = \mathbf{R}[\,1:n\,, :\,]$.

The factorization $\mathbf{A} = \mathbf{Q}_n\mathbf{R}_n$ is called the *truncated QR factorization* and it can be computed in $\mathcal{O}(mn^2)$ operations.

It holds $rank(\mathbf{A}) = rank(\mathbf{R}_n)$ and if $\mathbf{A}$ has full rank, then setting the diagonal elements of $\mathbf{R}_n$ to be positive makes the truncated factorization uniquely determined.

2.2 Matrix products

Apart from the standard matrix product, several other matrix products play a significant role when working with tensors. In the following we present their definitions and properties.

- *Hadamard (element-wise) product*: Given $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$,

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

- *Kronecker product*: Given $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{p\times q}$,

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{(mp)\times(nq)}.$$

- *Khatri-Rao product*: Given $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{p\times n}$

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{b}_n \end{bmatrix} \in \mathbb{R}^{(mp)\times n},$$

  where $\mathbf{a}_j$ and $\mathbf{b}_j$ denote the jth columns of $\mathbf{A}$ and $\mathbf{B}$, respectively.

- *Transpose Khatri-Rao product*: Given $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{m\times p}$,

$$\mathbf{A} \odot^T \mathbf{B} = \left(\mathbf{A}^T \odot \mathbf{B}^T\right)^T = \begin{bmatrix} \mathbf{a}_1^T \otimes \mathbf{b}_1^T \\ \mathbf{a}_2^T \otimes \mathbf{b}_2^T \\ \vdots \\ \mathbf{a}_m^T \otimes \mathbf{b}_m^T \end{bmatrix} \in \mathbb{R}^{m\times(np)}, \tag{2.28}$$

  where $\mathbf{a}_i^T$ and $\mathbf{b}_i^T$ denote the ith rows of $\mathbf{A}$ and $\mathbf{B}$, respectively.

Theorem 2.2.1. *For matrices and vectors of appropriate size, the following properties hold:*

(1) $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$,

(2) $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$,

(3) $\mathbf{A}$ *and* $\mathbf{B}$ *orthogonal* $\Rightarrow$ $\mathbf{A} \otimes \mathbf{B}$ *orthogonal,*

(4) $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \odot \mathbf{D}) = \mathbf{AC} \odot \mathbf{BD}$,

(5) $(\mathbf{A} \otimes \mathbf{B})\mathbf{v} = vec(\mathbf{BVA}^T), \quad \mathbf{v} = vec(\mathbf{V})$,

(6) $(\mathbf{A} \odot \mathbf{B})\mathbf{v} = vec(\mathbf{B}\ diag(\mathbf{v})\mathbf{A}^T)$,

(7) $(\mathbf{A} \odot^T \mathbf{B})\mathbf{v} = diag(\mathbf{BVA}^T), \quad \mathbf{v} = vec(\mathbf{V})$.

Here, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n\times n}$, $diag(\mathbf{A}) = [\, a_{11}\ a_{22}\ \cdots\ a_{nn}\,]^T \in \mathbb{R}^n$, *while for a vector* $\mathbf{v} = (v_i) \in \mathbb{R}^n$, $diag(\mathbf{v})$ *denotes diagonal* $n \times n$ *matrix with elements* $v_1, v_2 \ldots, v_n$ *on the diagonal. Vectorized matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *is a vector of size* mn, *denoted as* $vec(\mathbf{A})$, *obtained by stacking the columns of the matrix on top of one another.*

Proof. (1) Directly follows from the definition.

(2) For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$, $\mathbf{C} \in \mathbb{R}^{n\times r}$, $\mathbf{D} \in \mathbb{R}^{q\times s}$,

$$
\begin{aligned}
(\underbrace{\mathbf{A}\otimes\mathbf{B}}_{mp\times nq})(\underbrace{\mathbf{C}\otimes\mathbf{D}}_{nq\times rs}) &= \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \begin{bmatrix} c_{11}\mathbf{D} & \cdots & c_{1r}\mathbf{D} \\ \vdots & \ddots & \vdots \\ c_{n1}\mathbf{D} & \cdots & c_{nr}\mathbf{D} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{k=1}^{n} a_{1k}c_{k1}\mathbf{BD} & \cdots & \sum_{k=1}^{n} a_{1k}c_{kr}\mathbf{BD} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{mk}c_{k1}\mathbf{BD} & \cdots & \sum_{k=1}^{n} a_{mk}c_{kr}\mathbf{BD} \end{bmatrix} \\
&= \mathbf{AC}\otimes\mathbf{BD} \in \mathbb{R}^{mp\times rs}.
\end{aligned}
$$

(3) For $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$ orthogonal, from (2.2.1.(1)) and (2.2.1.(2)) follows

$$
\begin{aligned}
(\mathbf{A}\otimes\mathbf{B})^T(\mathbf{A}\otimes\mathbf{B}) &= (\mathbf{A}^T\otimes\mathbf{B}^T)(\mathbf{A}\otimes\mathbf{B}) = \mathbf{A}^T\mathbf{A}\otimes\mathbf{B}^T\mathbf{B} = \mathbf{I}_m\otimes\mathbf{I}_n \\
&= \mathbf{A}\mathbf{A}^T\otimes\mathbf{B}\mathbf{B}^T = (\mathbf{A}\otimes\mathbf{B})(\mathbf{A}^T\otimes\mathbf{B}^T) = (\mathbf{A}\otimes\mathbf{B})(\mathbf{A}\otimes\mathbf{B})^T,
\end{aligned}
$$

where $\mathbf{I}_m$ and $\mathbf{I}_n$ denote identity matrices of order m and n, respectively. From the definition of Kronecker product, obviously $\mathbf{I}_m \otimes \mathbf{I}_n = \mathbf{I}_{mn}$.

(4) For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$, $\mathbf{C} \in \mathbb{R}^{n\times r}$, $\mathbf{D} \in \mathbb{R}^{q\times r}$, from (2.2.1.(2))

$$
\begin{aligned}
(\underbrace{\mathbf{A}\otimes\mathbf{B}}_{mp\times nq})(\underbrace{\mathbf{C}\odot\mathbf{D}}_{nq\times r}) &= (\mathbf{A}\otimes\mathbf{B}) \begin{bmatrix} \mathbf{c}_1\otimes\mathbf{d}_1 & \cdots & \mathbf{c}_r\otimes\mathbf{d}_r \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{A}\otimes\mathbf{B})(\mathbf{c}_1\otimes\mathbf{d}_1) & \cdots & (\mathbf{A}\otimes\mathbf{B})(\mathbf{c}_r\otimes\mathbf{d}_r) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{A}\mathbf{c}_1\otimes\mathbf{B}\mathbf{d}_1 & \cdots & \mathbf{A}\mathbf{c}_r\otimes\mathbf{B}\mathbf{d}_r \end{bmatrix} \\
&= \mathbf{AC}\odot\mathbf{BD}.
\end{aligned}
$$

(5) For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$ and $\mathbf{v} \in \mathbb{R}^{nq}$,

$$
\begin{aligned}
(\mathbf{A}\otimes\mathbf{B})\mathbf{v} &= \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_{nq} \end{bmatrix} \\
&= \begin{bmatrix} a_{11}\mathbf{B}\mathbf{v}_{1:q} + a_{12}\mathbf{B}\mathbf{v}_{q+1:2q} + \cdots + a_{1n}\mathbf{B}\mathbf{v}_{(n-1)q:nq} \\ \vdots \\ a_{m1}\mathbf{B}\mathbf{v}_{1:q} + a_{m2}\mathbf{B}\mathbf{v}_{q+1:2q} + \cdots + a_{mn}\mathbf{B}\mathbf{v}_{(n-1)q:nq} \end{bmatrix} \\
&= vec\left(\begin{bmatrix} \mathbf{B}\mathbf{v}_{1:q} & \mathbf{B}\mathbf{v}_{q+1:2q} & \cdots & \mathbf{B}\mathbf{v}_{(n-1)q:nq} \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix} \right)
\end{aligned}
$$

$$= vec\left(\mathbf{B}\underbrace{\begin{bmatrix}\mathbf{v}_{1:q} & \mathbf{v}_{q+1:2q} & \cdots & \mathbf{v}_{(n-1)q:nq}\end{bmatrix}}_{\mathbf{V}}\mathbf{A}^T\right).$$

(6) For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, using (2.2.1.(5)),

$$\begin{aligned}(\mathbf{A}\odot\mathbf{B})\mathbf{v} &= \begin{bmatrix}\mathbf{a}_1\otimes\mathbf{b}_1 & \mathbf{a}_2\otimes\mathbf{b}_2 & \cdots & \mathbf{a}_n\otimes\mathbf{b}_n\end{bmatrix}\begin{bmatrix}v_1\\ \vdots\\ v_n\end{bmatrix}\\ &= (\mathbf{a}_1\otimes\mathbf{b}_1)v_1 + (\mathbf{a}_2\otimes\mathbf{b}_2)v_2 + \cdots + (\mathbf{a}_n\otimes\mathbf{b}_n)v_n\\ &= vec(\mathbf{b}_1v_1\mathbf{a}_1^T) + vec(\mathbf{b}_2v_2\mathbf{a}_2^T) + \cdots + vec(\mathbf{b}_nv_n\mathbf{a}_n^T)\\ &= vec\left(\mathbf{b}_1v_1\mathbf{a}_1^T + \mathbf{b}_2v_2\mathbf{a}_2^T + \cdots + \mathbf{b}_nv_n\mathbf{a}_n^T\right)\\ &= vec\left(\begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_n\end{bmatrix}\begin{bmatrix}v_1 & & & \\ & v_2 & & \\ & & \ddots & \\ & & & v_n\end{bmatrix}\begin{bmatrix}\mathbf{a}_1^T\\ \mathbf{a}_2^T\\ \vdots\\ \mathbf{a}_n^T\end{bmatrix}\right).\end{aligned}$$

(7) For $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{m\times p}$ and $\mathbf{v} \in \mathbb{R}^{nq}$, using (2.2.1.(5)),

$$\begin{aligned}(\mathbf{A}\odot^T\mathbf{B})\mathbf{v} &= \begin{bmatrix}\mathbf{a}_1^T\otimes\mathbf{b}_1^T\\ \vdots\\ \mathbf{a}_m^T\otimes\mathbf{b}_m^T\end{bmatrix}\mathbf{v} = \begin{bmatrix}(\mathbf{a}_1^T\otimes\mathbf{b}_1^T)\mathbf{v}\\ \vdots\\ (\mathbf{a}_m^T\otimes\mathbf{b}_m^T)\mathbf{v}\end{bmatrix} = \begin{bmatrix}vec\left(\mathbf{b}_1^T\mathbf{V}\mathbf{a}_1\right)\\ \vdots\\ vec\left(\mathbf{b}_m^T\mathbf{V}\mathbf{a}_m\right)\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{b}_1^T\mathbf{V}\mathbf{a}_1\\ \vdots\\ \mathbf{b}_m^T\mathbf{V}\mathbf{a}_m\end{bmatrix} = diag\left(\begin{bmatrix}\mathbf{b}_1^T\\ \vdots\\ \mathbf{b}_m^T\end{bmatrix}\mathbf{V}\begin{bmatrix}\mathbf{a}_1 & \cdots & \mathbf{a}_m\end{bmatrix}\right).\end{aligned}$$

□

2.2.1 Complexity reduction using products properties

Properties (2.2.1.(5))– (2.2.1.(7)) enable efficient computation of a matrix-vector product, when the matrix is given as Kronecker, Khatri-Rao or Transpose Khatri-Rao product of two matrices. We have implemented these products in Julia, creating functions `krontv`, `krtv` and `tkrtv`, which can also be used to perform multiplication of the products by a matrix, in which case the multiplication is done column by column. In the following we

present details of the functions and their usage, and compare them with the direct way of computing these multiplications - by first forming the product and then multiplying it by a vector or a matrix.

The computational environment is as presented in Section 1.4. For numerical experiments, we generate matrices and vectors from the standard normal distribution. In the case when we multiply the products by a tall matrix, using our functions still shows to work faster than the direct approach. The results are presented in Figures 2.1, 2.2 and 2.3.

Kronecker product. For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$ and vector $\mathbf{v} \in \mathbb{R}^{nq}$, forming the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ requires $mnpq$ multiplications, so the direct way of computing $(\mathbf{A} \otimes \mathbf{B})\mathbf{v}$ requires $\mathcal{O}(mnpq)$ operations. Using the property (2.2.1.(5)), instead of forming the Kronecker product, we reshape vector $\mathbf{v}$ into $q \times n$ matrix $\mathbf{V}$ and perform matrix-matrix multiplication

$$\underbrace{\mathbf{B}}_{p\times q}\,\mathbf{V}\,\underbrace{\mathbf{A}^T}_{n\times m},$$

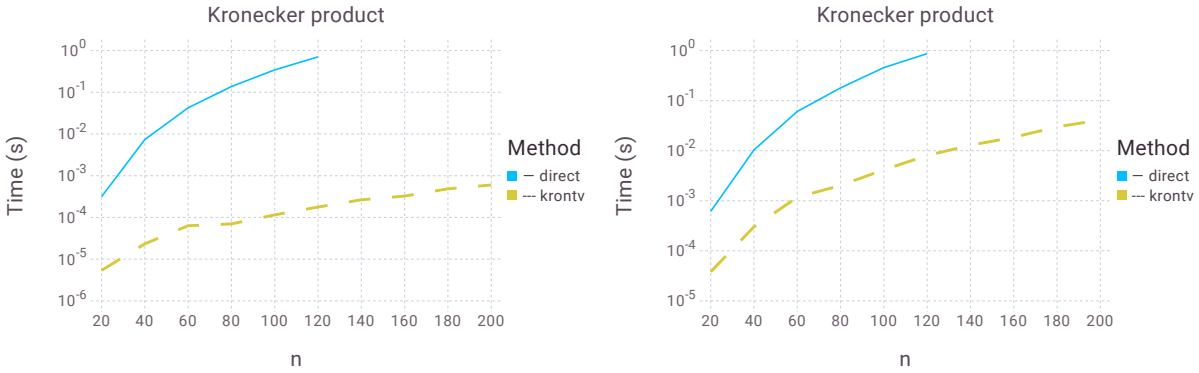
which reduces the number of operations to $\mathcal{O}(\min\{np(m+q), mq(n+p)\})$. Also, this reduces memory requirements from $\mathcal{O}(mnpq)$ to $\mathcal{O}(mp + \min\{np, mq\})$.

In Julia, function `krontv` performs this multiplication and it also works with any matrix $\mathbf{V} \in \mathbb{R}^{nq\times k}$, $k \in \mathbb{N}$; see Figure 2.1.

```
w=krontv(A,B,v)
W=krontv(A,B,V)
```

Khatri-Rao product. For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times n}$ and vector $\mathbf{v} \in \mathbb{R}^{n}$, forming the Khatri-Rao product $\mathbf{A} \odot \mathbf{B}$ requires mnp multiplications, so the direct way of computing $(\mathbf{A} \odot \mathbf{B})\mathbf{v}$ requires $\mathcal{O}(mnp)$ operations. Using the property (2.2.1.(6)), instead of forming the Khatri-Rao product, we perform the multiplication on the right-hand side by creating $n \times p$ matrix whose every column is exactly vector $\mathbf{v}$ and performing Hadamard product instead of the regular multiplication, in one of the two following ways

$$\mathbf{B}\; diag(\mathbf{v})\mathbf{A}^T = \underbrace{\mathbf{B}}_{p\times n}\Big(\underbrace{[\,v \mid \cdots \mid v\,] * \mathbf{A}^T}_{n\times m}\Big) = \underbrace{\Big(\mathbf{B} * \begin{bmatrix} v^T \\ \vdots \\ v^T \end{bmatrix}\Big)}_{p\times n}\underbrace{\mathbf{A}^T}_{n\times m},$$



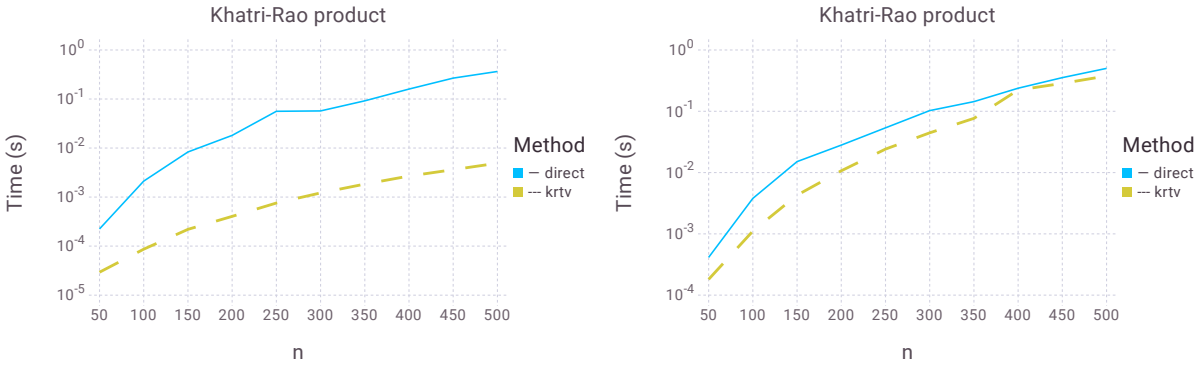
(a) Multiplication by a vector. (b) Multiplication by a matrix.

Figure 2.1: Execution times (in seconds) for calculating multiplication of Kronecker products of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, by a vector $\mathbf{v} \in \mathbb{R}^{n^2}$ and a matrix $\mathbf{V} \in \mathbb{R}^{n^2\times k}$, with $k = \frac{n}{4}$, directly (forming the product) and by function `krontv`. When $n \geq 140$ forming the product becomes too expensive.

ending up with $\mathcal{O}(mnp)$ operations. Even though the computational complexity is the same, direct approach requires formation of $mp \times n$ product matrix so by applying property (2.2.1.(6)) we reduce memory requirements from $\mathcal{O}(mnp)$ to $\mathcal{O}(mp + n \cdot \min\{m, p\})$.

Function `krtv` performs this multiplication in Julia and we can also multiply by any matrix $\mathbf{V} \in \mathbb{R}^{n\times k}$, $k \in \mathbb{N}$; see Figure 2.2.

```
w=krtv(A,B,v)
W=krtv(A,B,V)
```



(a) Multiplication by a vector. (b) Multiplication by a matrix.

Figure 2.2: Execution times (in seconds) for calculating multiplication of Khatri-Rao product of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, by a vector $\mathbf{v} \in \mathbb{R}^{n}$ and a matrix $\mathbf{V} \in \mathbb{R}^{n\times k}$, with $k = \frac{n}{10}$, directly (forming the product) and by function `krtv`.

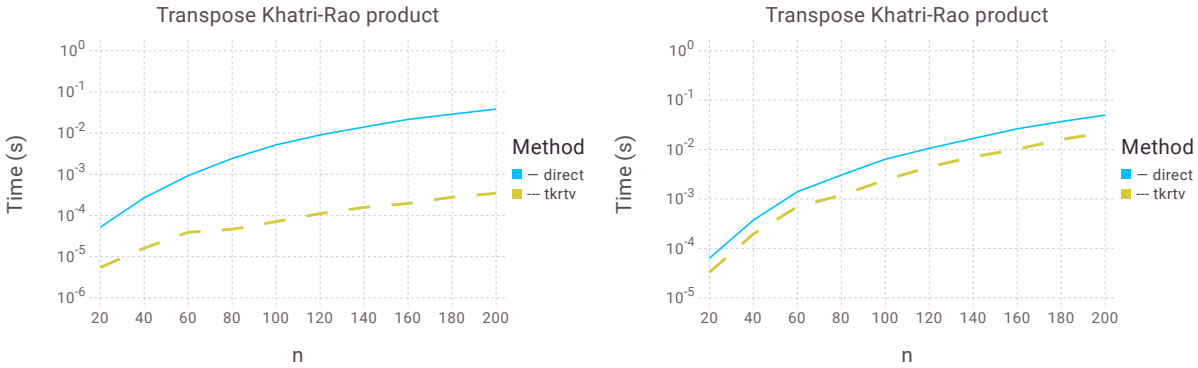
Transpose Khatri-Rao product. For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{m\times p}$ and vector $\mathbf{v} \in \mathbb{R}^{np}$, forming the Transpose Khatri-Rao product $\mathbf{A} \odot^T \mathbf{B}$ requires mnp multiplications, so the direct way of computing $(\mathbf{A} \odot \mathbf{B})\mathbf{v}$ requires $\mathcal{O}(mnp)$ operations. Using the property (2.2.1.(7)), instead of forming the Transpose Khatri-Rao product, we reshape vector $\mathbf{v}$ into a $p \times n$ matrix $\mathbf{V}$ and only calculate diagonal elements of $\mathbf{BVA}^T$ in one of the two following ways

$$diag(\mathbf{BVA}^T) = \sum_{j=1}^{n} \left[(\underbrace{\mathbf{B}}_{m\times p} \underbrace{\mathbf{V}}_{p\times n}) * \mathbf{A} \right]_{:j} = \sum_{j=1}^{p} \left[\mathbf{B} * (\underbrace{\mathbf{A}}_{m\times n} \underbrace{\mathbf{V}^T}_{n\times p}) \right]_{:j},$$

i.e. by summing the columns of the Hadamard products. This way the multiplication is done in $\mathcal{O}(mnp)$ operations. Again, as with Khatri-Rao product, property (2.2.1.(7)) did not improve the computational complexity, but did improve memory requirements. Forming the Transpose Khatri-Rao product means storing $m \times np$ matrix, while by exploiting property (2.2.1.(7)) we only have to store an $m \times n$ or $m \times p$ matrix. So all together we have reduced memory requirements form $\mathcal{O}(mnp)$ to $\mathcal{O}(m \cdot \min\{n, p\})$.

In Julia, we use function `tkrtv` and it also works with any matrix $\mathbf{V} \in \mathbb{R}^{np\times k}$, $k \in \mathbb{N}$; see Figure 2.3.

```
w=krtv(A,B,v)
W=krtv(A,B,V)
```



(a) Multiplication by a vector.

(b) Multiplication by a matrix.

Figure 2.3: Execution times (in seconds) for calculating multiplication of Transpose Khatri-Rao product of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, by a vector $\mathbf{v} \in \mathbb{R}^{n^2}$ and a matrix $\mathbf{V} \in \mathbb{R}^{n^2\times k}$, with $k = \frac{n}{4}$, directly (forming the product) and by function `tkrtv`.

2.3 Iterative methods for low-rank matrix approximation

As we will explain in Section 3.3, dealing with tensors in Tucker format requires approximating a dominant low-dimensional subspace for a column space of a matrix $\mathbf{Z} \in \mathbb{R}^{n \times p}$, which is usually wide, with $n \ll p$. This can be achieved by considering the $n \times n$ Gramian $\mathbf{A} = \mathbf{Z}\mathbf{Z}^T$ and aiming at a low-rank approximation of the form

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{2.29}$$

where matrix $\mathbf{U} \in \mathbb{R}^{n \times r}$, with $r < n$, is a basis of the desired subspace and $\mathbf{\Lambda} \approx \mathbf{U}^T\mathbf{A}\mathbf{U}$.

Moreover, dealing with Hadamard product of tensors in Tucker format will result in matrix $\mathbf{Z}$ having a special structure (4.3) and that, as explained in [29], can be exploited by iterative methods, which use matrix in the form of a *black box*:

$$\mathbf{x} \longrightarrow \boxed{\begin{matrix}\text{BLACK} \\ \text{BOX}\end{matrix}} \longrightarrow \mathbf{A}\mathbf{x}.$$

The iterative algorithm requires nothing more then the ability to determine $\mathbf{A}\mathbf{x}$ for any $\mathbf{x}$, and in this section we present two such algorithms for creating approximation (2.29) - *Lanczos method* and *randomized algorithm*.

2.3.1 Lanczos method

Lanczos method is based on the idea of projecting an n-dimensional problem into a lower-dimensional Krylov subspace. It is a special case of Arnoldi method, when the matrix we are dealing with is hermitian or real symmetric, which is the case we are interested in. Throughout this section we follow [28] and [2].

Description

For a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a vector $\mathbf{x} \in \mathbb{R}^n$, the associated *Krylov sequence* is the set of vectors $\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \mathbf{A}^3\mathbf{x}, \ldots$, which can be computed by the black box in the form $\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}(\mathbf{A}\mathbf{x}), \mathbf{A}(\mathbf{A}(\mathbf{A}\mathbf{x})), \ldots$. The corresponding *Krylov subspaces* are the spaces spanned by successively larger group of these vectors

$$\mathcal{K}_j(\mathbf{A}, \mathbf{x}) = \operatorname{span}\left\{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \ldots, \mathbf{A}^{j-1}\mathbf{x}\right\}.$$

When $\mathbf{A}$ and $\mathbf{x}$ are known from the context we will simply write $\mathcal{K}_j$.

Since $\mathcal{K}_j$ is a subspace of $\mathbb{R}^n$, $\dim(\mathcal{K}_j) \leq n$. The sequence of Krylov subspaces satisfies

$$\mathcal{K}_j(\mathbf{A}, \mathbf{x}) \subseteq \mathcal{K}_{j+1}(\mathbf{A}, \mathbf{x}) \quad \text{and} \quad \mathbf{A}\mathcal{K}_j(\mathbf{A}, \mathbf{x}) \subseteq \mathcal{K}_{j+1}(\mathbf{A}, \mathbf{x}).$$

If $\mathbf{x}$ is an eigenvector of $\mathbf{A}$ corresponding to eigenvalue λ, then $\mathbf{A}^j\mathbf{x} = \lambda^j\mathbf{x}$, and

$$\mathcal{K}_j(\mathbf{A}, \mathbf{x}) = \mathcal{K}_1(\mathbf{A}, \mathbf{x}), \quad j = 1, 2, \ldots.$$

In other words, the Krylov sequence terminates in the sense that vectors $\mathbf{A}^j\mathbf{x}$ after the first one provide no new information. More generally, we say that a Krylov sequence *terminates* at k if k is the smallest integer such that

$$\mathcal{K}_k(\mathbf{A}, \mathbf{x}) = \mathcal{K}_{k+1}(\mathbf{A}, \mathbf{x}), \tag{2.30}$$

and in that case $\dim(\mathcal{K}_k) = \dim(\mathcal{K}_{k+1}) = k$ and $\mathcal{K}_k$ is an invariant subspace of $\mathbf{A}$[1].

Lanczos method constructs an orthonormal basis of the invariant Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{x})$ for a given symmetric matrix $\mathbf{A}$ and a randomly generated vector $\mathbf{x}$. The randomness assures $k = n$ if $\mathbf{A}$ is of full rank, or $k = r + 1$ if $rank(\mathbf{A}) = r < n$.

The natural basis of $\mathcal{K}_k$ is evidently $\{\mathbf{x}, \mathbf{A}\mathbf{x}, \mathbf{A}^2\mathbf{x}, \ldots, \mathbf{A}^{k-1}\mathbf{x}\}$, but since the vectors $\mathbf{A}^j\mathbf{x}$ converge to the direction of the eigenvector corresponding to the largest eigenvalue (in modulus) of $\mathbf{A}$, this basis tends to be badly conditioned with increasing dimension k. Therefore, the Gram-Schmidt orthogonalization (see [13, Section 5.2.7]) process is applied to the basis vectors.

Suppose that $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j\} \subset \mathbb{R}^n$ is the orthonormal basis for $\mathcal{K}_j$, where $j \leq k$. We construct the vector $\mathbf{q}_{j+1}$ by first orthogonalizing $\mathbf{A}^j\mathbf{x}$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$,

$$\mathbf{r}_j = \mathbf{A}^j\mathbf{x} - \sum_{i=1}^{j} \mathbf{q}_i\mathbf{q}_i^T\mathbf{A}^j\mathbf{x}, \tag{2.31}$$

and then normalizing the resulting vector,

$$\mathbf{q}_{j+1} = \frac{\mathbf{r}_j}{\|\mathbf{r}_j\|_2}. \tag{2.32}$$

Then $\{\mathbf{q}_1, \ldots, \mathbf{q}_j, \mathbf{q}_{j+1}\}$ is an orthonormal basis of $\mathcal{K}_{j+1}$, which is called the *Lanczos basis* and vectors $\mathbf{q}_i$ the *Lanczos vectors*.

[1] $\mathcal{S}$ is an invariant subspace of $\mathbf{A}$ if $\mathbf{A}\mathcal{S} \subseteq \mathcal{S}$.

However, since $\mathbf{q}_1 = \mathbf{x}/\|\mathbf{x}\|_2$ and each $\mathbf{A}\mathbf{q}_i$ can be written as linear combination of vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{i+1}$, which follows from (2.31) and (2.32), we have

$$\begin{aligned}
\mathcal{K}_{j+1} &= \operatorname{span}\left\{\mathbf{x}, \mathbf{A}\mathbf{x}, \ldots, \mathbf{A}^j\mathbf{x}\right\} \\
&= \operatorname{span}\left\{\mathbf{q}_1, \mathbf{A}\mathbf{q}_1, \ldots, \mathbf{A}^j\mathbf{q}_1\right\} \\
&= \operatorname{span}\left\{\mathbf{q}_1, \alpha\mathbf{q}_1 + \beta\mathbf{q}_2, \mathbf{A}(\alpha\mathbf{q}_1 + \beta\mathbf{q}_2), \ldots, \mathbf{A}^{j-1}(\alpha\mathbf{q}_1 + \beta\mathbf{q}_2)\right\} \\
&= \operatorname{span}\left\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{A}\mathbf{q}_2, \ldots, \mathbf{A}^{j-1}\mathbf{q}_2\right\} \\
&= \cdots \\
&= \operatorname{span}\left\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j, \mathbf{A}\mathbf{q}_j\right\},
\end{aligned}$$

so instead of orthogonalizing $\mathbf{A}^j\mathbf{x}$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$, we can compute $\mathbf{q}_{j+1}$ more economically by orthogonalizing $\mathbf{A}\mathbf{q}_j$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$.

Now, instead of (2.31), the component $\mathbf{r}_j$ of $\mathbf{A}\mathbf{q}_j$ orthogonal to $\mathbf{q}_1, \ldots, \mathbf{q}_j$ is given by

$$\mathbf{r}_j = \mathbf{A}\mathbf{q}_j - \sum_{i=1}^{j} \mathbf{q}_i \left(\mathbf{q}_i^T \mathbf{A}\mathbf{q}_j\right). \tag{2.33}$$

If $\mathbf{r}_j = 0$, then the procedure stops, which means that we have found an invariant subspace, namely $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_j\}$. If $\|\mathbf{r}_j\|_2 > 0$ we obtain $\mathbf{q}_{j+1}$ again by (2.32).

From (2.33), (2.32) and the fact that $\mathbf{q}_{j+1}$ is orthogonal to all $\mathbf{q}_1, \ldots, \mathbf{q}_j$, we get

$$\mathbf{q}_{j+1}^T \mathbf{r}_j = \|\mathbf{r}_j\|_2 = \mathbf{q}_{j+1}^T \mathbf{A}\mathbf{q}_j.$$

Setting $t_{ij} = \mathbf{q}_i^T \mathbf{A}\mathbf{q}_j$, (2.33) can be written as

$$\mathbf{A}\mathbf{q}_j = \sum_{i=1}^{j+1} t_{ij}\mathbf{q}_i.$$

For $j = 1, 2, \ldots, k$, in the matrix form, we have

$$\mathbf{A}\left[\,\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k\,\right] = \left[\,\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k\ \mathbf{q}_{k+1}\,\right] \begin{bmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1k} \\ t_{21} & t_{22} & t_{23} & \cdots & t_{2k} \\ & t_{32} & t_{33} & \cdots & t_{3k} \\ & & \ddots & \ddots & \vdots \\ & & & t_{k,k-1} & t_{kk} \\ & & & & t_{k+1,k} \end{bmatrix}. \tag{2.34}$$

Since we are only interested in the case when $\mathbf{A}$ is symmetric, denoting

$$\mathbf{Q}_k = [\,\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k\,], \quad \mathbf{T}_k = \begin{bmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1k} \\ t_{21} & t_{22} & t_{23} & \cdots & t_{2k} \\ & t_{32} & t_{33} & \cdots & t_{3k} \\ & & \ddots & \ddots & \vdots \\ & & & t_{k,k-1} & t_{kk} \end{bmatrix},$$

and multiplying (2.34) with $\mathbf{Q}_k^T$ from the left, we get

$$\mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k = \mathbf{T}_k. \tag{2.35}$$

From here we have that matrix $\mathbf{T}_k$ is also symmetric, which makes it tridiagonal, i.e $t_{ij} = 0$, whenever $j > i + 1$. Moreover, setting $\alpha_j = t_{jj}$ and $\beta_j = t_{j,j+1}$, equation (2.33) simplifies to

$$\mathbf{r}_j = \mathbf{A}\mathbf{q}_j - \mathbf{q}_j \underbrace{\left(\mathbf{q}_j^T \mathbf{A} \mathbf{q}_j\right)}_{\alpha_j} - \mathbf{q}_{j-1} \underbrace{\left(\mathbf{q}_{j-1}^T \mathbf{A} \mathbf{q}_j\right)}_{\beta_{j-1}} = \mathbf{A}\mathbf{q}_j - \alpha_j \mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}. \tag{2.36}$$

Therefore,

$$\mathbf{r}_j = \beta_j \mathbf{q}_{j+1}, \quad \beta_j = \|\mathbf{r}_j\|_2.$$

This, together with (2.36) and setting $\beta_0 = 0$, yields a three term recurrence

$$\mathbf{A}\mathbf{q}_j = \beta_{j-1}\mathbf{q}_{j-1} + \alpha_j \mathbf{q}_j + \beta_j \mathbf{q}_{j+1},$$

or in the matrix form

$$\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_k \underbrace{\begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}}_{\mathbf{T}_k} + \beta_k [\,\mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{q}_{k+1}\,]. \tag{2.37}$$

The described procedure is called the *Lanczos tridiagonalization*. However, running it in floating-point arithmetics can cause Lanczos vectors $\mathbf{q}_j$ to lose their mutual orthogonality, so reorthogonalization has to be done in each iteration. This loss of orthogonality happens, as explained in [13], because if $\beta_j \leftarrow \|\mathbf{r}_j\|_2$ and $\mathbf{q}_{j+1} \leftarrow \mathbf{r}_j/\beta_j$ are computed in

floating point arithmetics, then $\beta_j \mathbf{q}_{j+1} \approx \mathbf{r}_j + \mathbf{w}_j$, where $\|\mathbf{w}_j\|_2 \approx \epsilon_m \|\mathbf{r}_j\|_2 \approx \epsilon_m \|\mathbf{A}\|_2$, with ϵ_m denoting machine epsilon. From this it follows

$$|\mathbf{q}_{j+1}^T \mathbf{q}_i| \approx \frac{|\mathbf{r}_j^T \mathbf{q}_i| + \epsilon_m \|\mathbf{A}\|_2}{|\beta_j|}, \quad i = 1, \ldots, j.$$

So when β_j is small, significant departures from orthogonality can be expected and this is cured by reorthogonalizing $\mathbf{r}_j$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$ before calculating $\mathbf{q}_{j+1}$ in each step, i.e. rewriting $\mathbf{r}_j$ as

$$\mathbf{r}_j = \mathbf{r}_j - \sum_{i=1}^{j} \mathbf{q}_i \mathbf{q}_i^T \mathbf{r}_j.$$

The *Lanczos tridiagonalization with full reorthogonalization* is presented in Algorithm 1.

There are other methods for reorthogonalization on line 13, such as semiorthogonal methods (see [28, Section 5.3.3]), which can lower the overall complexity of the algorithm in some cases, for example when the involved matrix is very large, but the matrix-vector multiplication on line 10 can be cheaply performed. However, the presented full orthogonalization fits our needs and we would not significantly benefit from other reorthogonalization methods.

Algorithm 1 stops once k satisfies (2.30) and then, from (2.37), $\beta_k = 0$ implies

$$\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_k \mathbf{T}_k.$$

For simplicity we remove the index and denote $\mathbf{Q} = \mathbf{Q}_k$ and $\mathbf{T} = \mathbf{T}_k$ as the output of Algorithm 1.

If $\mathbf{A}$ has full rank, then $k = n$ and $\mathbf{Q}$ is $n \times n$ orthogonal and $\mathbf{T}$ $n \times n$ symmetric tridiagonal, which gives $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$. Getting the spectral decomposition (2.24) of $\mathbf{T}$,

$$\mathbf{T} = \hat{\mathbf{U}} \mathbf{\Lambda} \hat{\mathbf{U}}^T, \tag{2.38}$$

with $\hat{\mathbf{U}}$ orthogonal and $\mathbf{\Lambda}$ diagonal, both or order n, and setting

$$\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}, \tag{2.39}$$

we have

$$\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T = \mathbf{Q}\hat{\mathbf{U}}\mathbf{\Lambda}\hat{\mathbf{U}}^T\mathbf{Q}^T = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T,$$

with $\mathbf{U}$ $n \times n$ orthogonal matrix whose columns are eigenvectors of $\mathbf{A}$ and $\mathbf{\Lambda}$ with eigenvalues on the diagonal.

Algorithm 1 Lanczos tridiagonalization algorithm with reorthogonalization

Given a symmetric matrix $\mathbf{A}$ *of order* n*, a tolerance* ϵ*, a maximal number of iterations* maxit *and optionally requested rank* r *and oversampling parameter* p*, the following procedure computes an orthonormal matrix* $\mathbf{Q}$ *and tridiagonal* $\mathbf{T}$ *such that* $\mathbf{A} \approx \mathbf{Q}\mathbf{T}\mathbf{Q}^T$*.*

procedure LANCZOS_TRIDIAG($\mathbf{A}$, $\epsilon = 10^{-8}$, maxit $= 1000$, $r = 0$, $p = 10$)
  maxit $= \min\{n, \text{maxit}\}$
  if $r \neq 0$ **then**
    maxit $= \min\{\text{maxit}, r + p\}$
  end if
  Choose a vector $\mathbf{x}$ of length n
  $\mathbf{q} = \mathbf{x}/\|\mathbf{x}\|_2$
  $\mathbf{Q} = [\,\mathbf{q}\,]$
  for $k = 1, 2, \ldots,$ maxit **do**
    $\mathbf{r} = \mathbf{A}\mathbf{q}$
    $\alpha_k = \mathbf{q}^T\mathbf{r}$
    $\mathbf{r} = \mathbf{r} - \alpha_k\mathbf{q}$
    Reorthogonalize $\mathbf{r} = \mathbf{r} - \mathbf{Q}(\mathbf{Q}^T\mathbf{r})$.
    Set $\beta_k = \|\mathbf{r}\|_2$ and compute ω_k according to (2.41).
    if $\omega_k < \epsilon$ **then**
      break
    end if
    $\mathbf{q} = \mathbf{r}/\beta_k$
    $\mathbf{Q} = [\,\mathbf{Q} \quad \mathbf{q}\,]$
  end for
  $\mathbf{T} = \text{tridiag}((\alpha_1, \ldots, \alpha_k), (\beta_1, \ldots, \beta_{k-1}))$
end procedure

On the other hand, if $rank(\mathbf{A}) = r < n$, encountering a zero β_k signals the computation of an exact subspace, in which case $\mathbf{Q}$ is $n \times k$ orthogonal and $\mathbf{T}$ $k \times k$ symmetric tridiagonal. From spectral decomposition (2.38) of $\mathbf{T}$ we have

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{T} = \mathbf{Q}\hat{\mathbf{U}}\mathbf{\Lambda}\hat{\mathbf{U}}^T \quad \Rightarrow \quad \mathbf{A}(\mathbf{Q}\hat{\mathbf{U}}) = (\mathbf{Q}\hat{\mathbf{U}})\mathbf{\Lambda}, \tag{2.40}$$

therefore $\mathbf{\Lambda}$ contains eigenvalues of $\mathbf{A}$ and it has exactly r nonzero elements, so we can

have (2.38) with $\hat{\mathbf{U}}$ $k \times r$ orthonormal and $\mathbf{\Lambda}$ $r \times r$ diagonal; see (2.25). Now $k \times r$ orthonormal $\mathbf{U}$ from (2.39) contains eigenvectors of $\mathbf{A}$, so $\mathbf{A}$ admits spectral decomposition (2.25), i.e.,

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T,$$

which also gives $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$.

However, an exact zero or even a small β_k is a rarity in practice, so we usually have $k \geq r + 1$. Algorithm 1 can work with given tolerance $\epsilon > 0$ or predefined dimension of the subspace r. In other words, we can find a solution to the *fixed-precision* or the *fixed-rank* problem.

In the case of fixed-precision problem, if $\|\mathbf{A}\|_F$ can be calculated, we can set $\omega_k = \|\mathbf{A} - \mathbf{Q}_k\mathbf{T}_k\mathbf{Q}_k^T\|_F$ and use $\omega_k \leq \epsilon$ as stopping criterion, since ω_k can be computed using (2.3), (2.4), (2.6) and (2.35), as

$$\begin{aligned}
\omega_k^2 &= \|\mathbf{A} - \mathbf{Q}_k\mathbf{T}_k\mathbf{Q}_k^T\|_F^2 \\
&= \|\mathbf{A}\|_F^2 + \|\mathbf{Q}_k\mathbf{T}_k\mathbf{Q}_k^T\|_F^2 - 2\langle \mathbf{A}, \mathbf{Q}_k\mathbf{T}_k\mathbf{Q}_k^T \rangle \\
&= \|\mathbf{A}\|_F^2 + \|\mathbf{T}_k\|_F^2 - 2\mathrm{tr}(\mathbf{A}\mathbf{Q}_k\mathbf{T}_k\mathbf{Q}_k^T) \\
&= \|\mathbf{A}\|_F^2 + \|\mathbf{T}_k\|_F^2 - 2\mathrm{tr}\left[\mathbf{Q}_k\mathbf{T}_k\left(\mathbf{Q}_k\mathbf{T}_k\right)^T\right] \\
&= \|\mathbf{A}\|_F^2 - \|\mathbf{T}_k\|_F^2 \\
&= \|\mathbf{A}\|_F^2 - \sum_{j=1}^{k-1}\left(\alpha_j^2 + 2\beta_j^2\right) - \alpha_k^2.
\end{aligned} \tag{2.41}$$

Now we have $\mathbf{A} \approx \mathbf{Q}\mathbf{T}\mathbf{Q}^T$, with $\mathbf{Q}$ $n \times k$ orthonormal matrix and $\mathbf{T}$ $k \times k$ symmetric tridiagonal, and getting spectral decomposition (2.38) of $\mathbf{T}$ with $\hat{\mathbf{U}}$ $k \times r$ orthonormal and $\mathbf{\Lambda}$ $r \times r$ diagonal, where $r \leq k$ is chosen such that

$$\|\mathbf{T} - \hat{\mathbf{U}}\mathbf{\Lambda}\hat{\mathbf{U}}^T\|_F \leq \epsilon, \tag{2.42}$$

gives the low-rank approximation (2.29) of $\mathbf{A}$,

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{2.43}$$

where $\mathbf{U}$ is $n \times r$ orthonormal calculated as (2.39).

Formula (2.41) potentially suffers from cancellation and one may instead use heuristic criterion $\beta_k \leq \epsilon$, since, from (2.37),

$$\|\mathbf{A}\mathbf{Q}_k - \mathbf{Q}_k\mathbf{T}_k\|_F = \beta_k\|\mathbf{q}_{k+1}\|_2 = \beta_k. \tag{2.44}$$

Following (2.40), but with approximation instead of equality, from (2.42) and (2.39) we have (2.43).

When dealing with fixed-rank problem, we add small oversampling parameter p to target rank r to increase accuracy, and stop Algorithm 1 when $k = r + p$. Again, from (2.37) we have (2.44), and, assuming $k < rank(\mathbf{A})$, we have full spectral decomposition (2.38) of $\mathbf{T}$. Since we were aiming at the r-dimensional approximation of $\mathbf{A}$, we only calculate first r columns of $\mathbf{U}$ in (2.39) and end up with $n \times r$ orthonormal matrix $\mathbf{U}$ and the $r \times r$ diagonal $\mathbf{\Lambda}$ such that (2.43) holds.

Complexity

In the case when $\mathbf{A}$ is a full symmetric matrix of order n and when we are dealing with fixed-rank problem with given r and oversampling parameter $p = 0$, Algorithm 1 has a computational complexity of $\mathcal{O}(n^2 r)$. Spectral decomposition of the symmetric tridiagonal $\mathbf{T}$ of order r (2.38) requires additional $\mathcal{O}(r^2)$ operations and creating matrix $\mathbf{U}$ from (2.39) $\mathcal{O}(nr^2)$ operations. Knowing $r \leq n$, the computational cost of Lanczos method is $\mathcal{O}(n^2 r)$.

Furthermore, in Algorithm 1 we have to store $n \times r$ matrix $\mathbf{Q}$ and scalars $\alpha_1, \ldots, \alpha_r$ and $\beta_1, \ldots, \beta_{r-1}$. Additionally, $r \times r$ matrix $\hat{\mathbf{U}}$ in (2.38) and $n \times r$ $\mathbf{U}$ in (2.43). All together, this procedure requires $\mathcal{O}(nr)$ memory.

As stated in the beginning of Section 2.3, our symmetric matrix $\mathbf{A}$ will be given as $\mathbf{A} = \mathbf{Z}\mathbf{Z}^T$, where $\mathbf{Z}$ is a $n \times p$ matrix, with $n < p$. In this case, the multiplication from line 10 in Algorithm 1 will be performed as $\mathbf{r} = \mathbf{Z}(\mathbf{Z}^T\mathbf{q})$, requiring $\mathcal{O}(np)$ operations for each of r iterations, which changes overall computational complexity to $\mathcal{O}(npr)$. And, since we have to create vector $\mathbf{Z}^T\mathbf{q}$ in each iteration, we need $\mathcal{O}(nr + p)$ memory.

Error analysis

If we stop Algorithm 1 when $\|\mathbf{A} - \mathbf{Q}\mathbf{T}\mathbf{Q}^T\|_F$ falls under given tolerance ϵ, getting spectral decomposition of $\mathbf{T}$ such that (2.42) holds, for $n \times r$ matrix $\mathbf{U}$ defined as (2.39) follows

$$\begin{aligned}\|\mathbf{A} - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\|_F &\leq \|\mathbf{A} - \mathbf{Q}\mathbf{T}\mathbf{Q}^T\|_F + \|\mathbf{Q}\mathbf{T}\mathbf{Q}^T - \mathbf{Q}\hat{\mathbf{U}}\mathbf{\Lambda}\hat{\mathbf{U}}^T\mathbf{Q}^T\|_F \\ &\leq \epsilon + \|\mathbf{T} - \hat{\mathbf{U}}\mathbf{\Lambda}\hat{\mathbf{U}}^T\|_F \leq 2\epsilon,\end{aligned}$$

In term of the projector $\mathbf{U}\mathbf{U}^T$, using (2.2) and (2.5), we have

$$\|\mathbf{A} - \mathbf{U}\mathbf{U}^T\mathbf{A}\|_F \leq \|\mathbf{A} - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\|_F + \|\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T - \mathbf{U}\mathbf{U}^T\mathbf{A}\|_F$$

$$\leq 2\epsilon + \|\mathbf{AU} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\mathbf{U}\|_F$$
$$\leq 2\epsilon + \|\mathbf{A} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\|_F\|\mathbf{U}\|_2 \leq 4\epsilon. \tag{2.45}$$

2.3.2 Randomized algorithm

Randomized algorithms provide a powerful tool for performing low-rank matrix approximations. Using random sampling for identifying a subspace that captures most of the action of a matrix and then compressing the matrix to this subspace leads to the desired low-rank factorization. In many cases, this approach beats its classical competitors in terms of speed, while usually being on par in terms of accuracy and robustness. Furthermore, it can produce factorizations that are accurate to any specified tolerance above machine precision, which allows the user to trade accuracy for speed if desired.

Throughout this section we follow [15] to explain how to not only get low-rank approximation (2.29) for a symmetric matrix, but also how to get the SVD (2.7) of a general rectangular matrix.

Description

Given general $\mathbf{A} \in \mathbb{R}^{m\times n}$, the task of computing a low-rank approximation can be split into two computational stages.

Stage A: Computing an approximate basis for the range of $\mathbf{A}$, i.e. finding an orthonormal $m \times l$ matrix $\mathbf{Q}$, with l as small as possible, for which

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A}, \tag{2.46}$$

up to a given tolerance.

Stage B: Given orthonormal $\mathbf{Q}$ that satisfies (2.46), compute the desired factorization of $\mathbf{A}$.

The task in Stage A can be executed very efficiently with random sampling methods, while Stage B can be computed with well established deterministic methods. Now we explain each stage in details.

Stage A. A motivational example is very simple. Suppose we seek a basis for the range of the matrix $\mathbf{A}$ with exact rank r. We refer to this problem as the *fixed-rank problem*. If

we draw a set of r random vectors $\boldsymbol{\omega}_j$ (for now, the precise distribution is unimportant) and form r products

$$\mathbf{y}_j = \mathbf{A}\boldsymbol{\omega}_j, \quad j = 1, \ldots, r,$$

then, owing to the randomness, it is likely that the set $\{\boldsymbol{\omega}_j : j = 1, \ldots, r\}$ forms a linearly independent set and no linear combination of vectors $\boldsymbol{\omega}_j$ falls in the null space of $\mathbf{A}$. As a result, the set $\{\mathbf{y}_j : j = 1, \ldots, r\}$ of sample vectors is also linearly independent so it spans the range of $\mathbf{A}$. Therefore, to produce an orthonormal basis for the range of $\mathbf{A}$, we just need to orthonormalize the sample vectors.

Now, imagine that $\mathbf{A} = \hat{\mathbf{A}} + \mathbf{E}$, where $\hat{\mathbf{A}}$ is a rank-r matrix containing the information we seek and $\mathbf{E}$ is a small perturbation. We want to obtain a basis that covers as much of the range of $\hat{\mathbf{A}}$ as possible, rather then to minimize the number of basis vector. Therefore, we fix a small number p, and generate $r + p$ samples

$$\mathbf{y}_j = \mathbf{A}\boldsymbol{\omega}_j = \hat{\mathbf{A}}\boldsymbol{\omega}_j + \mathbf{E}\boldsymbol{\omega}_j, \quad j = 1, \ldots, r + p.$$

The perturbation $\mathbf{E}$ shifts the direction of each sample vector outside the range of $\hat{\mathbf{A}}$, which can prevent the span of $\{\mathbf{y}_j : j = 1, \ldots, r\}$ from covering the entire range of $\hat{\mathbf{A}}$. In contrast, the enriched set $\{\mathbf{y}_j : j = 1, \ldots, r+p\}$ has a much better chance of spanning the required subspace. We refer to p as *oversampling parameter* and setting $p = 5$ or $p = 10$ has shown to often be adequate.

The most natural way to choose random vectors $\boldsymbol{\omega}_j$ is from the standard Gaussian distribution. Then by setting $l = r + p$, creating matrix $\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\omega}_1 & \boldsymbol{\omega}_2 & \cdots & \boldsymbol{\omega}_l \end{bmatrix}$ and setting $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, we get matrix $\mathbf{Q}$ that satisfies (2.46) from truncated QR factorization of $\mathbf{Y}$. The described procedure is presented in Algorithm 2.

Theorem 2.3.1. *[15, Theorem 1.1] Given matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, a target rank $r \geq 2$ and an oversampling parameter $p \geq 2$, with $r + p \leq \min\{m, n\}$, execute Algorithm 2 with a standard Gaussian test matrix to obtain $m \times (r + p)$ orthonormal matrix $\mathbf{Q}$. Then*

$$\mathbb{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 \leq \left[1 + \frac{4\sqrt{r+p}}{p-1} \cdot \sqrt{\min\{m, n\}}\right] \sigma_{r+1},$$

where $\mathbb{E}$ denoted expectation with respect to the random test matrix and σ_{r+1} is the $(r+1)$th singular value of $\mathbf{A}$.

So the resulting error of Algorithm 2 is not far away from best approximation error σ_{r+1} in norm $\|\cdot\|_2$ (see Theorem 2.1.2), as presented in the following example.

Algorithm 2 Fixed randomized range finder

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$*, a target rank* r *and an oversampling parameter* p*, the following procedure computes an* $m \times (r+p)$ *orthonormal matrix* $\mathbf{Q}$ *such that* $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A}$*.*

1: **procedure** FIXED_RAND_RANGE($\mathbf{A}$, r, $p = 10$)
2: $l = r + p$
3: Draw an $n \times l$ Gaussian random matrix $\mathbf{\Omega}$.
4: Form the $m \times l$ matrix $\mathbf{Y} = \mathbf{A\Omega}$
5: Construct an $m \times l$ matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$ using QR factorization $\mathbf{Y} = \mathbf{QR}$
6: **end procedure**

Example 2.3.1. *Let* $\mathbf{A}$ *be matrix of order* $n = 20$ *generated by evaluating the function* $f(x, y) = \frac{1}{x+y}$ *on the grid* $\{0.1, 0.2, 0.3, \ldots, 2\}$*. Its sixth and seventh singular values are* $\sigma_6 = 2.27423 \times 10^{-4}$ *and* $\sigma_7 = 1.45739 \times 10^{-5}$*. Running Algorithm 2 with* $r = 6$ *and* $p = 10$ *results in matrix* $\mathbf{Q}$ *for which it holds*

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 = 5.889302 \times 10^{-15},$$

while the bound from Theorem 2.3.1 equals 1.30443×10^{-4}*.*

However, in practice, the target rank r is rarely known in advance. Randomized algorithms are usually implemented in an adaptive fashion where the number of samples is increased until the error satisfies the desired tolerance, so the user does not choose the oversampling parameter. This type of problem is called the *fixed-precision problem*. For a given matrix $\mathbf{A}$ and a positive error tolerance ϵ, we seek a matrix $\mathbf{Q}$ with $l = l(\epsilon)$ orthonormal columns such that

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 \leq \epsilon. \tag{2.47}$$

To check if we have reached the desired tolerance we set a small integer k and use a fact that for a sequence $\{\boldsymbol{\omega}_j : j = 1, 2, \ldots, k\}$ of standard Gaussian vectors,

$$\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\|_2 \leq 10\sqrt{\frac{2}{\pi}} \max_{j=1,\ldots,k} \|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\boldsymbol{\omega}_j\|_2, \tag{2.48}$$

stands with probability at least $1 - 10^{-k}$. Here, the integer k is used to balance computational cost and reliability.

This statement follows by setting $\mathbf{B} = \left(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T\right)\mathbf{A}$ and $\alpha = 10$ in the following lemma, whose proof appears in [31].

Lemma 2.3.2. *[15, Lemma 4.1] For $\mathbf{B} \in \mathbb{R}^{m\times n}$, a positive integer k and real number $\alpha > 1$, drawing an independent family $\{\boldsymbol{\omega}_j : j = 1, \ldots, k\}$ of standard Gaussian vectors gives*

$$\|\mathbf{B}\|_2 \leq \alpha\sqrt{\frac{2}{\pi}} \max_{j=1,\ldots,k} \|\mathbf{B}\boldsymbol{\omega}_j\|_2$$

except with probability α^{-k}.

The estimate (2.48) is computationally inexpensive because it requires only a small number of matrix-vector products and it can be combined with any method for constructing an approximate basis for the range of a matrix.

We incorporate this in Algorithm 2 by generating the basis from Step 5 incrementally. Starting with an empty matrix $\mathbf{Q}_0$, we build $\mathbf{Q}$ column by column in the following way:

for $j = 1, 2, 3, \ldots$ **do**

Draw an $n \times 1$ Gaussian random vector $\boldsymbol{\omega}_j$ and set $\mathbf{y}_j = \mathbf{A}\boldsymbol{\omega}_j$.

Compute $\tilde{\mathbf{q}}_j = (\mathbf{I} - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^T)\mathbf{y}_j$

Normalize $\mathbf{q}_j = \tilde{\mathbf{q}}_j/\|\tilde{\mathbf{q}}_j\|_2$, and form $\mathbf{Q}_j = [\,\mathbf{Q}_{j-1} \quad \mathbf{q}_j\,]$.

end for

The vectors $\tilde{\mathbf{q}}_j$ are precisely the vectors that appear in the error bound (2.48), so we break the loop once we observe k consecutive vectors $\tilde{\mathbf{q}}_j$ whose norms are smaller than $\epsilon/(10\sqrt{2/\pi})$.

A potential complication of the method is that the vectors $\tilde{\mathbf{q}}_j$ become small as the basis starts to capture most of the action of $\mathbf{A}$. In finite-precision arithmetics, their direction is extremely unreliable. To address this problem, we reproject vector $\mathbf{q}_j$ onto the $\mathcal{R}(\mathbf{Q}_{j-1})^{\perp}$. The procedure is presented in Algorithm 3.

Moreover, when matrix $\mathbf{A}$ is symmetric, the columns of $\mathbf{Q}$ form a good basis for both the column space and the row space of $\mathbf{A}$ so that we have

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T. \tag{2.49}$$

More precisely, when (2.47) is in force, we have

$$\begin{aligned}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T\|_2 &= \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A} + \mathbf{Q}\mathbf{Q}^T\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T\|_2 \\ &\leq \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 + \|\mathbf{Q}\mathbf{Q}^T\left(\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}^T\right)\|_2 \leq 2\epsilon. \end{aligned} \tag{2.50}$$

Algorithm 3 Adaptive randomized range finder

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$*, a tolerance* ϵ*, an integer* k*, the following procedure computes an orthonormal matrix* $\mathbf{Q}$ *such that* $\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\|_2 \leq \epsilon$ *with probability at least* $1 - \min\{m, n\}10^{-k}$*.*

procedure ADAPTIVE_RAND_RANGE($\mathbf{A}$, $\epsilon = 10^{-8}$, $k = 10$, maxit $= 1000$)
  maxit $= \min\{m, n, \text{maxit}\}$
  Draw standard Gaussian vectors $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_k$ of length n
  $[\,\mathbf{y}_1\ \mathbf{y}_2\ \cdots\ \mathbf{y}_k\,] = \mathbf{A}[\,\boldsymbol{\omega}_1\ \boldsymbol{\omega}_2\ \cdots\ \boldsymbol{\omega}_k\,]$
  Set $j = 0$ and $\mathbf{Q} = [\ ]$ ($m \times 0$ empty matrix)
  while $\max\{\|\mathbf{y}_{j+1}\|_2, \|\mathbf{y}_{j+2}\|_2, \ldots, \|\mathbf{y}_{j+r}\|_2\} > \epsilon/(10\sqrt{2/\pi})$ **and** $j <$ maxit **do**
    $j = j + 1$
    $\mathbf{y}_j \leftarrow (\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{y}_j$
    $\mathbf{q} = \mathbf{y}_j/\|\mathbf{y}_j\|_2$
    $\mathbf{Q} = [\,\mathbf{Q}\quad \mathbf{q}\,]$
    Draw a standard Gaussian vector $\boldsymbol{\omega}_{j+k}$ of length n
    $\mathbf{y}_{j+k} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\boldsymbol{\omega}_{j+k}$
    for $i = j + 1, j + 2, \ldots, j + k - 1$ **do**
      $\mathbf{y}_i \leftarrow \mathbf{y}_i - \mathbf{q}(\mathbf{q}^T\mathbf{y}_i)$
    end for
  end while
end procedure

The last inequality relies on the fact that $\|\mathbf{Q}\mathbf{Q}^T\|_2 = 1$ and that

$$\|\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}^T\|_2 = \|\left(\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}^T\right)^T\|_2 = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2.$$

Stage B. Once we finish Stage A and have orthonormal matrix $\mathbf{Q}$ such that (2.46) holds (or (2.49), in case $\mathbf{A}$ is symmetric), we can calculate low-rank approximations of $\mathbf{A}$ in several steps. For a non-symmetric matrix, we are interested in its SVD (2.7), which we get by applying Algorithm 4, and for a symmetric matrix in its spectral decomposition (2.22), which we get from Algorithm 5.

Algorithm 4 Direct SVD

Given an $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and orthonormal* $\mathbf{Q} \in \mathbb{R}^{m\times l}$ *such that* $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 \leq \epsilon$*, the following procedure computes the SVD of* $\mathbf{A}$*,* $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$*, where* $\mathbf{U} \in \mathbb{R}^{m\times r}$*,* $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ *and* $\mathbf{V} \in \mathbb{R}^{n\times r}$ *and* $r \leq l$*.*

1: Form $l \times n$ matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$.
2: Compute an SVD of $\mathbf{B}$: $\mathbf{B} \approx \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^T$, with $\tilde{\mathbf{U}} \in \mathbb{R}^{l\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$, where r is chosen such that $\|\mathbf{B} - \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^T\|_2 \leq \epsilon$.
3: Set $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$.

Algorithm 5 Direct spectral decomposition

Given a symmetric $\mathbf{A} \in \mathbb{R}^{n\times n}$ *and orthonormal* $\mathbf{Q} \in \mathbb{R}^{n\times l}$ *such that* $\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T\|_2 \leq 2\epsilon$*, the following procedure computes the spectral decomposition of* $\mathbf{A}$*,* $\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$*, where* $\mathbf{U} \in \mathbb{R}^{n\times r}$ *and* $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$ *and* $r \leq l$*.*

1: Form $l \times l$ matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{A}\mathbf{Q}$.
2: Compute an spectral decomposition of $\mathbf{B}$: $\mathbf{B} \approx \tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^T$, with $\tilde{\mathbf{U}} \in \mathbb{R}^{l\times r}$ and $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$, where r is chosen such that $\|\mathbf{B} - \tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^T\|_2 \leq \epsilon$.
3: Set $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$.

Complexity

Assume that $m \times n$ matrix $\mathbf{A}$ and target rank $r < \min\{m, n\}$ are given. Stage A (Algorithm 2) requires $\mathcal{O}(mnr)$ operations - $\mathcal{O}(mnr)$ for multiplication on line 4 and $\mathcal{O}(mr^2)$ for the QR decomposition on line 5; using $\mathcal{O}(\max\{m, n\}r)$ memory. In Stage B, when calculating the SVD of $\mathbf{A}$ (Algorithm 4), forming matrix $\mathbf{B}$ takes $\mathcal{O}(mnr)$ operations, getting its SVD $\mathcal{O}(nr^2)$ and forming matrix $\mathbf{U}$ $\mathcal{O}(mr^2)$. All together, Stage B takes $\mathcal{O}(mnr)$ operations and $\mathcal{O}(\max\{m, n\}r)$ memory.

When $\mathbf{A}$ is symmetric with $m = n$, Stage A takes $\mathcal{O}(n^2r)$ operations and $\mathcal{O}(nr)$ memory, while in Stage B we use Algorithm 5 to get the spectral decomposition of $\mathbf{A}$, and it takes $\mathcal{O}(n^2r)$ operations to form matrix $\mathbf{B}$, $\mathcal{O}(r^3)$ to get its spectral decompositions and $\mathcal{O}(nr^2)$ for forming matrix $\mathbf{U}$; resulting in $\mathcal{O}(n^2r)$ operations and $\mathcal{O}(nr)$ memory in total.

Since the symmetric matrix $\mathbf{A}$ will often be given as $\mathbf{A} = \mathbf{Z}\mathbf{Z}^T$, where $\mathbf{Z}$ is an $n \times p$ matrix, with $n < p$ (see the beginning of Section 2.3), the multiplication from line 4 in Algorithm 2, with $m = n$, will be performed as $\mathbf{Y} = \mathbf{Z}(\mathbf{Z}^T\mathbf{\Omega})$, requiring $\mathcal{O}(npr)$

operations, which changes the complexity of Stage A to $\mathcal{O}(npr)$ operations and $\mathcal{O}(pr)$ memory. For Stage B, in Algorithm 5 we form matrix $\mathbf{B}$ by first setting it to $\mathbf{B} = \mathbf{Q}^T\mathbf{Z}$ and then $\mathbf{B} = \mathbf{B}\mathbf{B}^T$, in $\mathcal{O}(npr)$ operations and $\mathcal{O}(pr)$ memory, which is also the resulting complexity of Stage B.

Error analysis

For a rectangular $\mathbf{A}$, from (2.47) and Step 2 of Algorithm 4 we have

$$\begin{aligned}\|\mathbf{A} - \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T\|_2 &= \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A} + \mathbf{Q}\mathbf{Q}^T\mathbf{A} - \mathbf{Q}\hat{\mathbf{U}}\boldsymbol{\Sigma}\mathbf{V}^T\|_2 \\ &\leq \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 + \|\mathbf{Q}\left(\mathbf{B} - \hat{\mathbf{U}}\boldsymbol{\Sigma}\mathbf{V}^T\right)\|_2 \leq 2\epsilon.\end{aligned}$$

Similarly, for a symmetric $\mathbf{A}$, from (2.50) and Step 2 of Algorithm 5 we have

$$\begin{aligned}\|\mathbf{A} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\|_2 &= \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T + \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T - \mathbf{Q}\hat{\mathbf{U}}\boldsymbol{\Lambda}\hat{\mathbf{U}}^T\mathbf{Q}^T\|_2 \\ &\leq \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\mathbf{Q}\mathbf{Q}^T\|_2 + \|\mathbf{Q}\left(\mathbf{B} - \hat{\mathbf{U}}\boldsymbol{\Lambda}\hat{\mathbf{U}}^T\right)\mathbf{Q}^T\|_2 \leq 3\epsilon,\end{aligned}$$

and in term of the projector $\mathbf{U}\mathbf{U}^T$, similarly as in (2.45), using (2.5), we have

$$\begin{aligned}\|\mathbf{A} - \mathbf{U}\mathbf{U}^T\mathbf{A}\|_2 &\leq \|\mathbf{A} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\|_2 + \|\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T - \mathbf{U}\mathbf{U}^T\mathbf{A}\|_2 \\ &\leq 3\epsilon + \|\mathbf{A} - \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\|_2\|\mathbf{U}\|_2 \leq 6\epsilon. \qquad (2.51)\end{aligned}$$

2.3.3 Lanczos method vs. randomized algorithm

According to experiments from [4], Lanczos and randomized algorithms described above are often quite similar in terms of the number of matrix-vector multiplications needed to attain a certain accuracy. For slow singular value decays, randomized algorithms tend to require slightly more iterations. On the other hand, when dealing with fixed-rank problem, randomized algorithm can perform the matrix-vector multiplication $\mathbf{A}\boldsymbol{\Omega}$ in blocks. Also, using a random matrix $\boldsymbol{\Omega}$ that has some internal structure allows to evaluate the product $\mathbf{A}\boldsymbol{\Omega}$ rapidly. Therefore, we will use randomized algorithm when $\mathbf{A}$ is explicitly available or when we can benefit from the structure of $\boldsymbol{\Omega}$. When $\mathbf{A}$ is only available via matrix-vector products, we use the Lanczos algorithm.

Chapter 3

Tensors in Tucker format and the HOSVD

In this chapter we present basic tensor theory, operations like mode-n matricization and n-mode product, the HOSVD algorithm and its modification together with their properties, and we also explain how the presented operations and the HOSVD can be efficiently computed.

3.1 Tensors - basic operations and properties

Let $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. The *(Frobenius) norm* is defined as

$$\|\mathcal{X}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N}^2},$$

and the *inner product* as

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} y_{i_1 i_2 \cdots i_N}.$$

Obviously $\|\mathcal{X}\|_F^2 = \langle \mathcal{X}, \mathcal{X} \rangle$.

A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is called *diagonal* if $x_{i_1 i_2 \cdots i_N} \neq 0$ only if $i_1 = i_2 = \cdots = i_N$.

The same way we generalize Hadamard product of matrices to tensors (see Chapter 1), we can generalize Kronecker product from Section 2.2 to tensors.

The *Kronecker product* of two tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ is a tensor $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y} \in \mathbb{R}^{I_1 J_1 \times \cdots \times I_N J_N}$ with entries

$$z_{k_1 \cdots k_N} = x_{i_1 \cdots i_N} y_{j_1 \cdots j_N}, \quad k_n = j_n + (i_n - 1) J_n, \quad n = 1, \ldots, N. \tag{3.1}$$

Fixing a subset of indices of a tensor creates a subtensor. In particular, Hadamard product $\mathcal{X} * \mathcal{Y}$ is a subtensor of Kronecker product $\mathcal{X} \otimes \mathcal{Y}$.

Furthermore, for Hadamard and Kronecker product of tensors, just as with matrices, it holds

$$\|\mathcal{X} \otimes \mathcal{Y}\|_F = \|\mathcal{X}\|_F \|\mathcal{Y}\|_F \quad \text{and} \quad \|\mathcal{X} * \mathcal{Y}\|_F \leq \|\mathcal{X}\|_F \|\mathcal{Y}\|_F, \tag{3.2}$$

which can easily be shown.

3.1.1 Matricization of a tensor

Two types of subtensors are particularly important.

Fibers of a tensor are defined by fixing every index but one. Leaving index n free creates mode-n fibers (also sometimes called mode-n vectors). A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. Third-order tensors have column, row and tube fibers denoted by $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$, $\mathbf{x}_{ij:}$, respectively, see Figure 3.1.

Slices of a tensor are two-dimensional sections of a tensor, obtained by fixing all but two indices. Third-order tensors have horizontal, lateral and frontal slices, denoted by $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$, $\mathbf{X}_{::k}$, respectively, see Figure 3.2. Frontal slices are sometimes denoted more compactly as $\mathbf{X}_k$ and are of greatest importance, since they are used to display tensors in computers; see Figure 3.3.

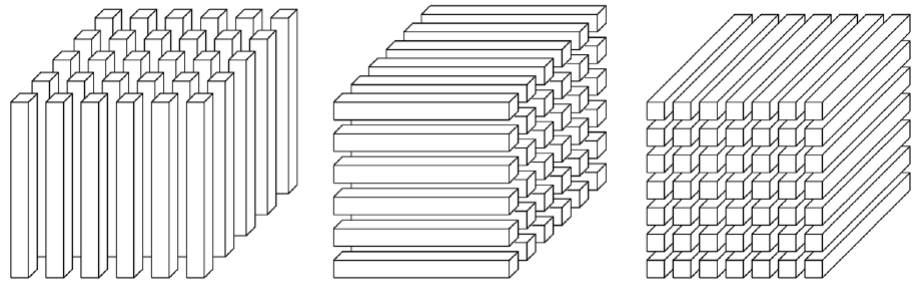
Tensors can be transformed into a matrix or a vector by processes called *matricization (unfolding, flattening)* and *vectorization*. In the following we define these operations.

Definition 3.1.1. The mode-n matricization *of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *is a process of transforming the tensor into an* $I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$ *matrix, denoted as* $\mathbf{X}_{(n)}$*, by arranging mode-n fibers of the tensor into columns of the resulting matrix. Formally, tensor element* $(i_1, \ldots, i_N)$ *is mapped to matrix element* (i_n, j)*, where*

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k, \quad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m.$$

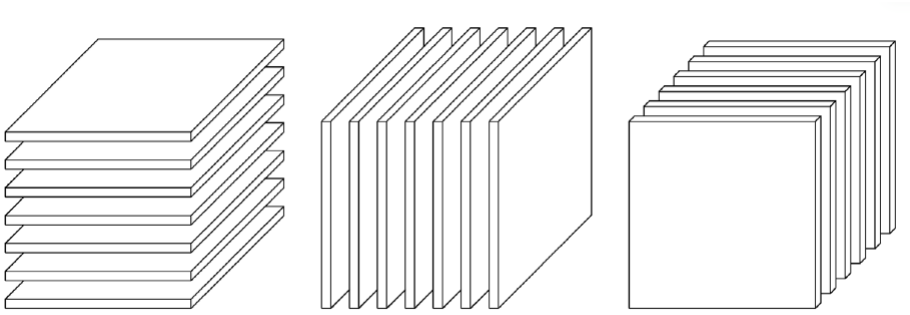
We will refer to $\mathbf{X}_{(n)}$ as the *matricized* tensor $\mathcal{X}$ or the *mode-n matricization* of $\mathcal{X}$. So we use the term matricization do denote the transformation, but also the matrix.

Sometimes, the mode-n matricization is defined with different ordering of the columns, but in general, the specific permutation is not important so long as it is consistent across related calculations [20].



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

Figure 3.1: Fibers of a third-order tensors. Taken from [20].



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

Figure 3.2: Slices of a third-order tensors. Taken from [20].

```
rand(3,2,3)
```

```
3×2×3 Array{Float64,3}:
[:, :, 1] =
 0.844833  0.460671
 0.344661  0.605859
 0.908012  0.533399

[:, :, 2] =
 0.744155  0.640433
 0.79224   0.159842
 0.97195   0.87076

[:, :, 3] =
 0.9222    0.162576
 0.034694  0.267297
 0.587094  0.915614
```

```
rand(2,2,2,2)
```

```
2×2×2×2 Array{Float64,4}:
[:, :, 1, 1] =
 0.92666   0.405135
 0.890238  0.597987

[:, :, 2, 1] =
 0.838868  0.59535
 0.141392  0.785136

[:, :, 1, 2] =
 0.248774  0.474257
 0.356674  0.934859

[:, :, 2, 2] =
 0.29251   0.743729
 0.307668  0.102512
```

Figure 3.3: In programming languages, tensors are displayed by their frontal slices, i.e., by fixing all indices except the first two.

Example 3.1.1. *Let* $\mathcal{X} \in \mathbb{R}^{4\times3\times2}$ *be defined by its frontal slices*

$$\mathbf{X}_1 = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 13 & 17 & 21 \\ 14 & 18 & 22 \\ 15 & 19 & 23 \\ 16 & 20 & 24 \end{bmatrix}, \tag{3.3}$$

then the three mode-n matricizations are

$$\mathbf{X}_{(1)} = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{bmatrix},$$

$$\mathbf{X}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 13 & 14 & 15 & 16 \\ 5 & 6 & 7 & 8 & 17 & 18 & 19 & 20 \\ 9 & 10 & 11 & 12 & 21 & 22 & 23 & 24 \end{bmatrix},$$

$$\mathbf{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}.$$

Definition 3.1.2. *To* vectorize *a tensor* $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$ *means to arrange its mode-*1 *fibers into one column. The result is an* $I_1 \cdots I_N$ *vector, denoted as* $vec(\mathcal{X})$.

Obviously, this definition fits standard matrix vectorization, where mode-1 fibers are exactly columns of the matrix, and $vec(\mathcal{X}) = vec\left(\mathbf{X}_{(1)}\right)$.

Example 3.1.2. *For tensor* $\mathcal{X}$ *from Example 3.1.1, we have*

$$vec(\mathcal{X}) = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 24 \end{bmatrix}.$$

Now we can express norm and inner product of a tensor in terms of matrix or vector norm of matricized and vectorized tensor.

Proposition 3.1.1. *Let* $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1\times\cdots\times I_N}$. *Then the following equalities hold:*

(1) $\|\mathcal{X}\|_F = \|\mathbf{X}_{(n)}\|_F, \quad n = 1, \ldots, N,$

(2) $\|\mathcal{X}\|_F = \|\, vec(\mathcal{X})\|_2$,

(3) $\langle \mathcal{X}, \mathcal{Y} \rangle = vec(\mathcal{X})^T\, vec(\mathcal{Y})$,

(4) $\|\mathcal{X} - \mathcal{Y}\|_F^2 = \|\mathcal{X}\|_F^2 + \|\mathcal{Y}\|_F^2 - 2\langle \mathcal{X}, \mathcal{Y} \rangle$.

Proof. All properties follow directly from the definitions of Frobenius norm and inner product of tensors. □

3.1.2 Tensor-matrix multiplication: The n-mode product

We can multiply a tensor by a matrix or a vector, by each of its modes. We call that product the n-mode product.

Definition 3.1.3. *The* n-mode (matrix) product *of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *and a matrix* $\mathbf{U} \in \mathbb{R}^{J \times I_n}$*, denoted by* $\mathcal{X} \times_n \mathbf{U}$*, is a tensor of size* $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$*, computed by multiplying each mode-n fiber of* $\mathcal{X}$ *by* $\mathbf{U}$*. Element-wise,*

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \cdots i_N} u_{j i_n}. \tag{3.4}$$

Proposition 3.1.2. *If* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *and* $\mathbf{U} \in \mathbb{R}^{J \times I_n}$*, then*

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \quad \Leftrightarrow \quad \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}. \tag{3.5}$$

Proof. Elements of $\mathcal{Y}$ are given by (3.4). Applying mode-n matricization, element $(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_n)$ maps to element (j, k), with

$$k = 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (i_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m;$$

see Definition 3.1.1. On the other side, (j, k) element of matrix $\mathbf{U}\mathbf{X}_{(n)}$, with k as stated above, is exactly

$$\sum_{i_n=1}^{I_n} u_{j i_n} \left(\mathbf{X}_{(n)}\right)_{i_n k} = \sum_{i_n=1}^{I_n} u_{j i_n} x_{i_1 \cdots i_N},$$

which proves the statement. □

Example 3.1.3. *Let* $\mathcal{X}$ *be the tensor from Example 3.1.1 and let* $\mathbf{U} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$*. Then the 2-mode product of* $\mathcal{X}$ *and* $\mathbf{U}$ *is*

$$\mathcal{Y} = \mathcal{X} \times_2 \mathbf{U} \in \mathbb{R}^{4 \times 2 \times 2}$$

with frontal slices

$$\mathbf{Y}_1 = \begin{bmatrix} 61 & 76 \\ 70 & 88 \\ 79 & 100 \\ 88 & 112 \end{bmatrix}, \quad \mathbf{Y}_2 = \begin{bmatrix} 169 & 220 \\ 178 & 232 \\ 187 & 244 \\ 196 & 256 \end{bmatrix}.$$

Proposition 3.1.3. *For tensors and matrices of appropriate size, the n-mode product has the following properties:*

(1) $\mathcal{X} \times_n (\mathbf{A} + \mathbf{B}) = \mathcal{X} \times_n \mathbf{A} + \mathcal{X} \times_n \mathbf{B}$

(2) $\mathcal{X} \times_m \mathbf{A} \times_n \mathbf{B} = \mathcal{X} \times_n \mathbf{B} \times_m \mathbf{A}, \quad m \neq n,$

(3) $\mathcal{X} \times_n \mathbf{A} \times_n \mathbf{B} = \mathcal{X} \times_n (\mathbf{BA}),$

(4) $\langle \mathcal{X}, \mathcal{Y} \times_n \mathbf{A} \rangle = \langle \mathcal{X} \times_n \mathbf{A}^T, \mathcal{Y} \rangle,$

(5) *if* $\mathbf{U}$ *is an orthonormal matrix, then*

(i) $\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \quad \Rightarrow \quad \mathcal{X} = \mathcal{Y} \times_n \mathbf{U}^T,$

(ii) $\|\mathcal{X}\|_F = \|\mathcal{X} \times_n \mathbf{U}\|_F,$

(6) $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)} \quad \Leftrightarrow$

$$\mathbf{Y}_{(n)} = \mathbf{A}^{(n)} \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right)^T.$$

(7) $\|\mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}\|_F \leq \|\mathcal{X}\|_F \|\mathbf{A}^{(1)}\|_F \|\mathbf{A}^{(2)}\|_F \cdots \|\mathbf{A}^{(N)}\|_F.$

Proof. (1) Denoting $\mathcal{Y} = \mathcal{X} \times_n (\mathbf{A} + \mathbf{B})$, from (3.5) we have

$$\mathbf{Y}_{(n)} = (\mathbf{A} + \mathbf{B}) \mathbf{X}_{(n)} = \mathbf{A}\mathbf{X}_{(n)} + \mathbf{B}\mathbf{X}_{(n)},$$

from which it follows $\mathcal{Y} = \mathcal{X} \times_n \mathbf{A} + \mathcal{X} \times_n \mathbf{B}$.

(2) Let $\mathcal{Y} = \mathcal{X} \times_m \mathbf{A}$ and $\mathcal{Z} = \mathcal{X} \times_n \mathbf{B}$ and, without loss of generality, assume $m < n$. Then for every set of indices, from (3.4),

$$\begin{aligned} (\mathcal{Y} \times_n \mathbf{B})_{i_1 \cdots i_{m-1} j i_{m+1} \cdots i_{n-1} k i_{n+1} \cdots i_N} &= \sum_{i_n=1}^{I_n} y_{i_1 \cdots i_{m-1} j i_{m+1} \cdots i_N} b_{k i_n} \\ &= \sum_{i_n=1}^{I_n} \left(\sum_{i_m=1}^{I_m} x_{i_1 \cdots i_N} a_{j i_m} \right) b_{k i_n} = \sum_{i_m=1}^{I_m} \left(\sum_{i_n=1}^{I_n} x_{i_1 \cdots i_N} b_{k i_n} \right) a_{j i_m} \\ &= \sum_{i_m=1}^{I_m} z_{i_1 \cdots i_{n-1} k i_{n+1} \cdots i_N} a_{j i_m} = (\mathcal{Z} \times_m \mathbf{A})_{i_1 \cdots i_{m-1} j i_{m+1} \cdots i_{n-1} k i_{n+1} \cdots i_N}. \end{aligned}$$

(3) Denoting $\mathcal{Y} = \mathcal{X} \times_n \mathbf{A} \times_n \mathbf{B}$, from (3.5) we have

$$\mathbf{Y}_{(n)} = \mathbf{B} (\mathcal{X} \times_n \mathbf{A})_{(n)} = \mathbf{B}\mathbf{A}\mathbf{X}_{(n)},$$

from which it follows $\mathcal{Y} = \mathcal{X} \times_n (\mathbf{BA})$.

(4) Follows directly from definition of inner product and Definition 3.1.3.

(5) For $I_1 \times \cdots \times I_N$ tensor $\mathcal{X}$ and $J \times I_n$ orthonormal matrix $\mathbf{U}$, $\mathcal{Y} = \mathcal{X} \times_n \mathbf{U}$ is a tensor of size $I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. From (3.5) follows $\mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$ and multiplying this from the left with $\mathbf{U}^T$, we get $\mathbf{X}_{(n)} = \mathbf{U}^T\mathbf{Y}_{(n)}$, from which it follows $\mathcal{X} = \mathcal{Y} \times_n \mathbf{U}^T$.

Furthermore, from (3.1.1.(1)) and orthogonal invariance of matrix Frobenius norm (2.6), we have

$$\|\mathcal{X} \times_n \mathbf{U}\|_F = \|\mathbf{U}\mathbf{X}_{(n)}\|_F = \|\mathbf{X}_{(n)}\|_F = \|\mathcal{X}\|_F.$$

(6) Let $\mathcal{X}$ be $I_1 \times \cdots \times I_N$ tensor and $\mathbf{A}^{(n)}$ $J_n \times I_n$ matrices. By Definition 3.1.1, each $y_{j_1 \cdots j_N}$ element of $J_1 \times \cdots \times J_N$ tensor $\mathcal{Y}$ is mapped to $y_{j_n k}$ element of matrix $\mathbf{Y}_{(n)}$, with

$$k = 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (j_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} J_m.$$

We will prove the statement by showing that every element

$$\left(\mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}\right)_{j_1 \cdots j_N}$$

maps to element

$$\left[\mathbf{A}^{(n)}\mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right)^T\right]_{j_n k},$$

with k as stated.

From Definition 3.1.3, we have

$$\left(\mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}\right)_{j_1 \cdots j_N} = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} x_{i_1 \cdots i_N} a^{(1)}_{j_1 i_1} \cdots a^{(N)}_{j_N i_N}.$$

On the other hand, by denoting $\mathbf{M}_n = \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right)^T$, we have

$$\begin{aligned}
\left(\mathbf{A}^{(n)}\mathbf{X}_{(n)}\mathbf{M}_n\right)_{j_n k} &= \mathbf{A}^{(n)}[\, j_n\,,\, :\,] \left(\mathbf{X}_{(n)}\mathbf{M}_n\right)[\, :\,,\, k\,] \\
&= \sum_{i_n=1}^{I_n} a^{(n)}_{j_n i_n} \left(\mathbf{X}_{(n)}\mathbf{M}_n\right)[\, i_n\,,\, k\,] \\
&= \sum_{i_n=1}^{I_n} a^{(n)}_{j_n i_n} \sum_{i=1}^{\hat{I}_n} \mathbf{X}_{(n)}[\, i_n\,,\, i\,]\mathbf{M}_n[\, i\,,\, k\,], \qquad (3.6)
\end{aligned}$$

with $\hat{I}_n = I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$. Now, $\mathbf{X}_{(n)}[\, i_n \,,\, i \,] = x_{i_1 \cdots i_N}$, with

$$i = 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (i_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m.$$

From the definition of Kronecker product follows that the same i stands in

$$\mathbf{M}_n[\, i \,,\, k \,] = \tilde{a}^{(N)}_{i_N j_N} \cdots \tilde{a}^{(n+1)}_{i_{n+1} j_{n+1}} \tilde{a}^{(n+1)}_{i_{n-1} j_{n-1}} \cdots \tilde{a}^{(1)}_{i_1 j_1},$$

with $\tilde{a}^{(m)}_{i_m j_m}$ denoting an element of $\mathbf{A}^{(m)^T}$. Using these conclusions, we can rewrite (3.6) as

$$\sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} x_{i_1 \cdots i_N} a^{(1)}_{j_1 i_1} \cdots a^{(N)}_{j_N i_N},$$

which completes the proof.

(7) Follows directly from (3.1.3.(6)), (3.1.1.(1)) and (3.2).

□

For a matrix $\mathbf{X}$, i.e. tensor of order $N = 2$, applying matricization from Definition 3.1.1 results in

$$\mathbf{X}_{(1)} = \mathbf{X}, \quad \mathbf{X}_{(2)} = \mathbf{X}^T.$$

If we multiply $\mathbf{X}$ by each mode with matrices $\mathbf{A}$ and $\mathbf{B}$ of appropriate sizes, from (3.1.3.(6)) follows

$$\mathbf{Y} = \mathbf{X} \times_1 \mathbf{A} \times_2 \mathbf{B} \quad \Leftrightarrow \quad \mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{B}^T, \tag{3.7}$$

so the SVD (2.7) of a matrix $\mathbf{X}$ can be rewritten in terms of the n-mode product as

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V}. \tag{3.8}$$

Also, we can use the n-mode product to get any element of a tensor. If $\mathbf{\mathcal{X}} = (x_{i_1 i_2 \cdots i_N}) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, then

$$x_{i_1 i_2 \cdots i_N} = \mathbf{\mathcal{X}} \times_1 \mathbf{e}^T_{i_1} \times_2 \mathbf{e}^T_{i_2} \times_3 \cdots \times_N \mathbf{e}^T_{i_N}, \tag{3.9}$$

where $\mathbf{e}_{i_n}$ is the i_n-th unit vector of length I_n, for $n = 1, \ldots, N$.

Definition 3.1.4. *The* n-mode (vector) product *of a tensor* $\mathbf{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *and a vector* $\mathbf{v} \in \mathbb{R}^{I_n}$*, denoted by* $\mathbf{\mathcal{X}} \,\bar{\times}_n\, \mathbf{v}$*, is a tensor of order* $N - 1$ *and size* $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$*, computed as inner product of each mode-*n *fiber of* $\mathbf{\mathcal{X}}$ *and vector* $\mathbf{v}$*. Element-wise,*

$$(\mathbf{\mathcal{X}} \,\bar{\times}_n\, \mathbf{v})_{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} v_{i_n}.$$

Example 3.1.4. *Let* $\mathcal{X}$ *be the tensor from Example 3.1.1 and let* $\mathbf{v} = [1 \quad 2 \quad 3 \quad 4]^T$. *Then*

$$\mathcal{X} \bar{\times}_1 \mathbf{v} = \begin{bmatrix} 30 & 150 \\ 70 & 190 \\ 110 & 230 \end{bmatrix}.$$

Proposition 3.1.4. *The following property holds*

$$\mathcal{X} \bar{\times}_m \mathbf{v} \bar{\times}_n \mathbf{w} = (\mathcal{X} \bar{\times}_m \mathbf{v}) \bar{\times}_{n-1} \mathbf{w} = (\mathcal{X} \bar{\times}_n \mathbf{w}) \bar{\times}_m \mathbf{v}, \text{ for } m < n.$$

3.2 Tensors in Tucker format and the multilinear rank

As explained in Section 1.1, Tucker format is one of the most commonly used low-rank representations of tensors, particularly suitable for function-related tensors of low order ($N = 3, 4, 5$).

Definition 3.2.1. *A tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *is said to be in* Tucker format *if it can be represented as*

$$\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}, \tag{3.10}$$

were $\mathcal{F} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ *is a tensor called the* core tensor *and* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ *are matrices called the* factor matrices*, with* $R_n \in \mathbb{N}$*, for* $n = 1, 2, \ldots, N$*. Element-wise,*

$$x_{i_1 i_2 \cdots i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_N=1}^{R_N} f_{r_1 r_2 \cdots r_N} a^{(1)}_{i_1 r_1} a^{(2)}_{i_2 r_2} \cdots a^{(N)}_{i_N r_N}.$$

We will also refer to the right-hand side of (3.10) as the *Tucker representation* of $\mathcal{X}$. Integers R_n usually satisfy $R_n \leq I_n$, since then, instead of storing $I_1 I_2 \cdots I_N$ entries of tensor $\mathcal{X}$, we can use its Tucker representation and store only $R_1 R_2 \cdots R_N + \sum_{n=1}^{N} I_n R_n$; see Figure 3.4. However, we still refer to (3.10) as Tucker format, even if $R_n > I_n$, for one or more n.

Example 3.2.1. *Let* $\mathcal{X}$ *be* $100 \times 100 \times 100$ *tensor which can be represented in Tucker format* (3.10) *with* $\mathcal{F}$ *of size* $10 \times 10 \times 10$ *and matrices* $\mathbf{A}^{(n)}$ *of size* 100×10*, for* $n = 1, 2, 3$. *Using this representation, instead of storing* $100^3 = 10^6$ *entries of tensor* $\mathcal{X}$*, we only have to store* $4 \cdot 10^3$ *elements -* 10^3 *elements of* $\mathcal{F}$ *and* $3 \cdot 100 \cdot 10$ *elements for matrices* $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}$.

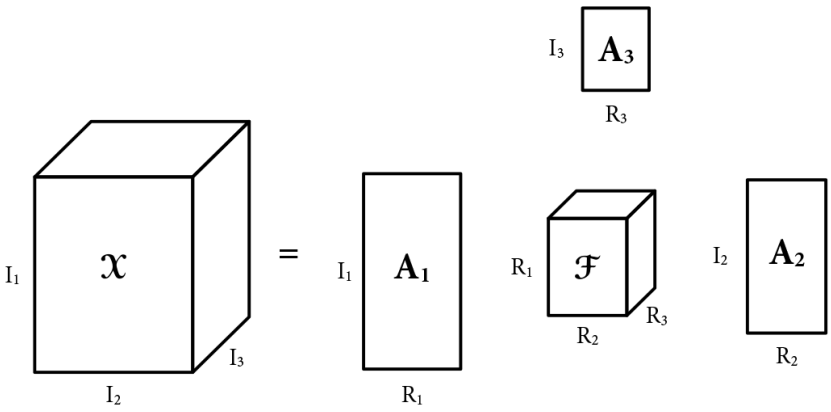


Figure 3.4: Tucker representation of a tensor of order 3.

From (3.8) we see that, in terms of tensors, the SVD of a matrix is exactly its Tucker format, with the diagonal matrix $\mathbf{\Sigma}$ being the core tensor (or the "core matrix") and matrices of left and right singular vectors $\mathbf{U}, \mathbf{V}$ the factor matrices.

However, in general, Tucker representation of a tensor of order N is not unique. For example, let $\mathbf{U} \in \mathbb{R}^{R_1 \times R_1}$ be orthogonal matrix, then from (3.1.3.(3)) follows the equivalence of (3.10) and

$$\mathcal{X} = (\mathcal{F} \times_1 \mathbf{U}) \times_1 \mathbf{A}^{(1)}\mathbf{U}^T \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)} \times_4 \cdots \times_N \mathbf{A}^{(N)}. \tag{3.11}$$

Sometimes, we will want the factor matrices of tensor $\mathcal{X}$ from (3.10) to be orthonormal (or orthogonal, if square), and in that case we can orthogonalize them by performing the truncated QR decomposition (see Section 2.1.6) and update core tensor by property (3.1.3.(3)),

$$\begin{aligned}\mathcal{X} &= \mathcal{F} \times_1 \mathbf{Q}^{(1)}\mathbf{R}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)}\mathbf{R}^{(N)} \\ &= \underbrace{\left(\mathcal{F} \times_1 \mathbf{R}^{(1)} \times_2 \cdots \times_N \mathbf{R}^{(N)}\right)}_{\tilde{\mathcal{F}}} \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)}.\end{aligned} \tag{3.12}$$

Furthermore, addition of two tensors in Tucker format preserves the format. Assume we have two tensors of order $N = 2$ given in their Tucker formats, $\mathbf{X} = \mathbf{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)}$ and $\mathbf{Y} = \mathbf{G} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)}$. From (3.7) follows

$$\begin{aligned}\mathbf{X} + \mathbf{Y} &= \mathbf{A}^{(1)}\mathbf{F}\mathbf{A}^{(2)^T} + \mathbf{B}^{(1)}\mathbf{G}\mathbf{B}^{(2)^T} \\ &= \begin{bmatrix}\mathbf{A}^{(1)} & \mathbf{B}^{(1)}\end{bmatrix} \begin{bmatrix}\mathbf{F} & \\ & \mathbf{G}\end{bmatrix} \begin{bmatrix}\mathbf{A}^{(2)^T} \\ \mathbf{B}^{(2)^T}\end{bmatrix}\end{aligned}$$

$$= \begin{bmatrix} \mathbf{F} & \\ & \mathbf{G} \end{bmatrix} \times_1 \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{B}^{(1)} \end{bmatrix} \times_2 \begin{bmatrix} \mathbf{A}^{(2)} & \mathbf{B}^{(2)} \end{bmatrix},$$

so the factor matrices of $\mathbf{X} + \mathbf{Y}$ are obtained by concatenating factor matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively, and the "core matrix" is a block diagonal matrix with "core matrices" of $\mathbf{X}$ and $\mathbf{Y}$ on the diagonal.

When dealing with tensors of order $N \geq 3$, the generalization is straightforward. Given $\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}$ and $\mathcal{Y} = \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \cdots \times_N \mathbf{B}^{(N)}$, the Tucker representation of $\mathcal{X} + \mathcal{Y}$ is given by

$$\mathcal{X} + \mathcal{Y} = diag\left(\mathcal{F}, \mathcal{G}\right) \times_1 \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{B}^{(1)} \end{bmatrix} \times_2 \cdots \times_N \begin{bmatrix} \mathbf{A}^{(N)} & \mathbf{B}^{(N)} \end{bmatrix}, \tag{3.13}$$

where $diag\left(\mathcal{F}, \mathcal{G}\right)$ denotes a tensor of order N whose two diagonal blocks are $\mathcal{F}$ and $\mathcal{G}$; for $N = 3$ see Figure 3.5. For more details on basic operations of tensors in Tucker format see [23].

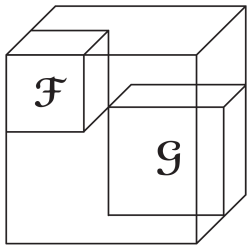


Figure 3.5: Block diagonal tensor of order $N = 3$ with tensors $\mathcal{F}$ and $\mathcal{G}$ on the diagonal.

Two terms are closely related to Tucker representation of a tensor - the n-rank and the multilinear rank.

Definition 3.2.2. *The* n-rank *of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$*, denoted* $rank_n(\mathcal{X})$*, is the rank of its mode-n matricization* $\mathbf{X}_{(n)}$*,*

$$rank_n(\mathcal{X}) = rank(\mathbf{X}_{(n)}).$$

Definition 3.2.3. *The* multilinear rank *of a tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ *is the* N*-tuple of* n*-ranks,*

$$\left(rank_1\left(\mathcal{X}\right), \ldots, rank_N\left(\mathcal{X}\right)\right).$$

For a tensor $\mathcal{X}$ with Tucker representation (3.10), from (3.1.3.(6)) follows that each mode-n matricization of $\mathcal{X}$ can be written as

$$\mathbf{X}_{(n)} = \mathbf{A}^{(n)} \mathbf{F}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right)^T. \tag{3.14}$$

Since $\mathbf{A}^{(n)}$ is an $I_n \times R_n$ matrix, assuming $R_n \leq I_n$ gives $rank_n(\mathcal{X}) \leq R_n$. Vice versa, any tensor can be represented in Tucker format with $R_n = rank_n(\mathcal{X})$. In that case the core tensor is the smallest possible and the storage reduction is maximal.

However, if the multilinear rank of a tensor is too large to achieve significant storage reduction, we can approximate the tensor by a Tucker representation with lower multilinear rank

$$\mathcal{X} \approx \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}, \quad \mathcal{F} \in \mathbb{R}^{R_1 \times \cdots \times R_N}. \tag{3.15}$$

Now the right-hand size of (3.15) is a tensor with multilinear rank $(R_1, \ldots, R_N)$, where $R_n < rank_n(\mathcal{X})$.

The process of obtaining Tucker representation (3.10) of a given tensor is called the *Tucker decomposition*.

3.3 The Higher-order SVD

One way to compute the Tucker decomposition (3.10) of a given tensor is the method known as the *higher-order singular value decomposition (HOSVD)*. In the following, we present the results from [8], which also show that the HOSVD is indeed a convincing generalization of the matrix SVD (2.7), justifying its name.

Theorem 3.3.1. *([8, Theorem 2]) Every tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ can be written as*

$$\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}, \tag{3.10 revisited}$$

in which

(1) $\mathbf{A}^{(n)} = \begin{bmatrix} \mathbf{a}_1^{(n)} & \mathbf{a}_2^{(n)} & \cdots & \mathbf{a}_{I_n}^{(n)} \end{bmatrix} \in \mathbb{R}^{I_n \times I_n}$ *is an orthogonal matrix,*

(2) $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *is a tensor of which the subtensors $\mathcal{F}_{i_n=\alpha}$, obtained by fixing the nth index to α, have the properties of*

(i) *all-orthogonality:*

$$\langle \mathcal{F}_{i_n=\alpha}, \mathcal{F}_{i_n=\beta} \rangle = 0, \text{ when } \alpha \neq \beta,$$

(ii) *ordering:*

$$\|\mathcal{F}_{i_n=1}\|_F \geq \|\mathcal{F}_{i_n=2}\|_F \geq \cdots \geq \|\mathcal{F}_{i_n=I_n}\|_F \geq 0,$$

for all possible values of n.

The norms $\|\boldsymbol{\mathcal{F}}_{i_n=i}\|_F$*, symbolized by* $\sigma_i^{(n)}$*, are* n-mode singular values *and the vectors* $\mathbf{a}_i^{(n)}$ *are* n-mode singular vectors *of* $\boldsymbol{\mathcal{X}}$*.*

Proof. Assume we are given tensors $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{F}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, related by (3.10), in which $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$ are orthogonal matrices. Or, in matrix format

$$\mathbf{X}_{(n)} = \mathbf{A}^{(n)} \mathbf{F}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right)^T. \qquad \text{(3.14 revisited)}$$

Now, consider the particular case when matrix $\mathbf{A}^{(n)}$ is obtained from the SVD of $\mathbf{X}_{(n)}$ as

$$\mathbf{X}_{(n)} = \mathbf{A}^{(n)} \boldsymbol{\Sigma}^{(n)} \mathbf{B}^{(n)T}, \tag{3.16}$$

in which $\mathbf{B}^{(n)}$ is orthogonal and $\boldsymbol{\Sigma}^{(n)} = diag(\sigma_1^{(n)}, \sigma_2^{(n)}, \ldots, \sigma_{I_n}^{(n)})$, with

$$\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \cdots \geq \sigma_{R_n}^{(n)} > \sigma_{R_n+1}^{(n)} = \cdots = \sigma_{I_n}^{(n)} = 0,$$

where $R_n \leq I_n$. Using the Kronecker product property (2.2.1.(3)) and comparing (3.14) and (3.16) shows that

$$\mathbf{F}_{(n)} = \boldsymbol{\Sigma}^{(n)} \mathbf{B}^{(n)T} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right).$$

This equation implies that for arbitrary orthogonal matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(n-1)}, \mathbf{A}^{(n+1)}, \ldots, \mathbf{A}^{(N)}$, matrix $\mathbf{F}_{(n)}$ has orthogonal rows. Since rows of $\mathbf{F}_{(n)}$ are exactly $vec(\boldsymbol{\mathcal{F}}_{i_n=1})$, $vec(\boldsymbol{\mathcal{F}}_{i_n=2}), \ldots, vec(\boldsymbol{\mathcal{F}}_{i_n=I_n})$, we can write the orthogonality property in terms of inner products using (3.1.1.(3))

$$vec(\boldsymbol{\mathcal{F}}_{i_n=\alpha})^T \, vec(\boldsymbol{\mathcal{F}}_{i_n=\beta}) = \langle \boldsymbol{\mathcal{F}}_{i_n=\alpha}, \boldsymbol{\mathcal{F}}_{i_n=\beta} \rangle = 0, \text{ when } \alpha \neq \beta.$$

Also, by (3.1.1.(2))

$$\|\boldsymbol{\mathcal{F}}_{i_n=1}\|_F = \| \, vec(\boldsymbol{\mathcal{F}}_{i_n=1})\|_2 = \| \left(\mathbf{F}_{(n)}\right)_{1:} \|_2 = \sigma_1^{(n)}.$$

By constructing the matrices $\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(n-1)}, \mathbf{A}^{(n+1)}, \cdots, \mathbf{A}^{(N)}$ in the same way as $\mathbf{A}^{(n)}$, $\boldsymbol{\mathcal{F}}$ will satisfy all the conditions from the theorem. So if we calculate every $\mathbf{A}^{(n)}$ from the SVD of $\mathbf{X}_{(n)}$, using (3.1.3.(5-i)) we get $\boldsymbol{\mathcal{F}}$ from

$$\boldsymbol{\mathcal{F}} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \times_3 \cdots \times_N \mathbf{A}^{(N)T}.$$

□

Algorithm 6 HOSVD for computing Tucker decomposition of a tensor $\mathcal{X}$.

1: **procedure** HOSVD($\mathcal{X}$)
2: **for** $n = 1, \ldots, N$ **do**
3: $\mathbf{A}^{(n)} \leftarrow$ left singular vectors of $\mathbf{X}_{(n)}$
4: **end for**
5: $\mathcal{F} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_2 \mathbf{A}^{(2)^T} \times_3 \cdots \times_N \mathbf{A}^{(N)^T}$
6: **return** $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
7: **end procedure**

The preceding proof indicates how to calculate the higher-order singular value decomposition of a given tensor. As presented in Algorithm 6, computing the HOSVD of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ leads to computation of SVDs of $I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$ matrices $\mathbf{X}_{(n)}$, for $n = 1, 2, \ldots, N$.

Remark 3.3.2. *If the HOSVD of $\mathcal{X}$ is given as in Theorem 3.3.1, then*

$$\mathbf{X}_{(n)} = \mathbf{A}^{(n)} \mathbf{\Sigma}^{(n)} \mathbf{B}^{(n)^T}$$

is the SVD of $\mathbf{X}_{(n)}$, where the diagonal matrix $\mathbf{\Sigma}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ and the orthonormal matrix $\mathbf{B}^{(n)} \in \mathbb{R}^{I_1 \cdots I_{n-1} I_{n+1} \cdots I_N \times I_n}$ are defined as

$$\mathbf{\Sigma}^{(n)} = diag\left(\sigma_1^{(n)}, \sigma_2^{(n)}, \ldots, \sigma_{I_n}^{(n)}\right),$$

$$\mathbf{B}^{(n)^T} = \hat{\mathbf{F}}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right)^T,$$

in which $\hat{\mathbf{F}}_{(n)}$ is a normalized version of $\mathbf{F}_{(n)}$,

$$\mathbf{F}_{(n)} = \mathbf{\Sigma}^{(n)} \hat{\mathbf{F}}_{(n)}.$$

If $rank_n(\mathcal{X}) = R_n \leq I_n$, then we can calculate the truncated SVD (2.12) of each matrix $\mathbf{X}_{(n)}$ and get Tucker representation of $\mathcal{X}$ with factor matrices $\mathbf{A}^{(n)}$ of size $I_n \times R_n$ and core tensor of size $R_1 \times R_2 \times \cdots \times R_N$.

Also, we can use HOSVD to get approximation of $\mathcal{X}$ by a tensor of multilinear rank $(R_1, \ldots, R_N)$ (3.15), when $rank_n(\mathcal{X}) > R_n$, if we set columns of each factor matrix $\mathbf{A}^{(n)}$ to contain R_n leading left singular vectors, in which case matrices $\mathbf{A}^{(n)}$ will be orthonormal; we discuss this in details in Corollary 3.3.4. And the process of obtaining such approximation is presented in Algorithm 7 and called the *truncated HOSVD*. It yields the low multilinear rank approximation to a given tensor.

Algorithm 7 Truncated HOSVD for computing approximate Tucker decomposition of a tensor $\mathcal{X}$ with predefined multilinear rank.

procedure HOSVD($\mathcal{X}$, $R_1, R_2, \ldots, R_N$)
  for $n = 1, \ldots, N$ **do**
    $\mathbf{A}^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathbf{X}_{(n)}$
  end for
  $\mathcal{F} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_2 \mathbf{A}^{(2)^T} \times_3 \cdots \times_N \mathbf{A}^{(N)^T}$
  return $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
end procedure

In contrast to the matrix case, where SVD yields the best low-rank approximation (see Theorem 2.1.2) for all unitarily invariant norms, tensor $\hat{\mathcal{X}}$ resulting from the truncated HOSVD is usually not optimal. However it satisfies a quasi-optimal condition

$$\|\mathcal{X} - \hat{\mathcal{X}}\|_F \leq \sqrt{N} \min\Big\{ \|\mathcal{X} - \mathcal{Y}\|_F \;\Big|\; rank_n(\mathcal{Y}) \leq R_n, n = 1, \ldots, N \Big\},$$

which we prove in Corollary 3.3.4, and the truncated HOSVD is a well-established approach to obtain such approximation.

Some properties and consequences of Theorem 3.3.1 are discussed below, for full list of properties see [8].

- In case of tensors of order 3, all-orthogonality means that all horizontal slices (see Figure 3.2) of $\mathcal{F}$ are mutually orthogonal matrices and the same property stands for lateral and frontal slices.
- The n-mode singular values are uniquely defined. When the n-mode singular value is distinct from other n-mode singular values, then its n-mode singular vector is determined up to the sign, while the n-mode singular vectors corresponding to the same n-mode singular value are unique up to the multiplication with an orthogonal matrix, as shown in (3.11).

Corollary 3.3.3. *Let the Tucker representation of $\mathcal{X}$ be given as in Theorem 3.3.1 and let $\mathcal{X}$ have multilinear rank $(R_1, \ldots, R_N)$. Then the following holds*

$$\|\mathcal{X}\|_F^2 = \sum_{i=1}^{R_1} \left(\sigma_i^{(1)}\right)^2 = \cdots = \sum_{i=1}^{R_N} \left(\sigma_i^{(N)}\right)^2 = \|\mathcal{F}\|_F^2.$$

Proof. Directly from (3.1.3.(5-ii)). □

Corollary 3.3.4. *Let the Tucker representation of* $\mathcal{X}$ *be given as in Theorem 3.3.1 and let* $\mathcal{X}$ *have multilinear rank* $(R_1, \ldots, R_N)$*. Define a tensor*

$$\hat{\mathcal{X}} = \hat{\mathcal{F}} \times_1 \hat{\mathbf{A}}^{(1)} \times_2 \cdots \times_N \hat{\mathbf{A}}^{(N)}$$

by discarding the smallest n-mode singular values $\sigma^{(n)}_{I'_n+1}, \sigma^{(n)}_{I'_n+2}, , \ldots, \sigma^{(n)}_{R_n}$ *for given values of* I'_n*, i.e. set* $\hat{\mathcal{F}} = \mathcal{F}[1 : I'_1, 1 : I'_2, \ldots, 1 : I'_N]$ *and* $\hat{\mathbf{A}}^{(n)} = \mathbf{A}^{(n)}[:, 1 : I'_n]$*, for* $n = 1, \ldots, N$*. Then*

(1) $\langle \mathcal{X}, \hat{\mathcal{X}} \rangle = \|\hat{\mathcal{F}}\|_F^2$,

(2) $\|\mathcal{X} - \hat{\mathcal{X}}\|_F^2 \leq \sum_{n=1}^{N} \sum_{i_n=I'_n+1}^{R_n} {\sigma^{(n)}_{i_n}}^2 \leq N \min_{\substack{rank_n(\mathcal{Y}) \leq I'_n, \\ n=1,\ldots,N}} \|\mathcal{X} - \mathcal{Y}\|_F^2.$

Proof. *(1)* By (3.1.3.(3)) and (3.1.3.(4))

$$\begin{aligned}
\langle \mathcal{X}, \hat{\mathcal{X}} \rangle =& \langle \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}, \hat{\mathcal{F}} \times_1 \hat{\mathbf{A}}^{(1)} \times_2 \cdots \times_N \hat{\mathbf{A}}^{(N)} \rangle \\
=& \langle \left(\mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)} \right) \times_1 {\hat{\mathbf{A}}^{(1)}}^T \times_2 \cdots \times_N {\hat{\mathbf{A}}^{(N)}}^T, \hat{\mathcal{F}} \rangle \\
=& \langle \mathcal{F} \times_1 {\hat{\mathbf{A}}^{(1)}}^T \mathbf{A}^{(1)} \times_2 \cdots \times_N {\hat{\mathbf{A}}^{(N)}}^T \mathbf{A}^{(N)}, \hat{\mathcal{F}} \rangle \\
=& \langle \hat{\mathcal{F}}, \hat{\mathcal{F}} \rangle = \|\hat{\mathcal{F}}\|_F^2.
\end{aligned}$$

(2) From (3.3.4.(1)), (3.1.1.(4)) and Corollary 3.3.3 we have

$$\begin{aligned}
\|\mathcal{X} - \hat{\mathcal{X}}\|_F^2 =& \|\mathcal{X}\|_F^2 - 2\langle \mathcal{X}, \hat{\mathcal{X}} \rangle + \|\hat{\mathcal{X}}\|_F^2 = \|\mathcal{F}\|_F^2 - \|\hat{\mathcal{F}}\|_F^2 \\
=& \sum_{i_1=1}^{R_1} \sum_{i_2=1}^{R_2} \cdots \sum_{i_N=1}^{R_N} f^2_{i_1 i_2 \cdots i_N} - \sum_{i_1=1}^{I'_1} \sum_{i_2=1}^{I'_2} \cdots \sum_{i_N=1}^{I'_N} f^2_{i_1 i_2 \cdots i_N} \\
=& \sum_{i_1=I'_1+1}^{R_1} \sum_{i_2=I'_2+1}^{R_2} \cdots \sum_{i_N=I'_N+1}^{R_N} f^2_{i_1 i_2 \cdots i_N} \\
\leq& \sum_{i_1=I'_1+1}^{R_1} \sum_{i_2=1}^{R_2} \cdots \sum_{i_N=1}^{R_N} f^2_{i_1 i_2 \cdots i_N} + \sum_{i_1=1}^{R_1} \sum_{i_2=I'_2+1}^{R_2} \cdots \sum_{i_N=1}^{R_N} f^2_{i_1 i_2 \cdots i_N} \\
& + \cdots + \sum_{i_1=1}^{R_1} \sum_{i_2=1}^{R_2} \cdots \sum_{i_N=I'_N+1}^{R_N} f^2_{i_1 i_2 \cdots i_N} \\
=& \sum_{i_1=I'_1+1}^{R_1} {\sigma^{(1)}_{i_1}}^2 + \sum_{i_2=I'_2+1}^{R_2} {\sigma^{(2)}_{i_2}}^2 + \cdots + \sum_{i_N=I'_N+1}^{R_N} {\sigma^{(N)}_{i_N}}^2.
\end{aligned}$$

This proves the first inequality, while the second one follows from the fact that

$$\sum_{i_n=I'_n+1}^{R_n} {\sigma^{(n)}_{i_n}}^2 = \|\mathcal{X} - \mathcal{X} \times_n \mathbf{P}_n\|_F^2,$$

where $\mathbf{P}_n = \hat{\mathbf{A}}^{(n)}\hat{\mathbf{A}}^{(n)^T}$ is an orthogonal projector. Namely, setting $\hat{\mathcal{F}}_n = \mathcal{F}_{r_n=1:I'_n}$,

$$\begin{aligned}\|\mathcal{X} - \mathcal{X} \times_n \mathbf{P}_n\|_F^2 =& \|\mathcal{F} - \hat{\mathcal{F}}_n\|_F^2 = \|\mathcal{F}\|_F^2 + \|\hat{\mathcal{F}}_n\|_F^2 - 2\langle \mathcal{F}, \hat{\mathcal{F}}_n\rangle \\ &= \sum_{i_n=1}^{R_n} \|\mathcal{F}_{r_n=i_n}\|_F^2 + \sum_{i_n=1}^{I'_n} \|\mathcal{F}_{r_n=i_n}\|_F^2 - 2\sum_{i_n=1}^{I'_n} \|\mathcal{F}_{r_n=i_n}\|_F^2 \\ &= \sum_{i_n=I'_n+1}^{R_n} \|\mathcal{F}_{r_n=i_n}\|_F^2.\end{aligned}$$

Now, for any $\mathcal{Y}$ with $rank_n(\mathcal{Y}) \leq I'_n$, for $n = 1, \ldots, N$,

$$\begin{aligned}\|\mathcal{X} - \mathcal{Y}\|_F^2 = \|\mathbf{X}_{(n)} - \mathbf{Y}_{(n)}\|_F^2 &= \|\mathbf{X}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)} - \left(\mathbf{Y}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\right)\|_F^2 \\ &= \|\mathbf{X}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\|_F^2 + \|\mathbf{Y}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\|_F^2,\end{aligned}$$

which follows from (3.1.1.(4)) and $\left(\mathbf{X}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\right)^T \left(\mathbf{Y}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\right) = 0$. This gives

$$\|\mathcal{X} - \mathcal{X} \times_n \mathbf{P}_n\|_F^2 = \|\mathbf{X}_{(n)} - \mathbf{P}_n\mathbf{X}_{(n)}\|_F^2 \leq \|\mathcal{X} - \mathcal{Y}\|_F^2,$$

which completes the proof.

□

Instead of specifying the desired multilinear rank of the approximate Tucker representation, we can also specify tolerance $\epsilon > 0$ and discard singular vectors whose associated singular values are lower than ϵ. The procedure is desribed in Algorithm 8.

Algorithm 8 Truncated HOSVD for computing approximate Tucker decomposition of a tensor $\mathcal{X}$ with given tolerance.

procedure HOSVD($\mathcal{X}$, ϵ)
  for $n = 1, \ldots, N$ **do**
    $\mathbf{A}^{(n)} \leftarrow$ leading left singular vectors of $\mathbf{X}_{(n)}$ whose associated singular values are greater or equal to ϵ
  end for
  $\mathcal{F} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_2 \mathbf{A}^{(2)^T} \times_3 \cdots \times_N \mathbf{A}^{(N)^T}$
  return $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
end procedure

The HOSVD as generalization of the SVD

HOSVD shows a clear analogy with matrix SVD. In Section 3.2 we have explained that getting the SVD of a matrix actually provides its Tucker representation in terms of tensors. Furthermore, in Theorem 3.3.1 the left and right singular vectors of a matrix are generalized as the n-mode singular vectors and the role of the singular values is taken over by the norms of the $(N-1)$th-order subtensors of the core tensor. Notice that in the matrix case (3.8), the singular values also correspond to the norm of the rows and the columns of the "core matrix" $\mathbf{\Sigma}$.

The essential difference is that that $\mathcal{F}$ is a full tensor and not diagonal, but instead $\mathcal{F}$ obeys the condition of all-orthogonality. In general it is impossible to reduce higher-order tensors to a diagonal form by means of orthogonal transformations, which is easily shown by counting degrees of freedom: diagonality of core tensor containing I nonzero elements would imply that the decomposition would exhibit not more than $I(\sum_{n=1}^{N} I_n + 1 - N(I+1)/2)$ degrees of freedom, while the original tensor contains $I_1 I_2 \cdots I_N$ independent entries. Only in the second-order case both quantities are equal for $I = \min\{I_1, I_2\}$. However, notice that in the matrix case (3.8) $\mathbf{\Sigma}$ is all-orthogonal as well; due to the diagonal structure, the scalar product of two different rows and columns also vanishes. Also, the n-mode singular values are by definition non-negative and real, like in the matrix case.

3.4 HOSVD based on approximate ranges

Here we present a modification of the HOSVD algorithm from Section 3.3, which, for a given $\mathcal{X}$, gets its factor matrices by approximating an orthonormal basis for the range of $\mathbf{X}_{(n)}$, instead of calculating its leading left singular vectors. Afterward, the core tensor is calculated the usual way. The procedure is presented in Algorithm 9 and we refer to it as HOSVD-AR.

Corollary 3.4.1. *For $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, let its Tucker representation be given by Algorithm 9, with step 3 obtained such that for each $n = 1, 2, \ldots, N$,*

$$\| \left(\mathbf{I} - \mathbf{A}^{(n)} \mathbf{A}^{(n)^T} \right) \mathbf{X}_{(n)} \|_F \leq \varepsilon, \tag{3.17}$$

for some $\varepsilon > 0$. Then

$$\| \mathcal{X} - \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)} \|_F \leq N\varepsilon.$$

Algorithm 9 HOSVD for computing Tucker decomposition of a tensor $\mathcal{X}$ based on approximate ranges.

1: **procedure** HOSVD-AR($\mathcal{X}$)
2: **for** $n = 1, 2, \ldots, N$ **do**
3: $\mathbf{A}^{(n)} \leftarrow$ orthonormal basis for approximation of range of $\mathbf{X}_{(n)}$
4: **end for**
5: $\mathcal{F} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_2 \mathbf{A}^{(2)^T} \times_3 \cdots \times_N \mathbf{A}^{(N)^T}$
6: **return** $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
7: **end procedure**

Proof. Setting $\mathbf{P}_n = \mathbf{A}^{(n)}\mathbf{A}^{(n)^T}$, we rewrite $\mathcal{X}$ as

$$\begin{aligned}\mathcal{X} &= \mathcal{X} \times_1 \mathbf{P}_1 + \mathcal{X} \times_1 (\mathbf{I} - \mathbf{P}_1) \\ &= \mathcal{X} \times_1 \mathbf{P}_1 \times_2 \mathbf{P}_2 + \mathcal{X} \times_1 \mathbf{P}_1 \times_2 (\mathbf{I} - \mathbf{P}_2) + \mathcal{X} \times_1 (\mathbf{I} - \mathbf{P}_1) = \cdots \\ &= \mathcal{X} \times_1 \mathbf{P}_1 \times_2 \cdots \times_N \mathbf{P}_N + \sum_{n=1}^{N} \mathcal{X} \times_1 \mathbf{P}_1 \times_2 \cdots \times_{n-1} \mathbf{P}_{n-1} \times_n (\mathbf{I} - \mathbf{P}_n).\end{aligned}$$

Now,

$$\begin{aligned}&\|\mathcal{X} - \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}\|_F \\ &\quad = \|\mathcal{X} - \left(\mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_1 \cdots \times_N \mathbf{A}^{(N)^T}\right) \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}\|_F \\ &\quad = \|\mathcal{X} - \mathcal{X} \times_1 \mathbf{P}_1 \times_2 \cdots \times_N \mathbf{P}_N\|_F \\ &\quad \leq \sum_{n=1}^{N} \|\mathcal{X} \times_1 \mathbf{P}_1 \times_2 \cdots \times_{n-1} \mathbf{P}_{n-1} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F \\ &\quad \leq \sum_{n=1}^{N} \|\mathcal{X} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F \leq N\varepsilon,\end{aligned}$$

where the first inequality in the last line follows from (2.18). □

For step 3 of Algorithm 9, we use Stage A of the randomized algorithm (see Section 2.3.2).[1] Given ε, (3.17) can be achieved by setting $\epsilon = \varepsilon/\sqrt{I_n}$ in Algorithm 3, assuming $I_n \leq \prod_{\substack{k=1 \\ k \neq n}}^{N} I_k$; see (2.47) and (2.11).

However, when adapting Algorithm 9 to give approximation of $\mathcal{X}$ of predefined multilinear rank in combination with randomized algorithm for fixed range (Algorithm 2), the required oversampling parameter will lead to a basis that is unnecessarily large. We

[1] This combination is presented in [7] under name HOSVD-RandSVD.

mitigate this effect by recompressing the core tensor. Algorithm 10 contains the truncated version of the algorithm.

Algorithm 10 Truncated HOSVD for computing Tucker decomposition of a tensor $\mathcal{X}$ based on approximate ranges.

1: **procedure** HOSVD-AR($\mathcal{X}$, $R_1, R_2, \ldots, R_N$)
2: **for** $n = 1, 2, \ldots, N$ **do**
3: Apply Algorithm 2: $\mathbf{Q}^{(n)} = \text{FIXED\_RAND\_RANGE}(\mathbf{X}_{(n)}, R_n)$
4: **end for**
5: $\tilde{\mathcal{F}} \leftarrow \mathcal{X} \times_1 \mathbf{Q}^{(1)^T} \times_2 \mathbf{Q}^{(2)^T} \times_3 \cdots \times_N \mathbf{Q}^{(N)^T}$
6: Apply Algorithm 7: $(\mathcal{F}, \tilde{\mathbf{A}}^{(1)}, \tilde{\mathbf{A}}^{(2)}, \ldots, \tilde{\mathbf{A}}^{(N)}) = \text{HOSVD}(\tilde{\mathcal{F}}, R_1, \ldots, R_N)$
7: **for** $n = 1, 2, \ldots, N$ **do**
8: $\mathbf{A}^{(n)} \leftarrow \mathbf{Q}^{(n)} \tilde{\mathbf{A}}^{(n)}$
9: **end for**
10: **return** $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
11: **end procedure**

The additional step changes the error bound from Corollary 3.4.1.

Corollary 3.4.2. *For $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, let its Tucker representation be given by Algorithm 10, with step 3 obtained such that for each $n = 1, 2, \ldots, N$,*

$$\| \left(\mathbf{I} - \mathbf{Q}^{(n)} \mathbf{Q}^{(n)^T} \right) \mathbf{X}_{(n)} \|_F \leq \varepsilon$$

and by requiring the 2-norm of the truncated n-mode singular values of $\tilde{\mathcal{F}}$ not to be larger than ε, for some $\varepsilon > 0$. Then

$$\|\mathcal{X} - \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}\|_F \leq (N + \sqrt{N})\varepsilon.$$

Proof. Setting $\tilde{\mathcal{X}} = \tilde{\mathcal{F}} \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)}$, Corollary 3.4.1 gives

$$\|\mathcal{X} - \tilde{\mathcal{X}}\|_F \leq N\varepsilon,$$

while Corollary 3.3.4 gives

$$\|\tilde{\mathcal{F}} - \mathcal{F} \times_1 \tilde{\mathbf{A}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{A}}^{(N)}\|_F \leq \sqrt{N}\varepsilon.$$

Now, using (3.1.3.(3)) and (3.1.3.(5-ii)),

$$\|\mathcal{X} - \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}\|_F$$

$$\leq \|\mathcal{X} - \tilde{\mathcal{X}}\|_F + \|\tilde{\mathcal{X}} - \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}\|_F$$
$$\leq \|\mathcal{X} - \tilde{\mathcal{X}}\|_F + \|\tilde{\mathcal{F}} - \mathcal{F} \times_1 \tilde{\mathbf{A}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{A}}^{(N)}\|_F \leq (N + \sqrt{N})\varepsilon.$$

□

If $\mathcal{X}$ has multilinear rank $(R_1, \ldots, R_N)$, from (2.11) we know that

$$\| \left(\mathbf{I} - \mathbf{A}^{(n)} \mathbf{A}^{(n)^T} \right) \mathbf{X}_{(n)} \|_F \leq \sqrt{R_n} \| \left(\mathbf{I} - \mathbf{A}^{(n)} \mathbf{A}^{(n)^T} \right) \mathbf{X}_{(n)} \|_2,$$

while Theorem 2.3.1 gives the expectation for $\| \left(\mathbf{I} - \mathbf{A}^{(n)} \mathbf{A}^{(n)^T} \right) \mathbf{X}_{(n)} \|_2$.

3.5 Efficient computation of the tensor operations

In this section we explain the connection between matricized Kronecker product of tensors and the Kronecker product of matricized tensors, and discuss the efficient way to compute the matrix-vector multiplication, when matrix is exactly matricized Kronecker product of tensors. Furthermore, we explain how to efficiently create tensor if we are only given its Tucker representation and we calculate the complexity of such procedure, together with presenting complexity of the HOSVD algorithm and a way to improve it. From these observations we have created Julia procedures for solving the presented problems; for details see Appendix A.

3.5.1 Matricization of Kronecker products of tensors

Given two tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, we will be interested in matrix-vector products

$$(\mathcal{X} \otimes \mathcal{Y})_{(n)} \mathbf{v} \text{ and } (\mathcal{X} \otimes \mathcal{Y})_{(n)}^T \mathbf{w}, \tag{3.18}$$

where $\mathbf{v}$ and $\mathbf{w}$ are vectors of appropriate size. Matrix $(\mathcal{X} \otimes \mathcal{Y})_{(n)}$ is the mode-n matricization of the Kronecker product of tensors and

$$(\mathcal{X} \otimes \mathcal{Y})_{(n)}^T \equiv \Big[(\mathcal{X} \otimes \mathcal{Y})_{(n)} \Big]^T$$

its transposition. Creating matrix $(\mathcal{X} \otimes \mathcal{Y})_{(n)}$ directly requires first forming an $I_1 J_1 \times I_2 J_2 \times \cdots \times I_N J_N$ tensor $\mathcal{X} \otimes \mathcal{Y}$ and then matricizing it into an $I_n J_n \times \prod_{\substack{k=1 \\ k \neq n}}^{N} I_k J_k$ matrix.

The problem here is that this matrix can be too large to store. But if we could substitute $(\mathcal{X} \otimes \mathcal{Y})_{(n)}$ with $\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)}$, then we could just apply the property (2.2.1.(5))

and avoid explicitly forming the Kronecker product $\mathcal{X} \otimes \mathcal{Y}$, as explained in Section 2.2.1. So the question is - what is the relation between $(\mathcal{X} \otimes \mathcal{Y})_{(n)}$ and $\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)}$?

The matricization of $\mathcal{X} \otimes \mathcal{Y}$ is in general *not* equal to the Kronecker product of matricizations of $\mathcal{X}$ and $\mathcal{Y}$, but is somewhat similar.

Lemma 3.5.1. *For two tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, there exists a permutation matrix $\mathbf{P}_n$, only depending on the sizes of $\mathcal{X}$, $\mathcal{Y}$ and the mode $n \in \{1, 2, \ldots, N\}$ of the matricization, such that*

$$(\mathcal{X} \otimes \mathcal{Y})_{(n)} = \left(\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)}\right) \mathbf{P}_n. \tag{3.19}$$

In [24, 25] this statement is presented and proved through block tensors and block matricizations. We offer alternative, for our purpose simpler proof.

Proof. The Kronecker product of two tensors is a tensor that consists of all possible combinations of elements of these two tensors and the mode-n matricization just rearranges the elements of the tensor into a matrix.

Let us denote elements of tensor $\mathcal{X}$ as $x_{i_1 i_2 \cdots i_N}$ and elements of $\mathcal{Y}$ as $y_{j_1 j_2 \cdots j_N}$, and left and right matrices from (3.19) as

$$\mathbf{L}_n = (\mathcal{X} \otimes \mathcal{Y})_{(n)}, \quad \mathbf{R}_n = \left(\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)}\right).$$

Then $\mathbf{L}_n$ and $\mathbf{R}_n$ are obviously of the same size and elements of both matrices are exactly $x_{i_1 i_2 \cdots i_N} y_{j_1 j_2 \cdots j_N}$, for some set of indices $\{(i_1, \ldots, i_N), (j_1, \ldots, j_N)\}$. Now we will compare how the elements of $\mathcal{X}$ and $\mathcal{Y}$ map on matrices $\mathbf{L}_n$ and $\mathbf{R}_n$.

As defined in (3.1), the elements of tensor $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ are $z_{k_1 k_2 \cdots k_N} = x_{i_1 i_2 \cdots i_N} y_{j_1 j_2 \cdots j_N}$, where $k_n = j_n + (i_n - 1)J_n$, for $n = 1, \ldots, N$. Performing mode-n matricization on $\mathcal{Z}$ maps its elements in the following way (see Definition 3.1.1),

$$\begin{aligned}
(k_1, k_2, \ldots, k_N) &\longrightarrow \left(k_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (k_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m J_m \right) \\
&= \left(j_n + (i_n - 1)J_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} \Big[j_l + (i_l - 1)J_l - 1 \Big] \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m J_m \right) \\
&= \left(j_n + (i_n - 1)J_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} \Big[(j_l - 1) + (i_l - 1)J_l \Big] \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m J_m \right),
\end{aligned}$$

and this gives us the rule how elements of $\mathcal{X}$ and $\mathcal{Y}$ map into elements of matrix $\mathbf{L}_n$.

On the other hand, by first performing the n-mode matricization on tensors $\mathcal{X}$ and $\mathcal{Y}$ with mappings

$$(i_1, i_2, \ldots, i_N) \longrightarrow \left(i_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (i_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m \right) \quad \text{for } \mathbf{X}_{(n)}, \text{ and}$$

$$(j_1, j_2, \ldots, j_N) \longrightarrow \left(j_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (j_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} J_m \right) \quad \text{for } \mathbf{Y}_{(n)},$$

and then multiplying $\mathbf{X}_{(n)} = (x_{l_1 l_2})$ and $\mathbf{Y}_{(n)} = (y_{m_1 m_2})$ by the Kronecker product, we get $\left[\mathcal{X}_{(n)} \otimes \mathcal{Y}_{(n)}\right]_{k_1 k_2} = x_{l_1 l_2} y_{m_1 m_2}$, where

$$\begin{aligned} k_1 &= m_1 + (l_1 - 1) J_n, \\ k_2 &= m_2 + (l_2 - 1) J_1 J_2 \ldots J_{n-1} J_{n+1} \ldots J_N. \end{aligned}$$

Including previous conclusions we get the rule for mapping elements of $\mathcal{X}$ and $\mathcal{Y}$ into elements of matrix $\mathbf{R}_n$:

$$\begin{aligned} &(k_1, k_2) = (m_1 + (l_1 - 1) J_n, m_2 + (l_2 - 1) J_1 J_2 \ldots J_{n-1} J_{n+1} \ldots J_N) \\ &= \left(j_n + (i_n - 1) J_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (j_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} J_m + \left[1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} (i_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m - 1 \right] \prod_{\substack{m=1 \\ m \neq n}}^{N} J_m \right) \\ &= \left(j_n + (i_n - 1) J_n, 1 + \sum_{\substack{l=1 \\ l \neq n}}^{N} \left[(j_l - 1) + (i_l - 1) \prod_{\substack{m=1 \\ m \neq n}}^{l-1} I_m \prod_{\substack{m=l \\ m \neq n}}^{N} J_m \right] \prod_{\substack{m=1 \\ m \neq n}}^{l-1} J_m \right). \end{aligned}$$

Comparing mappings for $\mathbf{L}_n$ and $\mathbf{R}_n$, we see that the row indices are identical but the column indices are not. In order to prove that the matrices are equal up to the column permutation, we still have to show that if two pairs of indices map to the same column in $\mathbf{L}_n$, they also map to the same column in $\mathbf{R}_n$.

Matrix $\mathbf{L}_n$ is the mode-n matricization of tensor $\mathcal{Z}$, so the columns of $\mathbf{L}_n$ are the mode-n fibers of $\mathcal{Z}$, i.e. vectors obtained by fixing every index of $\mathcal{Z}$ but nth, meaning that the two set of indices $((i_1, \ldots, i_N), (j_1, \ldots, j_N))$ and $((i'_1, \ldots, i'_N), (j'_1, \ldots, j'_N))$ will map to the same column of $\mathbf{L}_n$ only if $i_l = i'_l$ and $j_l = j'_l$, for every $l \neq n$. Since neither column mappings include indices i_n and j_n, the equality of column indices easily proves. □

Using Lemma 3.5.1, from (2.2.1.(5)) and (2.2.1.(1)) follows

$$(\mathcal{X} \otimes \mathcal{Y})_{(n)} \mathbf{v} = \left(\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)}\right) \underbrace{\mathbf{P}_n \mathbf{v}}_{\tilde{\mathbf{v}}} = vec\left(\mathbf{Y}_{(n)} \tilde{\mathbf{V}} \mathbf{X}_{(n)}^T\right), \quad \tilde{\mathbf{v}} = vec(\tilde{\mathbf{V}}),$$

and

$$(\mathcal{X} \otimes \mathcal{Y})_{(n)}^T \mathbf{w} = \mathbf{P}_n^T \left(\mathbf{X}_{(n)}^T \otimes \mathbf{Y}_{(n)}^T\right) \mathbf{w} = \mathbf{P}_n^T \underbrace{vec\left(\mathbf{Y}_{(n)}^T \mathbf{W} \mathbf{X}_{(n)}\right)}_{\tilde{\mathbf{w}}}, \quad \mathbf{w} = vec(\mathbf{W}),$$

and the problem of computing (3.18) comes down to correctly permuting vectors $\mathbf{v}$ and $\tilde{\mathbf{w}}$, both of length $I_1 J_1 \cdots I_{n-1} J_{n-1} I_{n+1} J_{n+1} \cdots I_N J_N$ and applying computations from above.

So we do not need an explicit expression for $\mathbf{P}_n$, which is tedious anyway, but only the realization of a matrix-vector product, when the matrix is either $\mathbf{P}_n$ or $\mathbf{P}_n^T$. Assuming now $\mathbf{v}$ is any vector of length $I_1 J_1 \cdots I_{n-1} J_{n-1} I_{n+1} J_{n+1} \cdots I_N J_N$, we explain how to calculate $\mathbf{P}_n \mathbf{v}$ and $\mathbf{P}_n^T \mathbf{v}$.

To get $\mathbf{P}_n \mathbf{v}$, we first reshape $\mathbf{v}$ into a tensor $\boldsymbol{\mathcal{V}}$ of size $J_1 \times I_1 \times \cdots \times J_{n-1} \times I_{n-1} \times J_{n+1} \times I_{n+1} \times \cdots \times J_N \times I_N$, which matches the structure of a row on the left-hand side of (3.19). In order to match the structure of a row on the right-hand side of (3.19), we need to apply the perfect shuffle permutation

$$\pi_n = \begin{bmatrix} 1 & 3 & 5 & \cdots & (2N-3) & 2 & 4 & \cdots & (2N-2) \end{bmatrix} \tag{3.20}$$

to the modes of $\boldsymbol{\mathcal{V}}$, that is, $\tilde{\boldsymbol{\mathcal{V}}}(i_1, \ldots, i_{2N-2}) = \boldsymbol{\mathcal{V}}(\pi_n(i_1), \ldots, \pi_n(i_{2N-2}))$. The vectorization of $\tilde{\boldsymbol{\mathcal{V}}}$ yields $\mathbf{P}_n \mathbf{v}$.

In Julia, the above procedure can be easily implemented using the commands `reshape` and `permutedims` for reshaping and permuting the modes of a multivariate array, respectively.

```
perfect_shuffle = [ [2*k-1 for k=1:N-1]; [2*k for k=1:N-1] ]
tenshape = vec([J[setdiff([1:N],n)] I[setdiff([1:N],n)]]')
w = vec(permutedims(reshape(v,tenshape),perfect_shuffle))
```

A matrix-vector product $\mathbf{P}_n^T \mathbf{v}$ is computed in an analogous fashion. First, $\mathbf{v}$ is reshaped into a tensor $\boldsymbol{\mathcal{V}}$ of size $J_1 \times \cdots \times J_{n-1} \times J_{n+1} \times \cdots \times J_N \times I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. After applying the inverse permutation of (3.20) to the modes of $\boldsymbol{\mathcal{V}}$, the vectorization of this tensor yields $\mathbf{P}_n^T \mathbf{v}$.

```
tenshape=[J[setdiff([1:N],n)];I[setdiff([1:N],n)]]
vec(permutedims(reshape(w,tenshape),invperm(perfect_shuffle)))
```

3.5.2 The n-mode multiplication

Now we explain how to efficiently perform the n-mode multiplication when multiplying a tensor by a matrix in each mode. This procedure is equivalent with the process of creating full tensor out of its Tucker representation.

Given the core tensor $\mathcal{F} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and the factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ from Tucker representation of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ (3.10), we create (full) tensor $\mathcal{X}$ by multiplying the core tensor by the factor matrices one by one,

$$\mathcal{X} = \underbrace{\underbrace{\mathcal{F} \times_1 \mathbf{A}^{(1)}}_{\mathcal{X}^{(1)}} \times_2 \mathbf{A}^{(2)}}_{\mathcal{X}^{(2)\ldots}} \times_3 \cdots \times_N \mathbf{A}^{(N)}. \tag{3.21}$$

Since order of the multiplication is not important (see (3.1.3.(2))), we perform multiplication in such order that tensors $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots \mathcal{X}^{(N-1)}$ are smallest possible, meaning we reorder modes $\{1, 2, \ldots, N\}$ such that values $I_n - R_n$ are in descending order.

For simplicity let us assume that that order is exactly $\{1, 2, \ldots, N\}$. First, we form $I_1 \times R_2 \times R_3 \times \cdots \times R_N$ tensor $\mathcal{X}^{(1)}$ from its mode-1 matricization

$$\mathbf{X}^{(1)}_{(1)} = \underbrace{\mathbf{A}^{(1)}}_{I_1 \times R_1} \quad \underbrace{\mathbf{F}_{(1)}}_{R_1 \times \prod_{n=2}^{N} R_n}$$

in $\mathcal{O}(I_1 R_1 R_2 \cdots R_N)$ operations. Then, we form $I_1 \times I_2 \times R_3 \times R_4 \times \cdots \times R_N$ tensor $\mathcal{X}^{(2)}$ from its mode-2 matricization as

$$\mathbf{X}^{(2)}_{(2)} = \underbrace{\mathbf{A}^{(2)}}_{I_2 \times R_2} \quad \underbrace{\mathbf{X}^{(1)}_{(2)}}_{R_2 \times I_1 \prod_{n=3}^{N} R_n}$$

in $\mathcal{O}(I_1 I_2 R_2 R_3 \cdots R_N)$ operations. We continue in the similar fashion:

- $I_1 \times I_2 \times I_3 \times R_4 \times \cdots \times R_N$ tensor $\mathcal{X}^{(3)}$ in $\mathcal{O}(I_1 I_2 I_3 R_3 R_4 \cdots R_N)$ operations,
- $I_1 \times \cdots \times I_4 \times R_5 \times \cdots \times R_N$ tensor $\mathcal{X}^{(4)}$ in $\mathcal{O}(I_1 I_2 I_3 I_4 R_4 \cdots R_N)$ operations,
- $\cdots$
- $I_1 \times I_2 \times \cdots \times I_{N-1} \times R_N$ tensor $\mathcal{X}^{(N-1)}$ in $\mathcal{O}(I_1 \cdots I_{N-1} R_{N-1} R_N)$ operations,
- $I_1 \times I_2 \times \cdots \times I_N$ tensor $\mathcal{X}$ in $\mathcal{O}(I_1 \cdots I_N R_N)$ operations.

So the overall complexity of (3.21) is

$$\mathcal{O}\left(\sum_{n=1}^{N}\prod_{i=1}^{n} I_i \prod_{j=n}^{N} R_j\right) \text{ operations and } \mathcal{O}\left(\sum_{n=1}^{N}\prod_{i=1}^{n} I_i \prod_{j=n+1}^{N} R_j\right) \text{ memory.} \tag{3.22}$$

Remark 3.5.2. *We will be interested in the simplified case when $\mathcal{X}$ is an $I \times \cdots \times I$ tensor of order N, with given core tensor $\mathcal{F}$ of size $R \times \cdots \times R$ and $I \times R$ factor matrices $\mathbf{A}^{(n)}$, for $n = 1, \ldots, N$, with additional assumption $R < I$. Setting $I = I_n$, $R = R_n$, for $n = 1, \ldots, N$, from* (3.22) *follows the complexity of $\mathcal{O}(I^N R)$ operations and $\mathcal{O}(I^N)$ memory. Further simplification on tensors of order $N = 3$ gives the complexity of $\mathcal{O}(I^3 R)$ operations and $\mathcal{O}(I^3)$ memory.*

3.5.3 The HOSVD

Assume we are given $I_1 \times \cdots \times I_N$ tensor $\mathcal{X}$ whose multilinear rank is $(R_1, \ldots, R_N)$ and we want to run the truncated HOSVD algorithm (Algorithm 7) to get its Tucker representation (3.10) with $R_1 \times \cdots \times R_N$ core tensor $\mathcal{F}$ and $I_n \times R_n$ factor matrices $\mathbf{A}^{(n)}$, for $n = 1, 2, \ldots, N$.

First, with assumption $I_n \leq I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$, calculating R_n left singular vectors of $I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$ mode-n matricizations $\mathbf{X}_{(n)}$ by getting their SVDs using standard methods requires $\mathcal{O}(I_1 \cdots I_{n-1} I_n^2 I_{n+1} \cdots I_N)$ operations (see Section 2.1.3), but this can be reduced to $\mathcal{O}(I_1 \cdots I_{n-1} I_n I_{n+1} \cdots I_N R_n)$ by using iterative methods, for example randomized algorithm presented in Section 2.3.2. In that case, forming nth factor matrix requires $\mathcal{O}(I_1 \cdots I_{n-1} I_{n+1} \cdots I_N R_n)$ memory.

Creating core tensor goes analogously to process explained in Section 3.5.2, with the complexity (3.22).

All together, the complexity of the truncated HOSVD combined with the randomized algorithm is

$$\begin{aligned} &\mathcal{O}\left(\max_{n=1,\ldots,N} R_n \cdot \prod_{n=1}^{N} I_n + \sum_{n=1}^{N-1}\prod_{i=1}^{n} I_i \prod_{j=n}^{N} R_j\right) \text{ operations and} \\ &\mathcal{O}\left(\max_{n=1,\ldots,N} R_n \cdot \prod_{\substack{i=1 \\ i\neq n}}^{N} I_i + \sum_{n=1}^{N}\prod_{i=1}^{n} I_i \prod_{j=n+1}^{N} R_j\right) \text{ memory.} \end{aligned} \tag{3.23}$$

Remark 3.5.3. *When $\mathcal{X}$ is an $I \times \cdots \times I$ tensor with multilinear rank $(R, \ldots, R)$ and $R < I$, the Tucker representation of $\mathcal{X}$ obtained by the truncated HOSVD algorithm will*

include core tensor $\mathfrak{F}$ *of size* $R \times \cdots \times R$ *and* $I \times R$ *factor matrices* $\mathbf{A}^{(n)}$*, for* $n = 1, \ldots, N$*. Setting* $I = I_n$*,* $R = R_n$*, for* $n = 1, \ldots, N$*, from* (3.23) *follows that the computational requirement for this simplified case is* $\mathcal{O}(I^N R)$ *operations and* $\mathcal{O}(I^N)$ *memory. Further simplification to tensors of order* $N = 3$ *gives the complexity of* $\mathcal{O}(I^3 R)$ *operations and* $\mathcal{O}(I^3)$ *memory.*

Chapter 4

Recompression of Hadamard products by HOSVD

In this chapter we discuss different ways of adjusting HOSVD algorithms from Sections 3.3 and 3.4 to give Tucker representation of Hadamard products of tensors given in their Tucker formats. We set motivation for forming new operations with Hadamard products that we later discuss in details in Chapter 5.

4.1 Hadamard products of tensors in Tucker format

Given two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ in their Tucker formats

$$\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \cdots \times_N \mathbf{A}^{(N)}, \qquad \mathcal{Y} = \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \cdots \times_N \mathbf{B}^{(N)}, \tag{4.1}$$

with $\mathcal{F} \in \mathbb{R}^{Q_1 \times \cdots \times Q_N}$, $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times Q_n}$ and $\mathcal{G} \in \mathbb{R}^{P_1 \times \cdots \times P_N}$, $\mathbf{B}^{(n)} \in \mathbb{R}^{I_n \times P_n}$, we are interested in getting the Tucker representation of their Hadamard product $\mathcal{X} * \mathcal{Y}$.

The following lemma gives an explicit expression for that representation.

Lemma 4.1.1. *Given two tensors $\mathcal{X}, \mathcal{Y}$ as in* (4.1)*, for $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ it holds that*

$$\mathcal{Z} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \cdots \times_N \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right). \tag{4.2}$$

Proof. For simplicity we offer proof for the case when $\mathcal{X}$ and $\mathcal{Y}$ are tensors of order $N = 3$. The generalization is straightforward.

First, we will show the following equality holds,

$$(\mathcal{F} \times_n \mathbf{a}^T) \otimes (\mathcal{G} \times_n \mathbf{b}^T) = (\mathcal{F} \otimes \mathcal{G}) \times_n (\mathbf{a}^T \otimes \mathbf{b}^T),$$

for vectors $\mathbf{a} \in \mathbb{R}^{Q_n}$, $\mathbf{b} \in \mathbb{R}^{P_n}$. Without loss of generality, we assume $n = 1$. By (3.1) and Definition 3.1.3, for every k_2, k_3 we have

$$\begin{aligned}\Big[\underbrace{(\mathcal{F} \times_1 \mathbf{a}^T)}_{1\times Q_2\times Q_3} \otimes \underbrace{(\mathcal{G} \times_1 \mathbf{b}^T)}_{1\times P_2\times P_3}\Big]_{1k_2k_3} &= (\mathcal{F} \times_1 \mathbf{a}^T)_{1i_2i_3}\,(\mathcal{G} \times_1 \mathbf{b}^T)_{1j_2j_3} \\ &= f_{1i_2i_3} a_1\, g_{1j_2j_3} b_1 \\ &= (\mathcal{F} \otimes \mathcal{G})_{1k_2k_3}\,(\mathbf{a}^T \otimes \mathbf{b}^T)_1 \\ &= \Big[(\mathcal{F} \otimes \mathcal{G}) \times_1 (\mathbf{a}^T \otimes \mathbf{b}^T)\Big]_{1k_2k_3}\end{aligned}$$

Now, using (3.9) and (3.1.3.(3)), this implies

$$\begin{aligned}&\big(\mathcal{X} * \mathcal{Y}\big)_{i_1i_2i_3} = x_{i_1i_2i_3} y_{i_1i_2i_3} \\ &= \big(\mathcal{X} \times_1 \mathbf{e}_{i_1}^T \times_2 \mathbf{e}_{i_2}^T \times_3 \mathbf{e}_{i_3}^T\big)\big(\mathcal{Y} \times_1 \mathbf{e}_{i_1}^T \times_2 \mathbf{e}_{i_2}^T \times_3 \mathbf{e}_{i_3}^T\big) \\ &= \big(\mathcal{F} \times_1 \mathbf{e}_{i_1}^T\mathbf{A}^{(1)} \times_2 \mathbf{e}_{i_2}^T\mathbf{A}^{(2)} \times_3 \mathbf{e}_{i_3}^T\mathbf{A}^{(3)}\big) \otimes \big(\mathcal{G} \times_1 \mathbf{e}_{i_1}^T\mathbf{B}^{(1)} \times_2 \mathbf{e}_{i_2}^T\mathbf{B}^{(2)} \times_3 \mathbf{e}_{i_3}^T\mathbf{B}^{(3)}\big) \\ &= (\mathcal{F} \otimes \mathcal{G}) \times_1 (\mathbf{e}_{i_1}^T\mathbf{A}^{(1)} \otimes \mathbf{e}_{i_1}^T\mathbf{B}^{(1)}) \times_2 (\mathbf{e}_{i_2}^T\mathbf{A}^{(2)} \otimes \mathbf{e}_{i_2}^T\mathbf{B}^{(2)}) \times_3 (\mathbf{e}_{i_3}^T\mathbf{A}^{(3)} \otimes \mathbf{e}_{i_3}^T\mathbf{B}^{(3)}) \\ &= (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{e}_{i_1}^T(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}) \times_2 \mathbf{e}_{i_2}^T(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}) \times_3 \mathbf{e}_{i_3}^T(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}) \\ &= \Big((\mathcal{F} \otimes \mathcal{G}) \times_1 \big(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\big) \times_2 \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \times_3 \big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\Big)_{i_1i_2i_3},\end{aligned}$$

which completes the proof. □

Representation (4.2) offers Tucker representation of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ with factor matrices equal to Transpose Khatri-Rao product of factor matrices of $\mathcal{X}$ and $\mathcal{Y}$ and core tensor to Kronecker product of core tensors of $\mathcal{X}$ and $\mathcal{Y}$. This core tensor $(\mathcal{F} \otimes \mathcal{G})$ is of size $Q_1P_1 \times \cdots \times Q_NP_N$ and this can be very large, even exceed the size of the tensor itself, which makes the representation (4.2) not always useful by itself; see Example 4.1.1.

Example 4.1.1. *Let $\mathcal{X}$ and $\mathcal{Y}$ be tensors of size $100 \times 100 \times 100$ generated by evaluating functions*

$$f(x, y, z) = \frac{1}{x + y + z}, \quad g(x, y, z) = \frac{1}{\sqrt{x + y + z}}$$

on the grid $\{0.1, 0.2, \ldots, 10\}$ for x, y, z. Then $\mathcal{X}$ and $\mathcal{Y}$ both have approximate multilinear rank $(12, 12, 12)$, so they admit Tucker representation (4.1) with core tensors $\mathcal{F}$ and $\mathcal{G}$ of size $12 \times 12 \times 12$. Using representation (4.2) for $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ results in its core tensor of size $144 \times 144 \times 144$, which is not only bigger then $\mathcal{Z}$ itself, but also a lot bigger then the actual multilinear rank of $\mathcal{Z}$, which, obtained by discarding singular values smaller then 10^{-8} is $(13, 13, 13)$.

So the representation (4.2) does not give any information about the multilinear rank of the Hadamard product $\mathcal{Z}$, but it does give it a structure. Now getting the SVD of the matrix $\mathbf{Z}_{(n)}$ in Algorithms 6, 7 and 8, or the basis for its approximate range in Algorithms 9 and 10 can be done more efficiently, since it admits the structure

$$\mathbf{Z}_{(n)} = \left(\mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}\right)(\mathcal{F} \otimes \mathcal{G})_{(n)} \Big[\left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(n+1)} \odot^T \mathbf{B}^{(n+1)}\right) \otimes \left(\mathbf{A}^{(n-1)} \odot^T \mathbf{B}^{(n-1)}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \Big]^T. \quad (4.3)$$

We are going to use this structure as a starting point for developing algorithms for efficient calculation of Tucker representation of Hadamard products of tensors given in Tucker format.

There are two types of problems that we deal with. Either we want to get the Tucker representation

$$\mathcal{Z} = \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \cdots \times_N \mathbf{C}^{(N)},$$

with $\mathcal{H}$ of size equal to multilinear rank of $\mathcal{Z}$, where the equality stands up to a certain tolerance, which we call the **fixed-precision problem**, or we want to approximate $\mathcal{Z}$ with Tucker representation of lower multilinear rank

$$\mathcal{Z} \approx \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \cdots \times_N \mathbf{C}^{(N)},$$

with $\mathcal{H}$ of predefined size $R_1 \times \cdots \times R_N$, with $R_n < rank_n(\mathcal{Z})$ for one or more n, which we refer to as the **fixed-rank problem**.

In both cases we want to *recompress* Tucker representation (4.2) of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ using HOSVD algorithms; Algorithms 7 and 10 for fixed-rank problem and Algorithms 8 and 9 for fixed-precision problem.

4.2 Recompression by HOSVD

In this section we present ideas on how to modify the HOSVD algorithms to work with Hadamard products, and propose structure-exploiting operations that can improve the computations, which we thoroughly analyze in Chapter 5.

First three of the presented algorithms are based on the HOSVD algorithm from Section 3.3, while the last one is based on the HOSVD-AR algorithm from Section 3.4, and all of them can be adapted to solve both fixed-rank and fixed-precision problem; for details on usage see Appendix A.2.

4.2.1 Algorithms

HOSVD1. The straightforward approach for getting Tucker representation of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ with $\mathcal{X}$ and $\mathcal{Y}$ given in their Tucker formats (4.1) is to first create full tensors $\mathcal{X}$ and $\mathcal{Y}$ out of their Tucker representations, multiply them element-wise and then apply the HOSVD algorithm on $\mathcal{Z}$.

Since HOSVD includes calculation of left singular vectors of matricizations $\mathbf{Z}_{(n)}$, which are matrices of size $I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$, it seems reasonable to work with $I_n \times I_n$ Gramians $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$ and speed up the algorithm using an iterative method. As discussed in Section 2.3.3, when the matrix is explicitly available, as in this case, we want to use randomized algorithm (Section 2.3.2), because the matrix-vector multiplication can be done in blocks. The resulting procedure is described in Algorithm 11.

Algorithm 11 HOSVD1

1: Create $\mathcal{Z} = \text{full}(\mathcal{X}) * \text{full}(\mathcal{Y})$.
2: **for** $n = 1, \ldots, N$ **do**
3: $\quad \mathbf{C}^{(n)} \leftarrow$ left singular vectors of $\mathbf{Z}_{(n)}$ by applying randomized algorithm to $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$
4: **end for**
5: $\mathcal{H} \leftarrow \mathcal{Z} \times_1 \mathbf{C}^{(1)^T} \times_2 \cdots \times_N \mathbf{C}^{(N)^T}$
6: **return** $\mathcal{H}, \mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}$

This approach includes creating full $I_1 \times I_2 \times \cdots \times I_N$ tensor $\mathcal{Z}$ and then iteratively computing eigenvectors of $I_n \times I_n$ matrices $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$. This can be computationally challenging, so in the following we investigate ways to exploit the structure of $\mathcal{Z}$ (4.2) to speed up the algorithm.

HOSVD2. Factor matrices in the representation (4.2) of $\mathcal{Z}$ are of size $I_n \times Q_n P_n$, so if $Q_n P_n < I_n$, for every n, we can combine HOSVD algorithm with a standard recompression technique and reduce the cost. For this purpose, we first orthonormalize the columns of the factor matrices $\left(\mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}\right)$ by the truncated QR decompositions and then apply property (3.1.3.(3)):

$$\mathcal{Z} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \underbrace{\left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right)}_{\mathbf{Q}^{(1)}\mathbf{R}^{(1)}} \times_2 \cdots \times_N \underbrace{\left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right)}_{\mathbf{Q}^{(N)}\mathbf{R}^{(N)}}$$

$$= \underbrace{\left((\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{R}^{(1)} \times_2 \cdots \times_N \mathbf{R}^{(N)}\right)}_{\hat{\mathcal{H}}} \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)}.$$

The new core tensor $\hat{\mathcal{H}}$ is a tensor of size $P_1Q_1 \times \cdots \times P_NQ_N$ and it can be efficiently computed using properties of matricization of Kronecker products from Section 3.5.1 and matrix Kronecker product property (2.2.1.(5)), which we study in details in Section 5.3. Also, it has the same multilinear rank as $\mathcal{Z}$, so getting the HOSVD of $\hat{\mathcal{H}}$ gives the Tucker representation of $\mathcal{Z}$, again by using the property (3.1.3.(3)):

$$\begin{aligned}\mathcal{Z} &= \left(\mathcal{H} \times_1 \tilde{\mathbf{C}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{C}}^{(N)}\right) \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)} \\ &= \mathcal{H} \times_1 (\mathbf{Q}^{(1)}\tilde{\mathbf{C}}^{(1)}) \times_2 \cdots \times_N (\mathbf{Q}^{(N)}\tilde{\mathbf{C}}^{(N)}).\end{aligned}$$

In the case of performing the truncated HOSVD on $\hat{\mathcal{H}}$, we will end up with approximation, not equality. And again we can use the combination of HOSVD and randomized algorithm applied on $\hat{\mathbf{H}}_{(n)}\hat{\mathbf{H}}_{(n)}^T$. The resulting procedure is described in Algorithm 12.

Algorithm 12 HOSVD2

1: **for** $n = 1, \ldots, N$ **do**
2: $\quad \hat{\mathbf{C}}^{(n)} \leftarrow \mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}$.
3: $\quad$ Compute QR decomposition $\hat{\mathbf{C}}^{(n)} = \mathbf{Q}^{(n)}\mathbf{R}^{(n)}$.
4: **end for**
5: $\hat{\mathcal{H}} \leftarrow (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{R}^{(1)} \times_2 \cdots \times_N \mathbf{R}^{(N)}$
6: Apply Algorithm 6 in combination with randomized algorithm applied to $\hat{\mathbf{H}}_{(n)}\hat{\mathbf{H}}_{(n)}^T$ on line 3: $(\mathcal{H}, \tilde{\mathbf{C}}^{(1)}, \ldots, \tilde{\mathbf{C}}^{(N)}) = \text{HOSVD}(\hat{\mathcal{H}})$
7: **for** $n = 1, \ldots, N$ **do**
8: $\quad \mathbf{C}^{(n)} \leftarrow \mathbf{Q}^{(n)}\tilde{\mathbf{C}}^{(n)}$
9: **end for**
10: **return** $\mathcal{H}, \mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}$

HOSVD3. The iterative methods presented in Section 2.3 are based on fast matrix-vector multiplication, therefore another way to make use of the structure (4.2) is to create structure-exploiting matrix-vector multiplication and adjust the HOSVD algorithm to use an iterative method in combination with this new multiplication. This way we avoid creating full tensor $\mathcal{Z}$ or its factor matrices from representation (4.2). Again, we work

with $I_n \times I_n$ Gramians $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$ and now use Lanczos method (Section 2.3.1), since we do not have $\mathfrak{Z}$ explicitly available; see the discussion in Section 2.3.3. Apart from adjusting Lanczos method to use the new multiplication in order to calculate factor matrices $\mathbf{C}^{(n)}$, we can improve the calculation of the core tensor by also exploiting the structure (4.2) and using property (3.1.3.(3)):

$$\begin{aligned}\mathcal{H} &= \mathfrak{Z} \times_1 \mathbf{C}^{(1)^T} \times_2 \cdots \times_N \mathbf{C}^{(N)^T} \\ &= \left[(\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \cdots \times_N \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right) \right] \times_1 \mathbf{C}^{(1)^T} \times_2 \cdots \times_N \mathbf{C}^{(N)^T} \\ &= (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{C}^{(1)^T} \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \cdots \times_N \mathbf{C}^{(N)^T} \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right). \end{aligned} \tag{4.4}$$

This approach is presented in Algorithm 13, while the new matrix-vector calculation and the structure-exploiting calculation of core tensor are studied in Sections 5.1 and 5.3, respectively.

Algorithm 13 HOSVD3

1: **for** $n = 1, \ldots, N$ **do**
2: $\quad \mathbf{C}^{(n)} \leftarrow$ left singular vectors of $\mathbf{Z}_{(n)}$ by applying Lanczos algorithm to $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$ in combination with structure-exploiting matrix-vector multiplication
3: **end for**
4: $\mathcal{H} \leftarrow (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{C}^{(1)^T} \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \cdots \times_N \mathbf{C}^{(N)^T} \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right)$
5: **return** $\mathcal{H}, \mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}$

HOSVD4. As explained in Section 2.3.3, one of the advantages of the randomized algorithm over Lanczos is that we can introduce structure in the random matrix to fit the structure of the given matrix and speed up the matrix-vector multiplication. We will combine this with the HOSVD-AR algorithm, finding an orthonormal basis for the range of matrices $\mathbf{Z}_{(n)}$. Without loss of generalization let us assume $n = 1$. Then (3.1.3.(6)), (2.2.1.(1)) and (2.28) give

$$\begin{aligned}\mathbf{Z}_{(1)} &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right]^T \\ &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(N)^T} \odot \mathbf{B}^{(N)^T}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \right].\end{aligned}$$

Setting rank-1 vector $\mathbf{w} = \mathbf{w}_N \otimes \cdots \otimes \mathbf{w}_2$, with $\mathbf{w}_2, \ldots, \mathbf{w}_N$ randomly generated vectors of appropriate size, the multiplication $\mathbf{Z}_{(1)}\mathbf{w}$ can be done by applying property (2.2.1.(2)),

$$\mathbf{Z}_{(1)}\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(N)^T} \odot \mathbf{B}^{(N)^T}\right) \mathbf{w}_N \otimes \cdots \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \mathbf{w}_2 \right].$$

Here, we do not form vector $\mathbf{w}$ of length $I_2 I_3 \cdots I_N$, but only vectors $\mathbf{w}_n$ of lengths I_n, for $n = 2, \ldots, N$. This multiplication will be studied in details in Section 5.2.

This approach uses orthonormal basis of the range of $\mathbf{Z}_{(n)}$ to compress the size of the tensor. Then we update the core tensor $\hat{\mathcal{H}}$ as in (4.4). When dealing with fixed-rank problem, this can result in $\hat{\mathcal{H}}$ being larger than the requested rank, so we additionally perform HOSVD algorithm on it and update factor matrices accordingly. When dealing with fixed-precision problem, we can skip line 5, set $\tilde{\mathbf{C}}^{(n)}$ to be identity matrices and $\mathcal{H} = \tilde{\mathcal{H}}$. This procedure is presented in Algorithm 14.

Algorithm 14 HOSVD4

1: **for** $n = 1, \ldots, N$ **do**
2: $\quad \mathbf{Q}^{(n)} \leftarrow$ orthonormal basis for approximation of range of $\mathbf{Z}_{(n)}$
3: **end for**
4: $\tilde{\mathcal{H}} \leftarrow (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{Q}^{(1)^T} \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \cdots \times_N \mathbf{Q}^{(N)^T} \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right)$
5: Apply Algorithm 6: $(\mathcal{H}, \tilde{\mathbf{C}}^{(1)}, \ldots, \tilde{\mathbf{C}}^{(N)}) = \text{HOSVD}(\tilde{\mathcal{H}})$
6: **for** $n = 1, \ldots, N$ **do**
7: $\quad \mathbf{C}^{(n)} \leftarrow \mathbf{Q}^{(n)} \tilde{\mathbf{C}}^{(n)}$
8: **end for**
9: **return** $\mathcal{H}, \mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}$

4.2.2 Error and perturbation analysis

Let $\mathcal{Z}$ be the Hadamard product of tensors $\mathcal{X}$ and $\mathcal{Y}$ from (4.1) and let the multilinear rank of $\mathcal{Z}$ be $(R_1, \ldots, R_N)$. Furthermore, let

$$\hat{\mathcal{Z}} = \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \cdots \times_N \mathbf{C}^{(N)}$$

be output of the algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4 as the solution of the fixed-precision problem with given $\varepsilon > 0$.

Error analysis

HOSVD1. In Algorithm 11, we compare $\hat{\mathcal{Z}}$ with $\mathcal{Z}$ calculated on line 1. Applying property (3.1.3.(3)),

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F = \|\mathcal{Z} - \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \cdots \times_N \mathbf{C}^{(N)}\|_F$$

$$= \|\mathcal{Z} - \mathcal{Z} \times_1 \mathbf{C}^{(1)}\mathbf{C}^{(1)^T} \times_2 \cdots \times_N \mathbf{C}^{(N)}\mathbf{C}^{(N)^T}\|_F.$$

Setting $\mathbf{P}_n = \mathbf{C}^{(n)}\mathbf{C}^{(n)^T}$, the same way as in the proof of Corollary 3.4.1, gives

$$\|\mathcal{Z} - \mathcal{Z} \times_1 \mathbf{P}_1 \times_2 \cdots \times_N \mathbf{P}_N\|_F \leq \sum_{n=1}^{N} \|\mathcal{Z} \times_n (\mathbf{I} - \mathbf{P}_n)\|_F.$$

Now, since we have obtained matrices $\mathbf{C}^{(n)}$ from the randomized algorithm applied on $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$, assumption

$$\|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T\|_2 \leq \varepsilon,$$

can be achieved using $\epsilon = \varepsilon/6$; see (2.51). This, together with (2.26) and (2.19) gives

$$\begin{aligned}\|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\|_2^2 &= \|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T(\mathbf{I} - \mathbf{P}_n)\|_2 \\ &\leq \|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T\|_2 \|\mathbf{I} - \mathbf{P}_n\|_2 \leq \varepsilon.\end{aligned}$$

From (2.11) we have

$$\|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\|_F \leq \sqrt{R_n}\|(\mathbf{I} - \mathbf{P}_n)\mathbf{Z}_{(n)}\|_2 \leq \sqrt{R_n\varepsilon}, \tag{4.5}$$

which, by (3.1.1.(1)), gives the bound

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq \sqrt{\varepsilon}\sum_{n=1}^{N}\sqrt{R_n}.$$

HOSVD2. In Algorithm 12 we apply HOSVD algorithm on tensor $\hat{\mathcal{H}}$, using randomized algorithm on $\hat{\mathbf{H}}_{(n)}\hat{\mathbf{H}}_{(n)}^T$ to get factor matrices. Since $\hat{\mathcal{H}}$ inherits the multilinear rank from $\mathcal{Z}$, then the same way as in HOSVD1, running randomized algorithm with $\epsilon = \varepsilon/6$ gives

$$\|\hat{\mathcal{H}} - \mathcal{H} \times_1 \tilde{\mathbf{C}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{C}}^{(N)}\|_F \leq \sqrt{\varepsilon}\sum_{n=1}^{N}\sqrt{R_n}.$$

Comparing $\hat{\mathcal{Z}}$ to $\mathcal{Z}$ from (4.2), from orthonormality of matrices $\mathbf{Q}^{(n)}$ and (3.1.3.(5-ii)) follows

$$\begin{aligned}\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F &= \|\hat{\mathcal{H}} \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)} - \\ &\qquad \left(\mathcal{H} \times_1 \tilde{\mathbf{C}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{C}}^{(N)}\right) \times_1 \mathbf{Q}^{(1)} \times_2 \cdots \times_N \mathbf{Q}^{(N)}\|_F \\ &= \|\hat{\mathcal{H}} - \mathcal{H} \times_1 \tilde{\mathbf{C}}^{(1)} \times_2 \cdots \times_N \tilde{\mathbf{C}}^{(N)}\|_F \leq \sqrt{\varepsilon}\sum_{n=1}^{N}\sqrt{R_n}.\end{aligned}$$

HOSVD3. Algorithm 13 gets factor matrices by applying Lanczos algorithm with structure exploiting matrix-vector multiplication on matrices $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$, where $\mathcal{Z}$ is given by (4.2). Again, following the procedure for HOSVD1 - setting $\mathbf{P}_n = \mathbf{C}^{(n)}\mathbf{C}^{(n)^T}$ and assuming

$$\| \left(\mathbf{I} - \mathbf{P}_n\right) \mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T\|_F \leq \varepsilon,$$

which can be achieved using $\epsilon = \varepsilon/4$ as tolerance for Lanczos algorithm (see (2.45)) and using 2-norm and Frobenius norm equivalence (2.11), we obtain the bound

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq \sqrt{\varepsilon} \sum_{n=1}^{N} \sqrt{R_n}.$$

HOSVD4. Algorithm 14 describes a variant of the HOSVD-AR algorithm, which uses Stage A of the randomized algorithm combined with rank-1 matrix-vector multiplication applied on matrix $\mathbf{Z}_{(n)}$, where $\mathcal{Z}$ is given by (4.2). Now, since with fixed precision problem matrix $\mathbf{C}^{(n)}$ is exactly matrix $\mathbf{Q}^{(n)}$, the output of Algorithm 3, with $\mathbf{P}_n = \mathbf{C}^{(n)}\mathbf{C}^{(n)^T}$,

$$\| \left(\mathbf{I} - \mathbf{P}_n\right) \mathbf{Z}_{(n)}\|_F \leq \varepsilon$$

can be achieved using $\epsilon = \varepsilon/\sqrt{R_n}$; see (2.11). From Corollary 3.4.1 we have

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq N\varepsilon.$$

With the above analysis we have proved the following theorem.

Theorem 4.2.1. *Let $\mathcal{Z}$ be the Hadamard product of tensors $\mathcal{X}$ and $\mathcal{Y}$ from (4.1) and let the multilinear rank of $\mathcal{Z}$ be $(R_1, \ldots, R_N)$. Denoting $\hat{\mathcal{Z}}$ to be output of the algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4 as the solution of the fixed-precision problem with given $\varepsilon > 0$, algorithms HOSVD1, HOSVD2 and HOSVD3 admit the bound*

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq \sqrt{\varepsilon} \sum_{n=1}^{N} \sqrt{R_n}, \tag{4.6}$$

while for the HOSVD4 algorithm the following bound holds,

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq N\varepsilon. \tag{4.7}$$

The fact that HOSVD4 algorithm gives best results in terms of accuracy is expected, since it is the only of the four algorithms not working with Gramians $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$, but directly with matrices $\mathbf{Z}_{(n)}$.

Perturbation analysis

Let tensors $\mathcal{X}$ and $\mathcal{Y}$ admit perturbations

$$\tilde{\mathcal{X}} = \mathcal{X} + \delta\mathcal{X}, \quad \tilde{\mathcal{Y}} = \mathcal{Y} + \delta\mathcal{Y},$$

where $\|\delta\mathcal{X}\|_F \leq \varepsilon_P$ and $\|\delta\mathcal{Y}\|_F \leq \varepsilon_P$, for some small positive ε_P. Denoting $\tilde{\mathcal{Z}} = \tilde{\mathcal{X}} * \tilde{\mathcal{Y}}$, we are interested in bounding $\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F$. Rewriting $\tilde{\mathcal{Z}}$ as $\tilde{\mathcal{Z}} = \mathcal{Z} + \delta\mathcal{Z}$ gives

$$\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F = \|\mathcal{Z} + \delta\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq \|\mathcal{Z} - \hat{\mathcal{Z}}\|_F + \|\delta\mathcal{Z}\|_F,$$

so, depending which algorithm we are using, we can use the results from Theorem 4.2.1 to bound the first term, while $\delta\mathcal{Z}$ has to be determined. This can be done easily by applying the linearity of Hadamard products,

$$\tilde{\mathcal{Z}} = \tilde{\mathcal{X}} * \tilde{\mathcal{Y}} = (\mathcal{X} + \delta\mathcal{X}) * (\mathcal{Y} + \delta\mathcal{Y}) = \underbrace{\mathcal{X} * \mathcal{Y}}_{\mathcal{Z}} + \underbrace{\mathcal{X} * \delta\mathcal{Y} + \delta\mathcal{X} * \mathcal{Y} + \delta\mathcal{X} * \delta\mathcal{Y}}_{\delta\mathcal{Z}}.$$

Now, by (3.2) and Corollary 3.3.3 and by ignoring the terms of order $O(\varepsilon_P^2)$, we get

$$\begin{aligned}
\|\delta\mathcal{Z}\|_F &= \|\mathcal{X} * \delta\mathcal{Y} + \delta\mathcal{X} * \mathcal{Y} + \delta\mathcal{X} * \delta\mathcal{Y}\|_F \\
&\leq \|\mathcal{X}\|_F\|\delta\mathcal{Y}\|_F + \|\delta\mathcal{X}\|_F\|\mathcal{Y}\|_F + \|\delta\mathcal{X}\|_F\|\delta\mathcal{Y}\|_F \\
&\leq \varepsilon_P\|\mathcal{X}\|_F + \varepsilon_P\|\mathcal{Y}\|_F + \varepsilon_P^2 \approx \varepsilon_P\left(\|\mathcal{F}\|_F + \|\mathcal{G}\|_F\right).
\end{aligned}$$

Theorem 4.2.2. *Let $\mathcal{Z}$ be the Hadamard product of tensors $\mathcal{X}$ and $\mathcal{Y}$ from* (4.1)*, and let $\hat{\mathcal{Z}}$ and ε be given as in Theorem 4.2.1. If the perturbations of $\mathcal{X}$ and $\mathcal{Y}$ are given as $\tilde{\mathcal{X}} = \mathcal{X} + \varepsilon_P\mathcal{X}^P$ and $\tilde{\mathcal{Y}} = \mathcal{Y} + \varepsilon_P\mathcal{Y}^P$, with ε_P small positive number and $\|\mathcal{X}^P\|_F \leq 1$ and $\|\mathcal{Y}^P\|_F \leq 1$, and we denote $\tilde{\mathcal{Z}} = \tilde{\mathcal{X}} * \tilde{\mathcal{Y}}$, then $\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F$ admits the following bounds. For the HOSVD1, HOSVD2 and HOSVD3 algorithms,*

$$\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F \leq \sqrt{\varepsilon}\sum_{n=1}^{N}\sqrt{R_n} + \varepsilon_P\left(\|\mathcal{F}\|_F + \|\mathcal{G}\|_F\right), \tag{4.8}$$

while for the HOSVD4 algorithm,

$$\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F \leq N\varepsilon + \varepsilon_P\left(\|\mathcal{F}\|_F + \|\mathcal{G}\|_F\right). \tag{4.9}$$

Remark 4.2.3. *If matricizations $\mathbf{Z}_{(n)}$ have quickly decaying singular values, then the difference between the 2-norm and the Frobenius norm will not be large, and instead of*

the term $\sqrt{R_n}$ *in* (4.5) *we will have some constant* $C_n > 1$*, and in that case* (4.6) *can be replaced with*

$$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F \leq \sqrt{\varepsilon} \sum_{n=1}^{N} C_n = C\sqrt{\varepsilon}, \tag{4.10}$$

and (4.8) *can be replaced by*

$$\|\tilde{\mathcal{Z}} - \hat{\mathcal{Z}}\|_F \leq C\sqrt{\varepsilon} + \varepsilon_P \left(\|\mathcal{F}\|_F + \|\mathcal{G}\|_F\right), \tag{4.11}$$

with $C > N$ *depending on the singular values of matrices* $\mathbf{Z}_{(n)}$*.*

For numerical results on accuracy and perturbation bounds see Section 6.2.2.

Chapter 5

Exploiting structure in the recompression of Hadamard products

In Chapter 4 we have set ideas on how to use the known structure (4.2) of Hadamard products of tensors in Tucker format to efficiently compute the Tucker representation of the products. Here, we create and analyze the new structure-exploiting operations needed for the algorithms proposed in Section 4.2.

Throughout this chapter, to simplify the expressions, we will restrict the discussion to tensors of order $N = 3$ with same size in each mode and simplified Tucker representations.

Let $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I \times I \times I}$ be tensors given in their Tucker formats

$$\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)}, \qquad \mathcal{Y} = \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \times_3 \mathbf{B}^{(3)}, \tag{5.1}$$

with $\mathcal{F}, \mathcal{G} \in \mathbb{R}^{R \times R \times R}$ and $\mathbf{A}^{(n)}, \mathbf{B}^{(n)} \in \mathbb{R}^{I \times R}$, for $n = 1, 2, 3$ and $R < I$. Now, Hadamard product $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ is an $I \times I \times I$ tensor and admits Tucker representation

$$\mathcal{Z} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right). \tag{5.2}$$

5.1 Fast matrix-vector multiplication with $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$

Given tensor $\mathcal{Z}$ in format (5.2), we develop a fast algorithm for multiplying $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T \in \mathbb{R}^{I \times I}$, where $\mathbf{Z}_{(n)}$ is the mode-n matricization of tensor $\mathcal{Z}$ (see Definition 3.1.1), with a vector $\mathbf{v} \in \mathbb{R}^I$.

Without loss of generality, we will assume $n = 1$. From (3.1.3.(6)), we have

$$\begin{aligned}\mathbf{Z}_{(1)} &= \left[(\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \right]_{(1)} \\ &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right]^T.\end{aligned}$$

Now, using Kronecker product property (2.2.1.(1)) and definition of Transpose Khatri-Rao product (2.28), it follows

$$\begin{aligned}\mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} &\left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right]^T \\ &\left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right] (\mathcal{F} \otimes \mathcal{G})_{(1)}^T \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right)^T \\ = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} &\left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \right] \\ &\left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right] (\mathcal{F} \otimes \mathcal{G})_{(1)}^T \left(\mathbf{A}^{(1)^T} \odot \mathbf{B}^{(1)^T}\right).\end{aligned}$$

We perform the matrix-vector multiplication $\mathbf{w} = \mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T\mathbf{v}$ in five steps.

Step 1. $\mathbf{w}_1 = \left(\mathbf{A}^{(1)^T} \odot \mathbf{B}^{(1)^T}\right) \mathbf{v}$

Step 2. $\mathbf{w}_2 = (\mathcal{F} \otimes \mathcal{G})_{(1)}^T \mathbf{w}_1$

Step 3. $\mathbf{w}_3 = \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right] \mathbf{w}_2,$
$\mathbf{w}_4 = \left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \right] \mathbf{w}_3$

Step 4. $\mathbf{w}_5 = (\mathcal{F} \otimes \mathcal{G})_{(1)} \mathbf{w}_4$

Step 5. $\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \mathbf{w}_5$

In the following, we discuss an efficient implementation for each of these steps.

Step 1. To get $\mathbf{w}_1$ we apply property (2.2.1.(6))

$$\mathbf{w}_1 = \left(\mathbf{A}^{(1)^T} \odot \mathbf{B}^{(1)^T}\right)\mathbf{v} = vec\left(\mathbf{B}^{(1)^T} diag(\mathbf{v}) \mathbf{A}^{(1)}\right).$$

To calculate this we have to form $R \times R$ matrix

$$\mathbf{W}_1 = \underbrace{\mathbf{B}^{(1)^T}}_{R \times I} diag(\mathbf{v}) \underbrace{\mathbf{A}^{(1)}}_{I \times R}.$$

Performing this matrix multiplication as explained in Section 2.2.1 requires $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory.

Step 2. For $\mathbf{w}_2$ we first apply property (3.19) and then (2.2.1.(5))

$$\mathbf{w}_2 = (\mathcal{F} \otimes \mathcal{G})^T_{(1)} \mathbf{w}_1 = \Big[\big(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\big) \mathbf{P}_1 \Big]^T = \mathbf{P}_1^T \big(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\big) \mathbf{w}_1$$
$$= \mathbf{P}_1^T \, vec\Big(\mathbf{G}_{(1)}^T \mathbf{W}_1 \mathbf{F}_{(1)}\Big).$$

Here we have to form $R^2 \times R^2$ matrix $\widetilde{\mathbf{W}}_2$

$$\widetilde{\mathbf{W}_2} = \underbrace{\mathbf{G}_{(1)}^T}_{R^2 \times R} \underbrace{\mathbf{W}_1}_{R \times R} \underbrace{\mathbf{F}_{(1)}}_{R \times R^2},$$

which requires $\mathcal{O}(R^5)$ operations and $\mathcal{O}(R^4)$ memory, while the cost for applying $\mathbf{P}_1^T$ on $vec(\widetilde{\mathbf{W}}_2)$ – using the techniques described in Section 3.5.1, is negligible.

Step 3. We present two variants for performing Step 3 and, as will be explained below, the choice of the variant depends on the relation between I and R.

Variant A. First, we create vector $\mathbf{w}_3$ by reshaping vector $\mathbf{w}_2$ into $R^2 \times R^2$ matrix $\mathbf{W}_2$ and applying property (2.2.1.(5))

$$\mathbf{w}_3 = \Big[\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \otimes \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \Big] \mathbf{w}_2 = vec\Big[\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \mathbf{W}_2 \big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \Big].$$

Now we perform the first multiplication and create matrix which we denote as $\widetilde{\mathbf{W}}_3$ and use $\mathbf{c}_j$ to denote jth column of $\mathbf{W}_2$, for $j = 1, \ldots, R^2$,

$$\widetilde{\mathbf{W}}_3 = \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \mathbf{W}_2 = \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \, [\, \mathbf{c}_1 \, \cdots \, \mathbf{c}_{R^2} \,].$$

Each column of the resulting matrix is calculated by reshaping vector $\mathbf{c}_j$, which is of length R^2, into $R \times R$ matrix $\mathbf{C}_j$ and applying property (2.2.1.(7)),

$$\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \mathbf{c}_j = diag\Big(\underbrace{\mathbf{B}^{(2)}}_{I \times R} \mathbf{C}_j \underbrace{\mathbf{A}^{(2)^T}}_{R \times I} \Big), \quad j = 1, \ldots, R^2,$$

which is then calculated as explained in Section 2.2.1 in $\mathcal{O}(IR^2)$ operations for each $\mathbf{c}_j$. Hence, $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^2)$ memory in total for computing $\widetilde{\mathbf{W}}_3 \in \mathbb{R}^{I \times R^2}$. The matrix

$$\mathbf{W}_3 = \widetilde{\mathbf{W}}_3 \big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) = \Big[\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \, \widetilde{\mathbf{W}}_3^T \Big]^T$$

is computed analogously - letting $\tilde{\mathbf{c}}_j \in \mathbb{R}^{R^2}$ denote the jth column of $\widetilde{\mathbf{W}}_3^T$ for $j = 1, \ldots, I$, and reshaping it into $R \times R$ matrix $\widetilde{\mathbf{C}}_j$, the jth column of $\mathbf{W}_3^T$ is given by

$$\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \tilde{\mathbf{c}}_j = diag\Big(\underbrace{\mathbf{B}^{(3)}}_{I \times R} \widetilde{\mathbf{C}}_j \underbrace{\mathbf{A}^{(3)^T}}_{R \times I} \Big), \quad j = 1, \ldots, I. \tag{5.3}$$

Therefore, the computation of $\mathbf{W}_3 \in \mathbb{R}^{I\times I}$ from $\widetilde{\mathbf{W}}_3$ requires $\mathcal{O}(I^2R^2)$ operations and $\mathcal{O}(I^2)$ memory.

We compute $\mathbf{w}_4$ by first using property (2.2.1.(5)) as

$$\mathbf{w}_4 = \Big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\Big]\mathbf{w}_3 = vec\,\Big[\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{W}_3\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\Big].$$

Letting $\mathbf{d}_j$ denote the jth column of $\mathbf{W}_3$, the jth column of

$$\widetilde{\mathbf{W}}_4 = \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{W}_3 = \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\,[\,\mathbf{d}_1 \cdots \mathbf{d}_I\,]$$

is given by

$$\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{d}_j = vec\,\Big(\underbrace{\mathbf{B}^{(2)^T}}_{R\times I} diag(\mathbf{d}_j)\underbrace{\mathbf{A}^{(2)}}_{I\times R}\Big), \tag{5.4}$$

which is computed as in Section 2.2.1 in $\mathcal{O}(IR^2)$ operations, for $j = 1,\ldots,I$. Therefore, the computation of $\widetilde{\mathbf{W}}_4 \in \mathbb{R}^{R^2\times I}$ requires $\mathcal{O}(I^2R^2)$ operations and $\mathcal{O}(IR)$ memory. Analogously, the jth column of $\mathbf{W}_4^T = \Big[\widetilde{\mathbf{W}}_4\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\Big]^T$ is computed from the jth column $\tilde{\mathbf{d}}_j$ of $\widetilde{\mathbf{W}_4}^T$ via

$$\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\tilde{\mathbf{d}}_j = vec\,\big(\mathbf{B}^{(3)^T} diag(\tilde{\mathbf{d}}_j)\mathbf{A}^{(3)}\big), \quad j = 1,\ldots,R^2.$$

This requires $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^2)$ memory for all columns of $\mathbf{W}_4$.

Overall, Variant A of Step 3 has the complexity of $\mathcal{O}(I^2R^2 + IR^4)$ operations and $\mathcal{O}(I^2 + IR^2)$ memory.

Variant B. For tensors of large mode sizes the quadratic growth with respect to I in the complexity of Variant A is undesirable. We can avoid this by utilizing (2.2.1.(2)) first:

$$\begin{aligned}
\mathbf{w}_4 &= \Big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\Big]\mathbf{w}_3 \\
&= \Big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\Big]\Big[\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \otimes \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\Big]\mathbf{w}_2 \\
&= \Big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\Big]\mathbf{w}_2 \\
&= vec\,\Big[\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\mathbf{W}_2\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\Big].
\end{aligned}$$

As in Variant A, we first perform

$$\widetilde{\mathbf{W}}_3 = \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\mathbf{W}_2,$$

which requires $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^2)$ memory. Then, analogous to (5.4), we compute the $R^2 \times R^2$ matrix

$$\mathbf{W}_3 = \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right)\widetilde{\mathbf{W}}_3$$

within $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^3)$ memory. Now, analogous to (5.3), the computation of the $R^2 \times I$ matrix

$$\widetilde{\mathbf{W}}_4 = \mathbf{W}_3\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right)$$

requires again $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(R^4)$ memory. Finally, as in Variant A, we compute

$$\mathbf{W}_4 = \widetilde{\mathbf{W}}_4\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right)$$

within $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^2)$ memory.

In total, Variant B has complexity of $\mathcal{O}(IR^4)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory.

Comparison of variants. Comparing the computational complexities of Variant A and Variant B, we see that Variant B avoids the term I^2R^2 in the complexity of Variant A. However, this does not mean that Variant B is always the better choice. Let us compare the sizes of the intermediate matrices created by the two variants:

	$\widetilde{\mathbf{W}}_3$	$\mathbf{W}_3$	$\widetilde{\mathbf{W}}_4$	$\mathbf{W}_4$
Variant A	$I \times R^2$	$I \times I$	$R^2 \times I$	$R^2 \times R^2$
Variant B	$I \times R^2$	$R^2 \times R^2$	$R^2 \times I$	$R^2 \times R^2$

We see a difference in matrix $\mathbf{W}_3$, which is in both variants calculated column by column and in the cases when $I < R^2$ Variant B will take more time to calculate this matrix. So, in terms of storage requirements it could be preferable to use Variant A when $I < R^2$. In this situation, term IR^4 dominates computational complexity of Variant A, so Variant A turns out to be faster than Variant B. This can be clearly seen in Figure 5.1, which shows that a matrix-vector multiplication based on Variant A becomes faster for $R \geq 35 \approx \sqrt{1000} = \sqrt{I}$; see Section 1.4 for a description of the computational environment. To conclude, we will use Variant A when $I < R^2$ and Variant B otherwise.

Step 4. In the fourth step, similar to the second step, we use properties (3.19) and (2.2.1.(5))

$$\mathbf{w}_5 = (\mathcal{F} \otimes \mathcal{G})_{(1)}\,\mathbf{w}_4 = \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right)\underbrace{\mathbf{P}_1\mathbf{w}_4}_{\hat{\mathbf{w}}_4}.$$

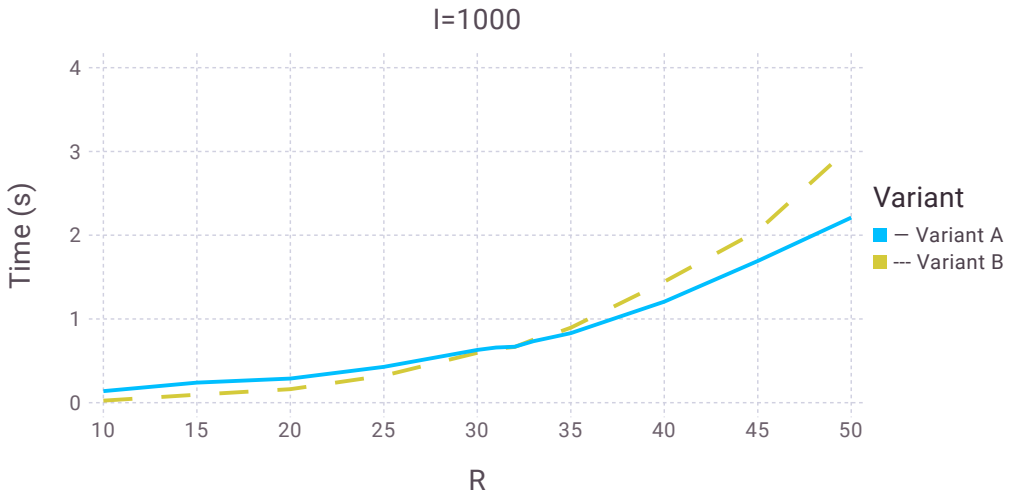


Figure 5.1: Execution time in seconds for matrix-vector multiplication with $\mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T$ for random tensors $\mathcal{X}$ and $\mathcal{Y}$ with $I = 1000$ and ranks $R = 10, 15, 20, 25, 30, 31, 32, 33, 34, 35, 40, 45, 50$ using either Variant A or Variant B in Step 3.

The permutation matrix $\mathbf{P}_1$ is costlessly applied as explained in Section 3.5.1. Then, reshaping $\hat{\mathbf{w}}_4$ into $R^2 \times R^2$ matrix $\hat{\mathbf{W}}_4$, we get

$$\mathbf{w}_5 = vec\left(\underbrace{\mathbf{G}_{(1)}}_{R\times R^2}\hat{\mathbf{W}}_4\underbrace{\mathbf{F}_{(1)}^T}_{R^2\times R}\right),$$

by forming $R \times R$ matrix $\mathbf{W}_5 = \mathbf{G}_{(1)}\hat{\mathbf{W}}_4\mathbf{F}_{(1)}^T$, in $\mathcal{O}(R^5)$ operations and $\mathcal{O}(R^4)$ memory.

Step 5. In the final step we calculate

$$\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right)\mathbf{w}_5 = diag\left(\underbrace{\mathbf{B}^{(1)}}_{I\times R}\mathbf{W}_5\underbrace{\mathbf{A}^{(1)^T}}_{R\times I}\right).$$

Again, making use of the fact that we only need diagonal elements, we get vector $\mathbf{w}$ in $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory.

Summary. All together, performing matrix-vector multiplication $\mathbf{w} = \mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T\mathbf{v}$ by exploiting the structure can be done in $\mathcal{O}(I^2R^2+IR^4+R^5)$ operations and $\mathcal{O}(I^2+IR^2+R^4)$ memory when using Variant A in Step 3 (for $I < R^2$), and $\mathcal{O}(IR^4 + R^5)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory, when using Variant B (for $I \geq R^2$).

5.2 Fast multiplication of $\mathbf{Z}_{(n)}$ with a rank-1 vector

Another multiplication we are interested in is the multiplication of $\mathbf{Z}_{(n)}$ with a rank-1 vector, where $\mathbf{Z}_{(n)} \in \mathbb{R}^{I \times I^2}$ is the mode-n matricization of tensor $\mathfrak{Z}$ with structure (5.2) and rank-1 vector is a vector of the form $\mathbf{w} = \mathbf{x} \otimes \mathbf{y}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^I$.

Again, without loss of generality we assume $n = 1$. Then

$$\begin{aligned}\mathbf{Z}_{(1)}\mathbf{w} &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\right]^T \mathbf{w} \\ &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right)\right] (\mathbf{x} \otimes \mathbf{y}).\end{aligned}$$

Applying property (2.2.1.(2)) gives

$$\mathbf{Z}_{(1)}\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \mathbf{x} \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \mathbf{y}\right].$$

Now we compute vectors $\tilde{\mathbf{x}} = \left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \mathbf{x}$, and $\tilde{\mathbf{y}} = \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \mathbf{y}$, both of length R^2, within $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory using procedure from Section 2.2.1, and then form vector of length R^4

$$\mathbf{z}_1 = \tilde{\mathbf{x}} \otimes \tilde{\mathbf{y}} \tag{5.5}$$

in $\mathcal{O}(R^4)$ operations. It follows

$$\mathbf{Z}_{(1)}\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \mathbf{z}_1.$$

To calculate the product

$$(\mathcal{F} \otimes \mathcal{G})_{(1)} \underbrace{\mathbf{z}_1}_{R^4 \times 1} = \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right) \underbrace{\mathbf{P}_1 \mathbf{z}_1}_{R^4 \times 1},$$

we first perform the permutation $\mathbf{z}_2 = \mathbf{P}_1 \mathbf{z}_1$ as discussed in Section 3.5.1. Reshaping vector $\mathbf{z}_2$ into $R^2 \times R^2$ matrix $\mathbf{Z}_2$ and using (2.2.1.(5)),

$$\mathbf{z}_3 = \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right) \mathbf{z}_2 = vec\left(\underbrace{\mathbf{G}_{(1)}}_{R \times R^2} \mathbf{Z}_2 \underbrace{\mathbf{F}_{(1)}^T}_{R^2 \times R}\right),$$

where forming $R \times R$ matrix $\mathbf{Z}_3 = \mathbf{G}_{(1)} \mathbf{Z}_2 \mathbf{F}_{(1)}^T$ requires $\mathcal{O}(R^5)$ operations and $\mathcal{O}(R^3)$ memory. Applying (2.2.1.(7)), last step

$$\mathbf{z} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \mathbf{z}_3 = diag\left(\underbrace{\mathbf{B}^{(1)}}_{I \times R} \mathbf{Z}_3 \underbrace{\mathbf{A}^{(1)^T}}_{R \times I}\right),$$

as discussed in Section 2.2.1, requires $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory.

Summary. The matrix-vector product $\mathbf{Z}_{(n)}\mathbf{w}$, where $\mathbf{w}$ is a rank-1 vector and $\mathbf{Z}_{(n)}$ the mode-n matricization of tensor $\mathcal{Z}$ with structure (5.2), can be performed in $\mathcal{O}(IR^2 + R^5)$ operations and $\mathcal{O}(IR + R^4)$ memory.

5.3 Calculating core tensor of Hadamard product

Assuming that we have calculated orthonormal factor matrices $\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \mathbf{C}^{(3)}$ of the Tucker representation of tensor $\mathcal{Z}$,

$$\mathcal{Z} \approx \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \mathbf{C}^{(3)} \times_3 \mathbf{C}^{(3)},$$

where $\mathcal{Z}$ has structure (5.2), we now discuss the efficient computation of the core tensor $\mathcal{H}$.

From Section 3.3 we know that we can get $\mathcal{H}$ as

$$\mathcal{H} = \mathcal{Z} \times_1 \mathbf{C}^{(1)^T} \times_2 \mathbf{C}^{(3)^T} \times_3 \mathbf{C}^{(3)^T}.$$

Exploiting the structure of $\mathcal{Z}$, as in the N-dimensional case (4.4), we get

$$\mathcal{H} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{C}^{(1)^T} \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \mathbf{C}^{(2)^T} \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \mathbf{C}^{(3)^T} \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right).$$

To simplify the complexity considerations, we assume that the size of each matrix $\mathbf{C}^{(n)}$ is $I \times R$, matching the sizes of $\mathbf{A}^{(n)}$ and $\mathbf{B}^{(n)}$. First, for $n = 1, 2, 3$, we calculate $R^2 \times R$ matrices

$$\mathbf{D}^{(n)} = \left(\mathbf{A}^{(n)^T} \odot \mathbf{B}^{(n)^T}\right)\mathbf{C}^{(n)},$$

by first forming the Khatri-Rao product in $\mathcal{O}(IR^3)$ operations and $\mathcal{O}(IR^2)$ memory. We rewrite

$$\mathcal{H} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{D}^{(1)^T} \times_2 \mathbf{D}^{(2)^T} \times_3 \mathbf{D}^{(3)^T} \tag{5.6}$$

in terms of the mode-1 matricization

$$\mathbf{H}_{(1)} = \mathbf{D}^{(1)^T} (\mathcal{F} \otimes \mathcal{G})_{(1)} \left(\mathbf{D}^{(3)} \otimes \mathbf{D}^{(2)}\right) = \left[\left(\underbrace{\mathbf{D}^{(3)^T} \otimes \mathbf{D}^{(2)^T}}_{R^2 \times R^4} \right) \underbrace{(\mathcal{F} \otimes \mathcal{G})_{(1)}^T}_{R^4 \times R^2} \underbrace{\mathbf{D}^{(1)}}_{R^2 \times R} \right]^T.$$

We start multiplying from the right and again use properties (3.19) and (2.2.1.(5)) column by column

$$\mathbf{M} = (\mathcal{F} \otimes \mathcal{G})_{(1)}^T \mathbf{D}^{(1)} = \mathbf{P}_1^T \left(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\right) \mathbf{D}^{(1)},$$

and we do not store the entire matrix $\mathbf{M}$, since its size is $R^4 \times R$, and we have to multiply $\left(\mathbf{D}^{(3)^T} \otimes \mathbf{D}^{(2)^T}\right)$ by $\mathbf{M}$ column by column again, anyway.

Denoting the jth column of $\mathbf{D}^{(1)}$ as $\mathbf{d}_j \in \mathbb{R}^{R^2}$ and reshaping it into $R \times R$ matrix $\mathbf{D}_j$, we calculate the jth column of $\mathbf{M}$,

$$\mathbf{m}_j = \mathbf{P}_1 \left(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\right) \mathbf{d}_j = \mathbf{P}_1 \, vec \left(\underbrace{\mathbf{G}_{(1)}^T}_{R^2 \times R} \mathbf{D}_j \underbrace{\mathbf{F}_{(1)}}_{R \times R^2} \right) \tag{5.7}$$

in $\mathcal{O}(R^5)$ operations, and perform another multiplication using that column right away by reshaping $\mathbf{m}_j$ into $R^2 \times R^2$ matrix $\mathbf{M}_j$

$$\left(\mathbf{D}^{(3)^T} \otimes \mathbf{D}^{(2)^T}\right) \mathbf{m}_j = vec \left(\underbrace{\mathbf{D}^{(2)^T}}_{R \times R^2} \mathbf{M}_j \underbrace{\mathbf{D}^{(3)}}_{R^2 \times R} \right), \tag{5.8}$$

which has complexity $\mathcal{O}(R^5)$ and gives the jth row of $\mathbf{H}_{(1)}$, for $j = 1, \ldots, R$. After we calculate all columns, we simply reshape the matrix into a tensor.

Summary. All together, the computation of core tensor $\mathcal{H}$ requires $\mathcal{O}(IR^3 + R^6)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory.

Remark 5.3.1. *When dealing with general tensors $\mathcal{X}$ and $\mathcal{Y}$ of order N as in* (4.1)*, after we calculate factor matrices $\mathbf{C}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$, we want to perform the multiplication* (5.6) *by choosing the mode-n matricization that will give biggest memory reduction. We choose mode n such that $P_n Q_n - R_n$ is maximal, since then*

$$\mathbf{M} = \mathbf{P}_n^T \underbrace{\left(\mathbf{F}_{(n)}^T \otimes \mathbf{G}_{(n)}^T\right)}_{\prod_{\substack{i=1 \\ i \neq n}}^{N} P_i Q_i \times P_n Q_n} \underbrace{\mathbf{D}^{(n)}}_{P_n Q_n \times R_n}$$

is of size $\prod_{\substack{i=1 \\ i \neq n}}^{N} P_i Q_i \times R_n$.

Chapter 6

Algorithmic complexities and numerical experiments

This chapter includes detailed analysis of the combination of algorithms presented in Section 4.2 and operations from Chapter 5. In Section 6.1 we provide complexity analysis and Section 6.2 contains numerical experiments which prove our theoretical conclusions.

6.1 Algorithmic complexities

6.1.1 Complexities for $N = 3$

To calculate the complexities of the algorithms from Section 4.2 we again assume we are dealing with tensors $\mathcal{X}$ and $\mathcal{Y}$ or order $N = 3$ as in (5.1), and that we want to get the Tucker representation of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$,

$$\mathcal{Z} \approx \mathcal{H} \times_1 \mathbf{C}^{(1)} \times_2 \mathbf{C}^{(2)} \times_3 \mathbf{C}^{(3)},$$

with $\mathcal{H}$ of size $R \times R \times R$. In other words, assume we are solving the fixed-rank problem.

HOSVD1 algorithm

The straightforward approach is presented in Algorithm 11 and we refer to it as the HOSVD1. It forms full tensors $\mathcal{X}$ and $\mathcal{Y}$ out of their Tucker representations, which requires $\mathcal{O}(I^3R)$ operations and $\mathcal{O}(I^3)$ memory (see Section 3.5.2), after which we need another $\mathcal{O}(I^3)$ operations for creating their Hadamard product $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$.

Next step is to use randomized algorithm to get R left singular vectors of matricizations $\mathbf{Z}_{(n)}$ by getting spectral decomposition of $I \times I$ matrices $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$, for $n = 1, 2, 3$, which requires $\mathcal{O}(I^3R)$ operations and $\mathcal{O}(I^2R)$ memory (see Section 2.3.2). Additionally, $\mathcal{O}(I^3R)$ operations and $\mathcal{O}(I^3)$ memory is needed to form the core tensor (see Section 3.5.2).

All together, the HOSVD1 algorithm requires $\mathcal{O}(I^3R)$ operations and $\mathcal{O}(I^3)$ memory.

HOSVD2 algorithm

The approach which exploits structure (5.2) by performing the QR decomposition of each factor matrix from (5.2), was presented in Algorithm 12 and we call it the HOSVD2.

First, forming the $I \times R^2$ factor matrices $\hat{\mathbf{C}}^{(n)} = \mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}$, for $n = 1, 2, 3$, requires $\mathcal{O}(IR^2)$ operations (see Section 2.2.1), and creating their QR decompositions $\hat{\mathbf{C}}^{(n)} = \mathbf{Q}^{(n)}\mathbf{R}^{(n)}$ another $\mathcal{O}(IR^4)$ operations, where matrices $\mathbf{Q}^{(n)}$ are of size $I \times R^2$ and $\mathbf{R}^{(n)}$ of size $R^2 \times R^2$. Now updating tensor $\hat{\mathcal{H}}$ can be done similarly as in Section 5.3, with $I = R^2$ and $\mathbf{D}^{(n)} = \mathbf{R}^{(n)^T}$, through steps (5.7) and (5.8), resulting in $\mathcal{O}(R^8)$ operations and $\mathcal{O}(R^4)$ memory.

Performing HOSVD on $R^2 \times R^2 \times R^2$ tensor $\hat{\mathcal{H}}$ in combination with randomized algorithm applied on matrices $\hat{\mathbf{H}}_{(n)}\hat{\mathbf{H}}_{(n)}^T$ with predefined multilinear rank (R, R, R) requires $\mathcal{O}(R^7)$ operations and $\mathcal{O}(R^6)$ memory (see Sections 3.5.3). Finally, updating $I \times R$ factor matrices needs $\mathcal{O}(IR^3)$ operations.

In total, the HOSVD2 algorithm requires $\mathcal{O}(IR^4 + R^8)$ operations and $\mathcal{O}(IR^2 + R^6)$ memory.

HOSVD3 algorithm

The HOSVD3 algorithm uses the combination of Lanczos method and structure-exploiting matrix-vector multiplication from Section 5.1, together with calculation of core tensor from Section 5.3. It is presented in Algorithm 13.

To get R left singular vectors of matrix $\mathbf{Z}_{(n)}$ we have to perform R matrix-vector multiplications inside Lanczos method, and each multiplication requires $\mathcal{O}(I^2R^2 + IR^4 + R^5)$ operations and $\mathcal{O}(I^2 + IR^2 + R^4)$ memory if using Variant A of Step 3, or $\mathcal{O}(IR^4 + R^5)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory if using Variant B. Additionally, performing spectral decomposition on $R \times R$ tridiagonal symmetric matrix and forming the matrix of R left singular vectors requires $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory (see Section 2.3.1). The

calculation of core tensor requires $\mathcal{O}(IR^3 + R^6)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory.

Depending on the variant used in Step 3 of matrix-vector multiplication, which depends on the relation between I and R, we end up with:

- if $I < R^2$, the HOSVD3 algorithms requires $\mathcal{O}(I^2R^3 + IR^5 + R^6)$ operations and $\mathcal{O}(I^2 + IR^2 + R^4)$ memory,
- if $I \geq R^2$, the HOSVD3 algorithms requires $\mathcal{O}(IR^5+R^6)$ operations and $\mathcal{O}(IR^2+R^4)$ memory.

HOSVD4 algorithm

The HOSVD4 algorithm, presented in Algorithm 14, uses randomized algorithm for calculation of left singular vectors of matrices $\mathbf{Z}_{(n)}$, but instead of multiplying matrix with random vectors, we multiply with Kronecker products of random vectors, which follows the structure of the matrices.

To get R left singular vectors, we need to perform R such multiplication, each of which requires $\mathcal{O}(IR^2 + R^5)$ operations, as explained in Section 5.2. Updating core tensor $\hat{\mathcal{H}}$ as in Section 5.3 requires $\mathcal{O}(IR^3 + R^6)$ operations and $\mathcal{O}(IR^2 + R^4)$ memory.

All together, the HOSVD4 algorithm results in $\mathcal{O}(IR^3+R^6)$ operational and $\mathcal{O}(IR^2+R^4)$ memory requirements.

6.1.2 Complexities for $N > 3$

The discussion above on the complexity easily generalizes to $N > 3$, with $\mathcal{X} \in \mathbb{R}^{I\times\cdots\times I}$, $\mathcal{F} \in \mathbb{R}^{R\times\cdots\times R}$ and $\mathbf{A}^{(n)} \in \mathbb{R}^{I\times R}$, for $n = 1, 2 \ldots, N$, and $R < I$.

HOSVD1 algorithm

From Section 3.5.2 we know that forming full tensors $\mathcal{X}$ and $\mathcal{Y}$ out of their Tucker representation requires $\mathcal{O}(I^N R)$ operations and $\mathcal{O}(I^N)$ memory, while multiplying $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ takes $\mathcal{O}(I^N)$ operations. Applying randomized algorithm to $I \times I$ matrix $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$, with $I \times I^{N-1}$ matrix $\mathbf{Z}_{(n)}$, to get R leading left singular vectors $\mathbf{Z}_{(n)}$ takes $\mathcal{O}(I^N R)$ operations and $\mathcal{O}(I^{N-1}R)$ memory (see Section 2.3.2). Finally, forming the core tensor is equivalent to forming full tensors $\mathcal{X}$ and $\mathcal{Y}$. All together, HOSVD1 algorithm requires $\mathcal{O}(I^N R)$ operations and $\mathcal{O}(I^N)$ memory.

HOSVD2 algorithm

Just as with $N = 3$, we first form $I \times R^2$ factor matrices $\hat{\mathbf{C}}^{(n)}$, $n = 1, 2, \ldots, N$, within $\mathcal{O}(IR^2)$ operations and their QR decompositions within $\mathcal{O}(IR^4)$ operations, using $\mathcal{O}(IR^2 + R^4)$ memory. Now we have to update $R^2 \times \cdots \times R^2$ tensor $\hat{\mathcal{H}}$ of order N, which again can be done similarly as in Section 5.3, with $I = R^2$ and $\mathbf{D}^{(n)} = \mathbf{R}^{(n)^T}$ of size $R^2 \times R^2$. Matrices $\mathbf{F}_{(1)}$ and $\mathbf{G}_{(1)}$ are now $R \times R^{N-1}$ matrices, so creating R^2 vectors $\mathbf{m}_j$ from (5.7), now of size R^{2N-2}, takes $\mathcal{O}(R^{2N+1})$ operations. The multiplication (5.8) is then done in a loop as

$$\left(\mathbf{R}^{(N)} \otimes \cdots \otimes \mathbf{R}^{(3)} \otimes \mathbf{R}^{(2)}\right) \mathbf{m}_j = \Big[\left(\mathbf{R}^{(N)} \otimes \cdots \otimes \mathbf{R}^{(3)}\right) \otimes \mathbf{R}^{(2)} \Big] \mathbf{m}_j,$$

by applying Kronecker product property (2.2.1.(5)) in each iteration; all together resulting in $\mathcal{O}(R^{2N+2})$ operations and $\mathcal{O}(R^{2N})$ memory.

Running HOSVD algorithm on tensor $\hat{\mathcal{H}}$ in combination with randomized algorithm applied on matrices $\hat{\mathbf{H}}_{(n)}\hat{\mathbf{H}}_{(n)}^T$ for getting R left singular vectors of $\hat{\mathbf{H}}_{(n)}$ requires $\mathcal{O}(R^{2N+1})$ operations and $\mathcal{O}(R^{2N-1})$ memory (see Section 3.5.3). Finally, for updating $I \times R$ factor matrices we need $\mathcal{O}(IR^3)$ operations.

Overall, the complexity for HOSVD2 algorithm is $\mathcal{O}(IR^4 + R^{2N+2})$ operations and $\mathcal{O}(IR^2 + R^{2N})$ memory.

HOSVD3 algorithm

When generalizing matrix-vector multiplication from Section 5.1 to order $N > 3$, Step 1 and Step 5 remain the same, they require $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory, while Step 2 and Step 4 now apply on $R^2 \times \cdots \times R^2$ tensor $\mathcal{F} \otimes \mathcal{G}$, resulting in $\mathcal{O}(R^{2N-1})$ operations and $\mathcal{O}(R^{2N-2})$ and $\mathcal{O}(R^N)$ memory, respectively.

Step 3 will be performed in a loop; for Variant A, we first create vector $\mathbf{w}_3$ in $N - 2$ steps by

$$\begin{aligned} \mathbf{w}_3 &= \Big[\left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \Big] \mathbf{w}_2 = \\ &= \Big[\Big\{ \left(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \Big\} \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \Big] \mathbf{w}_2 \end{aligned}$$

and then $\mathbf{w}_4$ similarly, in $\mathcal{O}(\sum_{n=1}^{N-1} I^n R^{2(N-n)})$ operations and $\mathcal{O}(\sum_{n=1}^{N-1} I^{N-n} R^{2n-2})$ memory. In Variant B we get vector $\mathbf{w}_4$ in $N - 2$ steps by

$$\mathbf{w}_4 = \Big[\left(\mathbf{A}^{(N)^T} \odot \mathbf{B}^{(N)^T}\right) \otimes \cdots \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right) \Big]$$

$$\Big[\big(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\big) \otimes \cdots \otimes \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\Big]\mathbf{w}_2$$
$$= \Big[\big(\mathbf{A}^{(N)^T} \odot \mathbf{B}^{(N)^T}\big)\big(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\big) \otimes \cdots \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\Big]\mathbf{w}_2$$
$$= \Big[\Big\{\big(\mathbf{A}^{(N)^T} \odot \mathbf{B}^{(N)^T}\big)\big(\mathbf{A}^{(N)} \odot^T \mathbf{B}^{(N)}\big) \otimes \cdots \otimes \big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\Big\}$$
$$\otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\Big]\mathbf{w}_2,$$

requiring $\mathcal{O}(IR^{2N-2})$ operations and $\mathcal{O}(IR^{2N-2})$ memory. To get R left singular vectors of matrix $\mathbf{Z}_{(n)}$ we have to do this multiplication R times. The rest of the Lanczos method takes $\mathcal{O}(IR^2)$ operations.

Forming the core tensor following the procedure described in Section 5.3 requires creating R vectors $\mathbf{m}_j$ of length R^{2N-2} from (5.7), which is done within $\mathcal{O}(R^{2N})$ operations and $\mathcal{O}(R^{2N-2})$ memory, and then the multiplication (5.8), which we perform in a loop as

$$\Big(\mathbf{D}^{(N)^T} \otimes \cdots \otimes \mathbf{D}^{(3)^T} \otimes \mathbf{D}^{(2)^T}\Big)\,\mathbf{m}_j = \Big[\Big(\mathbf{D}^{(N)^T} \otimes \cdots \otimes \mathbf{D}^{(3)^T}\Big) \otimes \mathbf{D}^{(2)^T}\Big]\mathbf{m}_j,$$

ending up with the complexity of $\mathcal{O}(R^{2N})$ operations and $\mathcal{O}(R^{2N-2})$ memory.

Overall, for HOSVD3 algorithm,

- when $I < R^2$, using Variant A in Step 3 of matrix-vector multiplication,

  $$\mathcal{O}(\sum_{n=1}^{N-1} I^n R^{2(N-n)+1}) \text{ operations and } \mathcal{O}(\sum_{n=1}^{N-1} I^{N-n} R^{2n-2} + R^{2N-2}) \text{ memory}$$

  is required,

- while when $I \geq R^2$ and Variant B is used in Step 3 of matrix-vector multiplication, it requires

  $$\mathcal{O}(IR^{2N-1} + R^{2N}) \text{ operations and } \mathcal{O}(IR^{2N-4} + R^{2N-2}) \text{ memory.}$$

HOSVD4 algorithm

Performing matrix-vector multiplication from Section 5.2, where the vector we are multiplying with is now of the form $\mathbf{w} = \mathbf{x}_N \otimes \cdots \otimes \mathbf{x}_2$, with $\mathbf{x}_n$ of length I, requires forming vectors $\tilde{\mathbf{x}}_n = \Big(\mathbf{A}^{(n)^T} \odot \mathbf{B}^{(n)^T}\Big)\,\mathbf{x}_n$, for $n = 2, 3 \ldots, N$, and that takes $\mathcal{O}(IR^2)$ operations and $\mathcal{O}(IR)$ memory. Then we form vector $\mathbf{z}_1$ of length R^{2N-2} from (5.5) as

$$\mathbf{z}_1 = \tilde{\mathbf{x}}_N \otimes \tilde{\mathbf{x}}_{N-1} \otimes \cdots \otimes \tilde{\mathbf{x}}_2$$

in $\mathcal{O}(R^{2N-2})$ operations. Afterward, we follow the procedure from Section 5.2, which overall, including now different dimension of tensors $\mathcal{F}$ and $\mathcal{G}$, requires $\mathcal{O}(IR^2 + R^{2N-1})$ operations and $\mathcal{O}(R^N)$ memory.

Inside Stage A of the randomized algorithm we perform R such matrix-vector multiplications, while Stage B with matrix $\mathbf{Z}_{(n)}$ of size $I \times I^{N-1}$ requires $\mathcal{O}(I^N R)$ operations; see Section 2.3.2.

Forming $R \times \cdots \times R$ core tensor $\hat{\mathcal{H}}$ goes exactly as with HOSVD3 algorithm within $\mathcal{O}(R^{2N})$ operations and $\mathcal{O}(R^{2N-2})$ memory.

All together, HOSVD4 algorithm requires $\mathcal{O}(IR^3 + R^{2N})$ operations and $\mathcal{O}(IR^2 + R^{2N-2})$ memory.

6.2 Numerical experiments

We have implemented and tested our algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4 from Section 4.2, analyzed in Section 6.1, for both fixed-rank and fixed precision problem and on tensors of different order, size and multilinear rank. The computational environment is set in Section 1.4.

6.2.1 Execution times for function-related tensors

To test our algorithms, we generate function-related tensors $\mathcal{X}$, $\mathcal{Y}$ by evaluating the functions

$$f(x, y, z) = \frac{1}{x + y + z}, \quad g(x, y, z) = \frac{1}{\sqrt{x + y + z}}$$

on the grid $\{0.1, 0.2, \ldots, I/10\}$ for x, y, z. The following table reports the approximate multilinear ranks (R, R, R) of $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ obtained when discarding singular values smaller than 10^{-8}:

	$I = 50$	$I = 100$	$I = 200$	$I = 400$
$\mathcal{X}$	11	12	15	17
$\mathcal{Y}$	11	12	15	17
$\mathcal{Z}$	12	13	15	17

We first apply Algorithm 8 with $\epsilon = 10^{-8}$ to get Tucker representations of $\mathcal{X}$ and $\mathcal{Y}$ and then run the four different HOSVD algorithms to get Tucker representation of $\mathcal{Z}$, targeting an accuracy of 10^{-8}, and then compare the results.

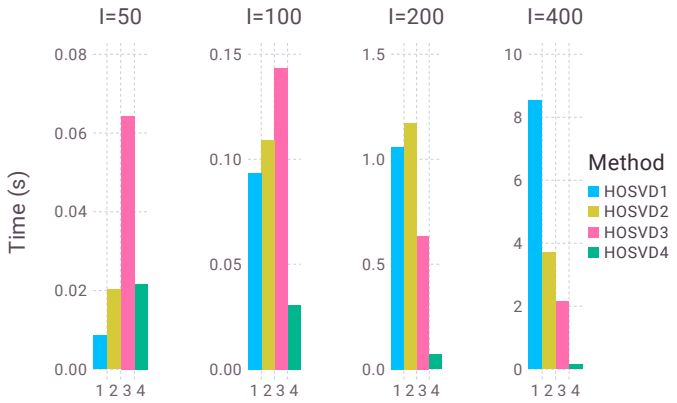


Figure 6.1: Execution times (in seconds) of HOSVD algorithms applied to function-related tensors from Section 6.2.1.

Figure 6.1 shows the execution times. Except for $I = 50$, HOSVD4 outperforms all other algorithms. For example, for $I = 400$, HOSVD4 requires 0.16 seconds while HOSVD3, the next best algorithm, requires 2 seconds.

6.2.2 Accuracy and perturbation bounds

We compared the accuracy of the algorithms studied in Section 4.2.2 in dependence of the oversampling parameter p used in Lanczos tridiagonalization algorithm (Algorithm 1) and Stage A of the randomized algorithm (Algorithm 2). We used the Tucker representations $\mathcal{X}_T$ and $\mathcal{Y}_T$ of the function-related tensors $\mathcal{X}$ and $\mathcal{Y}$ from Section 6.2.1 with $I = 50$ and required the multilinear rank to be (R, R, R), with $R = 9$ and $R = 15$. Figure 6.2 shows the resulting errors calculated as $\|\text{full}(\mathcal{X}_T) * \text{full}(\mathcal{Y}_T) - \text{full}(\mathcal{T})\|_F$, where $\mathcal{T}$ is the output of the corresponding algorithm. Generally, it can be said that very small values of p are sufficient for all algorithms, in the sense that the limiting accuracy determined by the choice of R is reached. HOSVD4 is more accurate in situations where the limiting accuracy is below $\mathcal{O}(10^{-8})$, because it works directly with $\mathbf{Z}_{(n)}$ instead of the Gramian $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$.

To test the error and perturbation bounds from Section 4.2.2 we will rerun the algorithms on the same function-related tensors with $I = 50$, now defining $\varepsilon = \varepsilon_P = 10^{-8}$, setting $p = 10$ and using tolerances for Lanczos and randomized algorithms as explained in Section 4.2.2. We perturb tensors $\mathcal{X}$ and $\mathcal{Y}$ as $\tilde{\mathcal{X}} = \mathcal{X} + \varepsilon_P \mathcal{X}^P$ and $\tilde{\mathcal{Y}} = \mathcal{Y} + \varepsilon_P \mathcal{Y}^P$, with $\mathcal{X}^P$ and $\mathcal{Y}^P$ randomly generated tensors such that $\|\mathcal{X}^P\|_F \leq 1$ and $\|\mathcal{Y}^P\|_F \leq 1$.

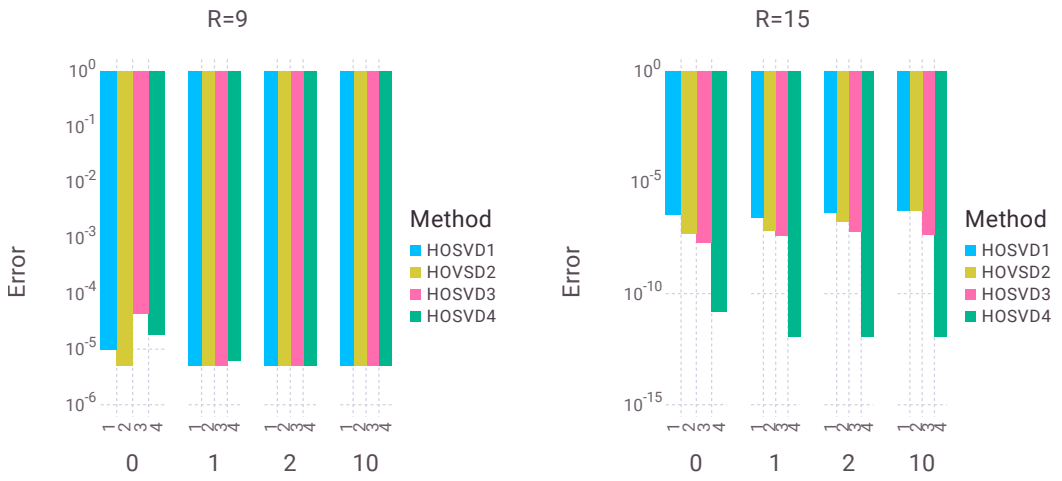


Figure 6.2: Error of HOSVD3 and HOSVD4 versus oversampling parameter $p = 0, 1, 2, 10$.

Tensor $\mathcal{Z}$ denotes the exact Hadamard product, tensor $\hat{\mathcal{Z}}$ output of our algorithms and $\tilde{\hat{\mathcal{Z}}}$ the exact Hadamard product of the perturbed tensors. Since each matricization $\mathbf{Z}_{(n)}$ has quickly decaying singular values, precisely

$$34.2866, 9.29112, 1.96579, 0.365423, 0.0620113, 0.00973023, 0.00141463,$$

$$0.000191021, 2.4126 \times 10^{-5}, 2.86846 \times 10^{-6}, 3.179 \times 10^{-7}, 2.38147 \times 10^{-8},$$

we can use Remark 4.2.3 and, since the first two singular values dominate, set $C_n = \sqrt{2}$, i.e. $C = N\sqrt{2}$. We use B_{err} to denote error bounds (4.10) and (4.7) and B_{pert} to denote perturbation bounds (4.11) and (4.9). The results are presented in Table 6.1.

Table 6.1: Comparison of real errors with bounds from Section 4.2.2 for algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4.

	$\|\mathcal{Z} - \hat{\mathcal{Z}}\|_F$	B_{err}	$\|\tilde{\hat{\mathcal{Z}}} - \hat{\mathcal{Z}}\|_F$	B_{pert}
HOSVD1	4.1134214×10^{-5}	4.2426406×10^{-4}	4.1134219×10^{-5}	4.2625748×10^{-4}
HOSVD2	4.113422×10^{-5}	4.2426406×10^{-4}	4.1134226×10^{-5}	4.2625748×10^{-4}
HOSVD3	4.1440793×10^{-5}	4.2426406×10^{-4}	4.1440794×10^{-5}	4.2625748×10^{-4}
HOSVD4	$4.0898896 \times 10^{-11}$	3×10^{-8}	1.082579×10^{-9}	2.0234116×10^{-6}

6.2.3 Execution times for random tensors

To test fixed-rank problem, we choose tensors $\mathcal{X}$ and $\mathcal{Y}$ of prescribed multilinear rank by letting all coefficients of the Tucker format contain random numbers from the standard normal distribution.

Tensors of order N=3

Setting $\mathcal{X}$ and $\mathcal{Y}$ to be $I \times I \times I$ tensors of multilinear rank (R, R, R), we measure the time needed by our algorithms for truncating $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ back to multilinear rank (R, R, R).

The times obtained for $I = 50, 100, 200, 400$ with respect to R are shown in Figure 6.3.

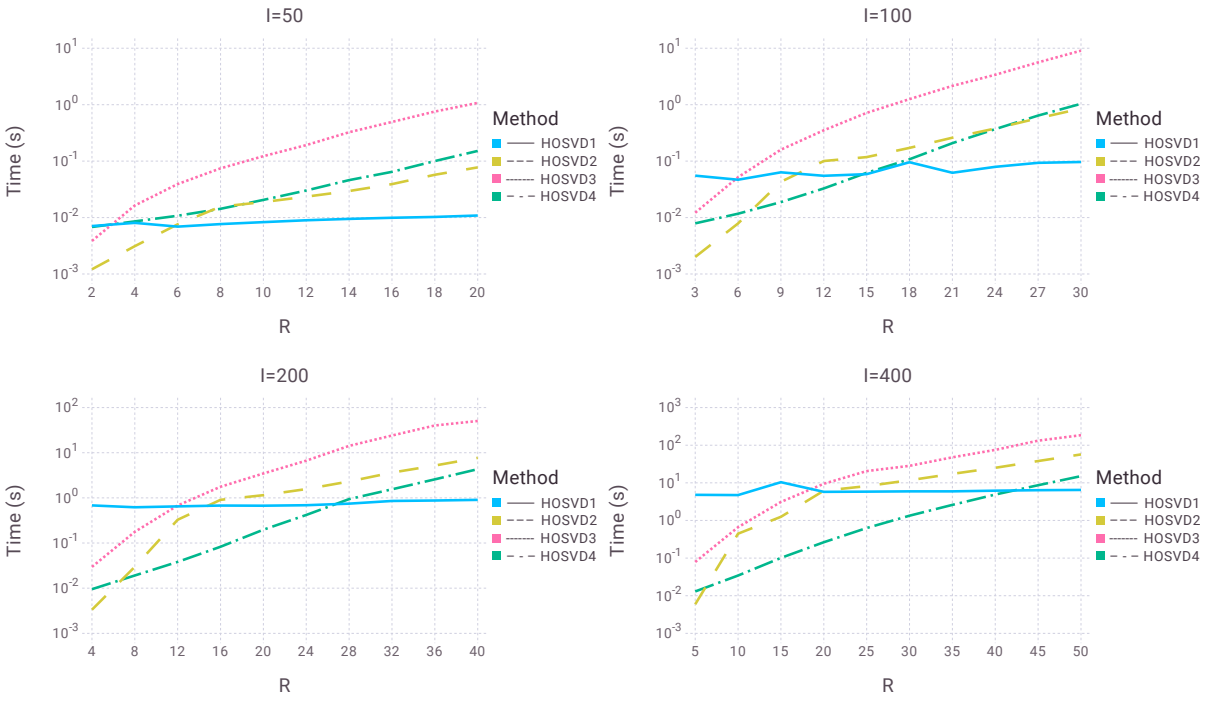


Figure 6.3: Execution times (in seconds) of HOSVD algorithms applied to random tensors of size $I \times I \times I$ and multilinear rank (R, R, R).

As expected the performance of HOSVD1, based on forming the full tensor $\mathcal{Z}$, depends only mildly on R. All other algorithms have a cost that is initially smaller than HOSVD1 but grows as R increases. The observed breakeven points match the theoretical breakeven points between $R = \mathcal{O}(I^{2/5})$ and $R = \mathcal{O}(I^{3/5})$ quite well.

For larger I, it is impossible to store $\mathcal{Z}$ and hence Figure 6.4 only displays the times of HOSVD2, HOSVD3, and HOSVD4 for $I = 500, 900$. Among these three algorithms, HOSVD4 is nearly always the best.

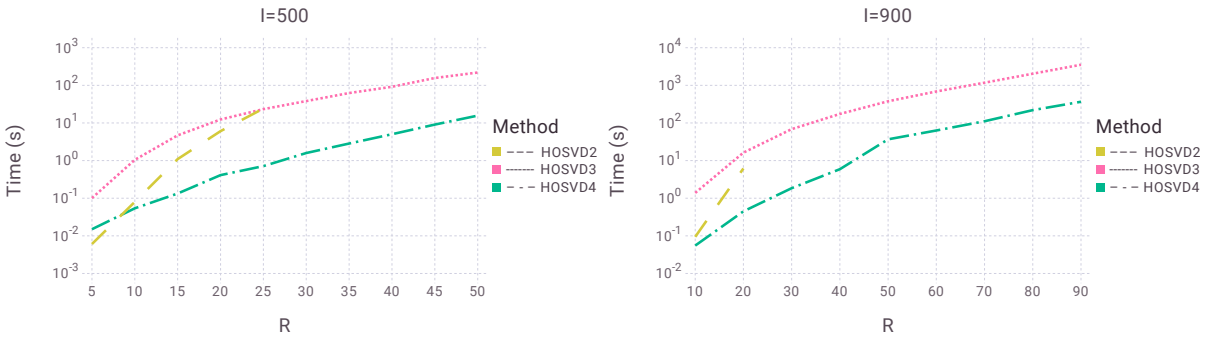


Figure 6.4: Execution times (in seconds) of HOSVD algorithms applied to random tensors of size $I \times I \times I$ and multilinear rank (R, R, R).

Figure 6.5 shows the performance of HOSVD4 with respect to R and I. Note that some of the displayed configurations, such as $I = 5\,000$ and $R = 90$, are intractable for any other algorithm discussed in this thesis.

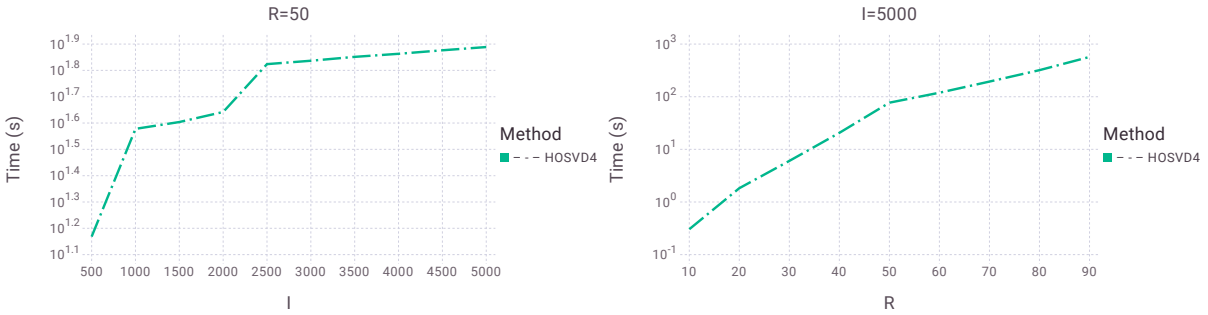


Figure 6.5: Execution times (in seconds) of HOSVD4 applied to random tensors of size $I \times I \times I$ and multilinear rank (R, R, R).

What about when tensors $\mathcal{X}$ and $\mathcal{Y}$ have different multilinear ranks? Figure 6.6 shows execution times for the four algorithms in that case, when $\mathcal{X}$ and $\mathcal{Y}$ are $I \times I \times I$ tensors, one tensor has multilinear rank (R, R, R) and another $(10R, 10R, 10R)$, with $I = 200$ and $I = 400$, and requested multilinear rank for $\mathcal{Z}$ is $(20, 20, 20)$. Again HOSVD4 algorithm stands out.

Same is in the case when $\mathcal{X}$ and $\mathcal{Y}$ have modes of different sizes and different n-ranks. Several combinations are presented in Figure 6.7, where the size of tensors $\mathcal{X}$ and $\mathcal{Y}$ is stated in the title of each subfigure and their multilinear ranks, together with the requested multilinear rank for $\mathcal{Z}$, are stated in the caption.

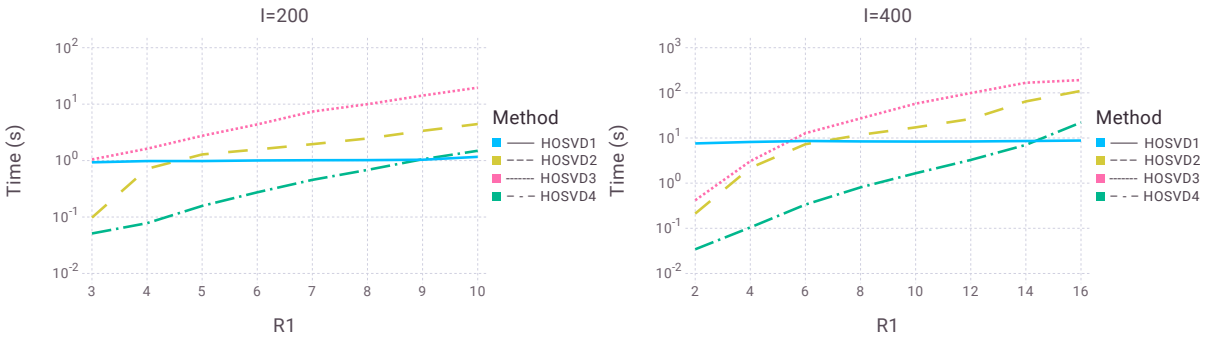
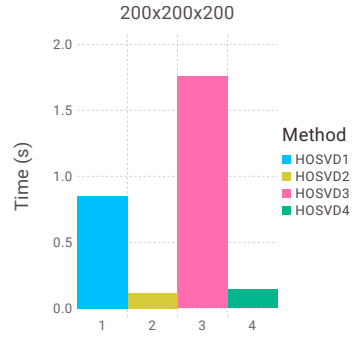
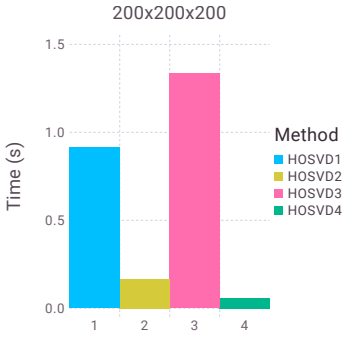


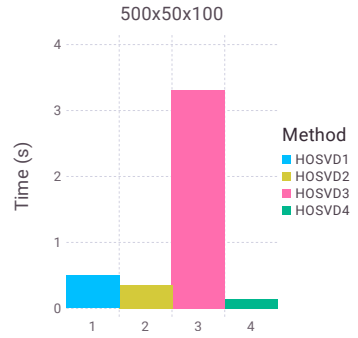
Figure 6.6: Execution times (in seconds) of HOSVD algorithms applied to random tensors of size $I \times I \times I$ and multilinear ranks (R, R, R) and $(10R, 10R, 10R)$ with requested multilinear rank of the Hadamard product $(20, 20, 20)$.



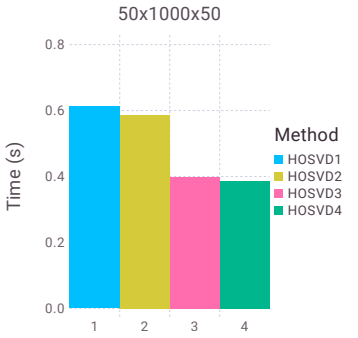
(a) $P = Q = R = (2, 20, 40)$



(b) $P = (2, 20, 40)$, $Q = (40, 20, 2)$, $R = (20, 20, 20)$



(c) $P = (50, 5, 10)$, $Q = (10, 5, 50)$, $R = (20, 20, 20)$



(d) $P = (5, 100, 5)$, $Q = (2, 20, 2)$, $R = (10, 10, 10)$

Figure 6.7: Execution times (in seconds) of HOSVD algorithms applied to random tensors $\mathcal{X}$ and $\mathcal{Y}$ of different sizes, with P and Q being the multilinear ranks of $\mathcal{X}$ and $\mathcal{Y}$, respectively, and R the requested multilinear rank for the product $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$.

Tensors of order N=4 and N=5

Now we set $\mathcal{X}$ and $\mathcal{Y}$ to be $I \times \cdots \times I$ tensors of multilinear rank $(R, \ldots, R)$ and measure the time needed by our algorithms for truncating $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ back to multilinear rank $(R, \ldots, R)$. Figure 6.8 shows results for $N = 4$ and Figure 6.9 for $N = 5$.

As with tensors of order $N = 3$, when increasing I, forming full tensor $\mathcal{Z}$ becomes impossible, so HOSVD1 algorithm does not work, while HOSVD2 works only for small R. HOSVD4 gives best results for these cases, too.

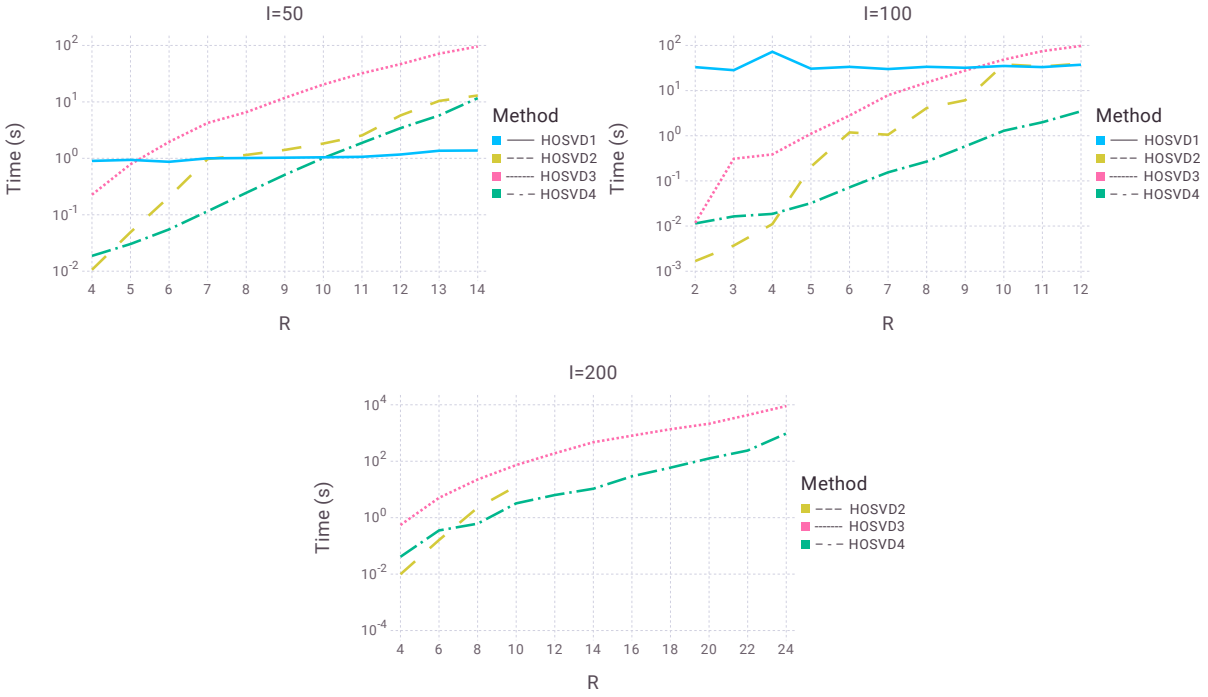


Figure 6.8: Execution times (in seconds) of HOSVD algorithms applied to the Hadamard product of random tensors of size $I \times I \times I \times I$ and multilinear rank (R, R, R, R).

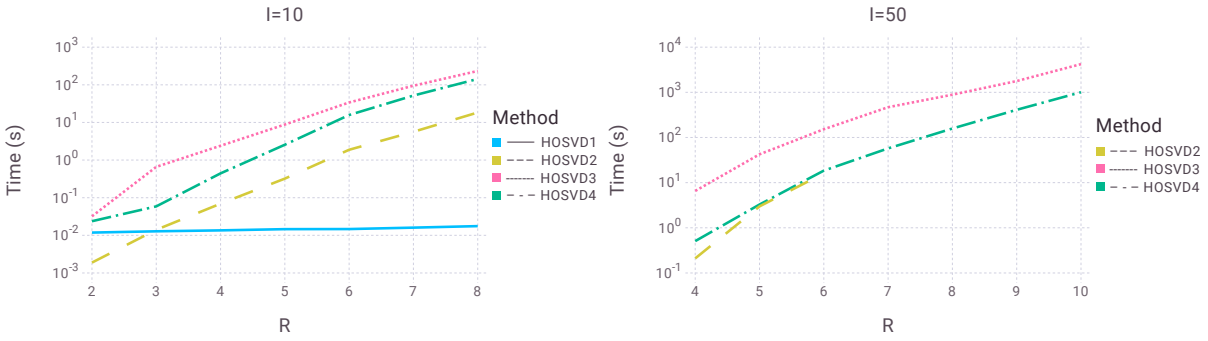


Figure 6.9: Execution times (in seconds) of HOSVD algorithms applied to the Hadamard product of random tensors of size $I \times I \times I \times I \times I$ and multilinear rank (R, R, R, R, R).

6.2.4 Conclusion and recommendation

Based on observations from the previous sections, we conclude the following recommendation:

- Use HOSVD1 when $\mathcal{Z}$ fits into memory *and* the involved multilinear ranks are expected to exceed $I^{3/5}$.
- In all other situations, use HOSVD4.

6.2.5 Testing complexities

Here we test that the presented complexities of algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4 match their actual complexities. Using randomly generated tensors $\mathcal{X}$ and $\mathcal{Y}$, as in Section 6.2.3, of size $I \times I \times I$ and multilinear ranks (R, R, R), with $t(I, R)$ we denote the time needed by our algorithms to recompress $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ to multilinear rank (R, R, R). We measure times for two different pairs (I_1, R_1) and (I_2, R_2) and compare the ratio $t(I_1, R_1)/t(I_2, R_2)$ with the expected ratio e, which is for each algorithm defined as follows:

- HOSVD1. Since I^3 is the dominating term in the complexity of HOSVD1 algorithm, we choose $I_2 = 2I_1$ and $R_1 = R_2$, so the expected ratio is

$$e = \frac{R_1 I_1^3}{R_2 I_2^3} = \frac{R_1 I_1^3}{R_1 2^3 I_1^3} = \frac{1}{2^3},$$

- HOSVD2. Choosing $I_2 = 2^4 I_1$ and $R_2 = 2R_1$, we have

$$e = \frac{R_1^4 I_1 + R_1^8}{R_2^4 I_2 + R_2^8} = \frac{R_1^4 I_1 + R^8}{2^4 R_1^4 2^4 I_1 + 2^8 R^8} = \frac{1}{2^8},$$

- HOSVD3 + Variant A. We use this variant when $I < R^2$ so we have to choose I and R accordingly. If $I_2 = 2I_1$ and $R_2 = 2R_1$, then

$$e = \frac{R_1^3 I_1^2 + R_1^5 I_1 + R_1^6}{R_2^3 I_2^2 + R_2^5 I_2 + R_2^6} = \frac{R_1^3 I_1^2 + R_1^5 I_1 + R_1^6}{2^3 R_1^3 2^2 I_1^2 + 2^5 R_1^5 2 I_1 + 2^6 R_1^6} \approx \frac{1}{2^6},$$

- HOSVD3 + Variant B. We use this variant when $I \geq R^2$ so we have to choose I and R accordingly. If $I_2 = 2I_1$ and $R_2 = 2R_1$, then

$$e = \frac{R_1^5 I_1 + R_1^6}{R_2^5 I_2 + R_2^6} = \frac{R_1^5 I_1 + R^6}{2^5 R_1^5 2 I_1 + 2^6 R^6} = \frac{1}{2^6},$$

- HOSVD4. Choosing $I_2 = 2^3 I_1$ and $R_2 = 2R_1$, we have

$$e = \frac{R_1^3 I_1 + R_1^6}{R_2^3 I_2 + R_2^6} = \frac{R_1^3 I_1 + R^6}{2^3 R_1^3 2^3 I_1 + 2^6 R^6} = \frac{1}{2^6},$$

The results are presented in Table 6.2 and we can see that the theoretical complexities match the actual complexities of the algorithms.

Table 6.2: Comparison of ratios $t(I_1, R_1)/t(I_2, R_2)$ and expected ratios e for algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4.

	I_1	I_2	R_1	R_2	$t(I_1, R_1)/t(I_2, R_2)$	e
HOSVD1	200	400	20	20	0.11615999	0.125
HOSVD2	50	800	10	20	0.00225188	0.00390625
HOSVD3 + Var. A	200	400	20	40	0.06338479	0.015625
HOSVD3 + Var. B	400	800	10	20	0.06642512	0.015625
HOSVD4	100	800	20	40	0.02764474	0.015625

Appendix A

Julia package

We have created a package for tensors and tensors in Tucker format in programming language Julia [5], following the nomenclature of MATLAB's Tensor Toolbox [3]. A notable difference to the Tensor Toolbox is that we do not construct a separate object `tensor` for dealing with full tensors but instead directly use built-in multidimensional arrays. Our package is available at `https://github.com/lanaperisa/TensorToolbox.jl` and has a standard Julia package structure:

```
TensorToolbox.jl/
├── src/
│   ├── TensorToolbox.jl
│   ├── tensor.jl
│   ├── ttensor.jl
│   └── helper.jl
└── test/
    ├── create_test_data.jl
    ├── thesisTests.jl
    ├── runtests.jl
    └── test_data_func.jld
```

The module *TensorToolbox* is contained in `TensorToolbox.jl`, functionality for full tensors is in `tensor.jl`, and functionality for tensors in the Tucker format is in `ttensor.jl`. Functions regarding matrices, such as the ones presented in Section 2.2, are set in `helper.jl` file. The object (or composite type) for tensors in Tucker format is called `ttensor` and consists of the fields `cten` (core tensor) and `fmat` (factor matrices), as well as `isorth` (flag for orthonormality of factor matrices).

```
type ttensor{T<:Number}
  cten::Array{T}
  fmat::Array{Matrix,1}
  isorth::Bool
end
```

All figures from this thesis can be recreated using functions from `test/thesisTests.jl` file, following instructions in the file.

In the following we present and explain the functionality of the package.

A.1 Tensors and basic tensor operations

As already mentioned, we do not construct a new object for tensors, but define them as multidimensional arrays, for example

```
X=rand(5,4,3)
Y=rand(5,4,3)
```

So all basic operations, such as addition, vectorization, element-wise multiplication, size, equality etc. are already built-in Julia. Frobenius norm of a tensor is also already implemented as function `vecnorm`. We have created additional functions, such as inner product and Kronecker product of tensors with $N \geq 3$.

```
innerprod(X,Y)
tkron(X,Y) #tensor Kronecker product
```

The n-rank and the multilinear rank of a tensor is calculated by functions `nrank` and `mrank`, which also accept tolerance.

```
n=2;nrank(X,n) #2-rank of X
mrank(X) #multilinear rank of X
n=1;nrank(X,1,1e-8) #1-rank of X, disarding singular values lower than
    1e-8
mrank(X,1e-8) #multilinear rank of X, disarding singular values lower
    than 1e-8
```

The mode-n matricization of a tensor can also be inverted.

```
n=1
Xn=tenmat(X,n) #mode-n matricization
X=matten(Xn,n,[5,4,3]) #transforming a matrix into a tensor of size 5
    x4x3 by mode n
```

The n-mode multiplications are implemented in functions `ttm` (tensor-times-matrix) and `ttv` (tensor-times-vector). We can use them to multiply a tensor by a matrix or

a vector, or a set of matrices or vectors, specifying the modes of multiplication. For example,

```
A=rand(6,5)
X1=ttm(X,A,1) #1-mode multiplication
B=rand(2,4)
X2=ttm(X,[A,B],[1,2]) #1-mode and 2-mode multiplications
X2=ttm(X,[A,B],-3) #same as above
C=rand(7,3)
X3=ttm(X,[A,B,C]) #same as ttm(X,[A,B,C],[1,2,3])
D=rand(5,2)
X4=ttm(X,D,1,'t') #1-mode multiplication with matrix transpose
```

and

```
a=rand(5)
X1=ttv(X,a,1) #1-mode multiplication
b=rand(4)
X2=ttv(X,[a,b],[1,2]) #1-mode and 2-mode multiplications
X2=ttv(X,[a,b],-3) #same as above
c=rand(3)
X3=ttv(X,[a,b,c]) #same as ttv(X,[a,b,c],[1,2,3])
```

The procedure presented in Section 3.5.1 is implemented in function `mkrontv`, which excepts both vectors and matrices for multiplication with matricized Kronecker product. In case when multiplying with matrix, the multiplication is performed column by column.

```
v=rand(4^2*3^2)
mkrontv(X,Y,v,1) #mode-1 matricization od Kronecker product of X and Y
    times vector v
V=rand(4^2*3^2,10)
mkrontv(X,Y,V,1) #mode-1 matricization od Kronecker product of X and Y
    times matrix V
```

Function `hosvd` implements Algorithms 6, 7 and 8. It creates (approximate) Tucker representation of a given tensor. Algorithm 8, with $\epsilon = 10^{-8}$ in combination with LAPACK function `gesvd!` [1] for calculation of left singular vectors of matricizations of tensors is the default one. We can also use Lanczos or randomized algorithm from Section 2.3 and combine it with truncated HOSVD with predefined multilinear rank (Algorithm 7).

One more option is to, for a given ϵ_{rel}, discard singular values lower then $\sigma_1 \cdot \epsilon_{rel}$, where σ_1 is the largest singular value. Values ϵ and ϵ_{rel} can also be vectors, in which case n-th component is applied for mode-n matricization. The behavior of Algorithm 6 can be achieved by setting multilinear rank to be equal to size of a tensor.

```
T1=hosvd(X) #same as hosvd(T,method="lapack",eps_abs=1e-8)
T2=hosvd(X,eps_abs=1e-5) #discard singular values lower than 1e-5
T2=hosvd(X,eps_abs=[1e-5,1e-2,1e-8]) #discard singular values lower than
    1e-5 for mode-1 matricization, lower than 1e-2 for mode-2
   matricization and lower than 1e-8 for mode-3 matricization
T3=hosvd(X,eps_rel=1e-3) #discard singular values lower than maximal
   singular value times 1e-3
T4=hosvd(X,method="lanczos")
T5=hosvd(X,method="randsvd")
T6=hosvd(X,reqrank=[2,2,2]) #appproximate Tucker representation with
   multilinear rank (2,2,2)
T7=hosvd(X,reqrank=[5,4,3]) #exact Tucker representation
```

Overview of all functions in `tensor.jl` is in the following table. Functions denoted by * are not part of MATLAB's Tensors Toolbox.

Table A.1: Overview of all functions for tensors.

Functions for tensors - `tensor.jl`	
`hosvd`*	Higher-order singular value decomposition (HOSVD).
`innerprod`	Inner product of two tensors.
`krontm`*	Kronecker product of two tensors times matrix (n-mode multiplication).
`matten`*	Matrix tensorization - fold matrix into a tensor.
`mkrontv`*	Multiplication of matricized Kronecker product of two tensors by a vector or a matrix.
`mrank`*	Multilinear rank of a tensor.
`mttkrp`	Matricized tensor times Khatri-Rao product.
`nrank`*	n-rank of a tensor.
`sthosvd`*	Sequentially truncated HOSVD.
`tenmat`	Tensor matricization - unfold tensor into matrix.
`tkron`*	Kronecker product of two tensors.
`ttm`	Tensor times matrix (n-mode multiplication).

`ttt`	Outer product of two tensors.
`ttv`	Tensor times vector (n-mode multiplication).

A.2 Tensors in Tucker format

Apart from calling HOSVD on a given tensor as in the previous section, tensors in Tucker format can be defined by their core tensor and factor matrices

```
A=MatrixCell(3) #alias for array of matices of length 3
A[1]=rand(10,5)
A[2]=rand(9,4)
A[3]=rand(8,3)
F=rand(5,4,3)
T=ttensor(F,A) #creates tensor in Tucker format with core tensor F and
    factor matrices from A
```

Since very often we need to create `ttensor` whose core tensor and factor matrices contain random numbers, we have created function `randttensor` to simplify the procedure. It creates `ttensor` whose coefficients contain random numbers from the standard normal distribution.

```
T1=randttensor([10,9,8],[5,4,3]) #creates ttensor of size 10x9x8 with
   core tensor of size 5x4x3
T2=randttensor(5,2,3) #same as randttensor([5,5,5],[2,2,2])
T3=randttensor(5,2,4) #same as randttensor([5,5,5,5],[2,2,2,2])
```

Now we can get core tensor and factor matrices of a tensor and check the orthonormality of factor matrices by

```
T.cten
T.fmat
T.isorth
```

If we want factor matrices to be orthonormal, we can reorthogonalize them as in (3.12) and create new `ttensor` with function `reorth`, or rewrite the existing tensor with function `reorth!`.

```
if T.isorth == false
   reorth!(T)
```

```
end
```

Basic functions for `ttensor` type include size, number of modes (dimensions), size of the core tensor and multiplication with a scalar.

```
size(T)
ndims(T) #number of modes of T
coresize(T) #same as size(T.cten)
a=3; mtimes(a,T) #same as a*T, with scalar a
```

The n-rank of a `ttensor` is calculated as rank of its nth factor matrix, and multilinear rank accordingly.

```
n=2; nrank(T,n)
mrank(T) #multilinear rank
```

Norm and inner product can be directly computed using Proposition 3.1.3, while Hadamard product without recompression is given by expression (4.2).

```
T1=randttensor([5,4,3],[2,2,2])
T2=randttensor([5,4,3],[3,3,3])
vecnorm(T1)
innerprod(T1,T2)
T1.*T2 #same as had(T1,T2)
```

Furthermore, on two `ttensors` we can perform addition (3.13) and subtraction, which is just addition with `ttensors` multiplied by -1.

```
T1+T2
T1-T2 #same as T1+(-1)*T2
```

The n-mode multiplication of a `ttensor` by a matrix or a vector is implemented by (3.1.3.(3)) and can be used just as the n-mode multiplication with a tensor from the previous section.

```
A=rand(12,10)
ttm(T,A,1) #1-mode multiplication
v=rand(10)
ttv(T,v,1) #1-mode multiplication
```

The process of creating a full tensor out of its Tucker representation, explained in

Section 3.5.2, is done by function `full`, and matricization of a `ttensor` is done by first creating full tensor and then using `tenmat` function on a tensor.

```
full(T)
n=2; tenmat(T,n) #mode-n matricization
```

We can recompress `ttensor` by performing HOSVD on its core tensor and then using (3.1.3.(3)), which is implemented in `hosvd` function, which can be used in combination with Lanczos and randomized algorithm, with specified tolerance or defined multilinear rank, the same way as with full tensors from previous section.

```
hosvd(T)
hosvd(T,method="randsvd")
hosvd(T,reqrank=[2,2,2])
```

The `ttensor.jl` file also contains operations from Chapters 4 and 5 needed for recompression of Hadamard products of `ttensors`. Matrix-vector multiplication $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T\mathbf{v}$ from Section 5.1 is done by function `mhadtv` and can also be used for matrix-matrix multiplication, in which case it performs it column by column. Variant B of Step 3 of the multiplication is the default one. The function can also perform multiplications $\mathbf{Z}_{(n)}\mathbf{v}$ and $\mathbf{Z}_{(n)}^T\mathbf{v}$.

```
v=rand(5)
n=1; mhadtv(T1,T2,v,n) #multiplication of Gramian of matricized Hadamard
     product of tensors T1 and T2 by vector v
V=rand(5,10)
mhadtv(T1,T2,V,n,variant='A') #multiplication of Gramian of matricized
    Hadamard product of tensors T1 and T2 by matrix V using Varinat A in
    Step 3
mhadtv(T1,T2,v,n,'t') #multiplication of transposed matricized Hadamard
    product of tensors T1 and T2 by vector v
w=rand(4*3)
mhadtv(T1,T2,w,n,'n') #multiplication of matricized Hadamard product of
    tensors T1 and T2 by vector w
```

The calculation of the core tensor of Hadamard products from Section 5.3 is done by function `hadcten`.

```
C=MatrixCell(3) #alias for array of matices of length 3
```

```
C[1]=rand(5,2)
C[2]=rand(4,2)
C[3]=rand(3,2)
F=hadcten(T1,T2,C)
```

Algorithms HOSVD1, HOSVD2, HOSVD3 and HOSVD4 from Section 4.2 are implemented in functions `hosvd1`, `hosvd2`, `hosvd3` and `hosvd4`. The default behavior is discarding singular values smaller than $\epsilon = 10^{-8}$, but we can also, for a given ϵ_{rel}, discard singular values lower then $\sigma_1 \cdot \epsilon_{rel}$, where σ_1 is the largest singular value, or predefine the multilinear rank and use different methods for calculation of the singular vectors. We present usage of `hosvd1`, for other algorithms it is analogous.

```
Z1=hosvd1(T1,T2)
Z2=hosvd1(T1,T2,reqrank=[2,2,2])
Z3=hosvd1(T1,T2,method="lanczos",reqrank=[5,4,3])
Z4=hosvd1(T1,T2,method="lapack",eps_rel=1e-3)
```

Overview of all functions in `ttensor.jl` is in the following table. Functions denoted by * are not part of MATLAB's Tensors Toolbox and the function `permutedims` is called `permute` in MATLAB.

Table A.2: Overview of all functions for tensors in Tucker format.

Functions for tensors in Tucker format - `ttensor.jl`	
`ttensor`	Construct Tucker tensor for specified core tensor and factor matrices.
`randttensor`	Construct random Tucker tensor.
`coresize`*	Size of the core of `ttensor`.
`full`	Construct full tensor (array) from `ttensor`.
`had (.*)`*	Hadamard product of two `ttensor`.
`hadcten`*	Construct core tensor for Hadamard product of two `ttensor` with specified factor matrices.
`hosvd`*	HOSVD for `ttensor`.
`hosvd1`*	HOSVD1 - computes Tucker representation of Hadamard product of two `ttensor`.
`hosvd2`*	HOSVD2 - computes Tucker representation of Hadamard product of two `ttensor`.

`hosvd3`*	HOSVD3 - computes Tucker representation of Hadamard product of two `ttensor`.
`hosvd4`*	HOSVD4 - computes Tucker representation of Hadamard product of two `ttensor`.
`innerprod`	Inner product of two `ttensor`.
`isequal (==)`	True if each component of two `ttensor` is numerically identical.
`lanczos`*	Lanczos algorithm adapted for getting factor matrices in algorithm `hosvd3`.
`mhadtv`*	Matricized Hadamard product of two `ttensor` multiplied by a vector or a matrix.
`minus (-)`*	Subtraction of two `ttensor`.
`mrank`*	Multilinear rank of a `ttensor`.
`msvdvals`*	Singular values of matricized `ttensor`.
`mtimes (*)`	Scalar multiplication for a `ttensor`.
`mttkrp`	Matricized `ttensor` times Khatri-Rao product.
`ndims`	Number of modes for `ttensor`.
`nrank`*	n-rank of a `ttensor`.
`nvecs`	Compute the leading mode-n vectors for `ttensor`.
`permutedims`	Permute dimensions of `ttensor`.
`plus (+)`*	Addition of two `ttensor`.
`randrange`*	Range approximation using randomized algorithm adapted for Hadamard product of two `ttensor`.
`randsvd`*	Randomized SVD algorithm adapted for getting factor matrices in algorithm `hosvd3`.
`reorth`*	Reorthogonalization of `ttensor`. Creates new `ttensor`.
`reorth!`*	Reorthogonalization of `ttensor`. Overwrites existing `ttensor`.
`size`	Size of a `ttensor`.
`tenmat`*	Unfold tensor into matrix - matricization.
`ttm`	Tensor times matrix for `ttensor` (n-mode multiplication).
`ttv`	Tensor times vector for `ttensor` (n-mode multiplication).
`uminus (-)`	Unary minus for `ttensor`.
`vecnorm`	Norm of a `ttensor`.

Bibliography

[1] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[2] P. Arbenz. Lecture notes on solving large scale eigenvalue problems, 2016. Available from `http://people.inf.ethz.ch/arbenz/ewp/Lnotes/lsevp.pdf`.

[3] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.

[4] Jonas Ballani and Daniel Kressner. Matrices with hierarchical low-rank structures. In *Exploiting hidden structure in matrix computations: algorithms and applications*, volume 2173 of *Lecture Notes in Math.*, pages 161–209. Springer, Cham, 2016.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: a fresh approach to numerical computing. *SIAM Rev.*, 59(1):65–98, 2017.

[6] F. Bonizzoni, F. Nobile, and D. Kressner. Tensor train approximation of moment equations for elliptic equations with lognormal coefficient. *Comput. Methods Appl. Mech. Engrg.*, 308:349–376, 2016.

[7] M. N. Da Costa, R. R. Lopes, and J. M. T Romano. Randomized methods for higher-order subspace separation. In *Signal Processing Conference (EUSIPCO), 24th European*, 2016.

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

[9] Sergey Dolgov, Boris N. Khoromskij, Alexander Litvinenko, and Hermann G. Matthies. Polynomial chaos expansion of random coefficients and the solution of

stochastic partial differential equations in the tensor train format. *SIAM/ASA J. Uncertain. Quantif.*, 3(1):1109–1135, 2015.

[10] A. Doostan and G. Iaccarino. A least-squares approximation of partial differential equations with high-dimensional random inputs. *J. Comput. Phys.*, 228(12):4332–4345, 2009.

[11] M. Espig, W. Hackbusch, A. Litvinenko, H. Matthies, and E. Zander. Efficient analysis of high dimensional data in tensor formats. In J. Garcke and M. Griebel, editors, *Sparse Grids and Applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 31–56. Springer, 2013.

[12] M. Filipović and A. Jukić. Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional Systems and Signal Processing*, 26(3):677–692, 2013.

[13] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, fourth edition, 2013.

[14] Michael Griebel and Helmut Harbrecht. Approximation of bi-variate functions: singular value decomposition versus sparse grids. *IMA J. Numer. Anal.*, 34(1):28–54, 2014.

[15] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[16] B. Hashemi and N. Trefethen. Chebfun in three dimensions. Technical report, 2016.

[17] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *J. Math. Phys*, 6(1):164–189, 1927.

[18] F. L. Hitchcock. Multiple invariants and generalized rank of a p-way matrix or tensor. *J. Math. Phys*, 7(1):39–79, 1927.

[19] T. G. Kolda. Multilinear operators for higher-order decompositions, 2006.

[20] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[21] D. Kressner and L. Periša. Recompression fo Hadamard products of tensors in Tucker format. *SIAM J. Sci. Comput.*, 39(5):A1879–A1902, 2017.

[22] D. Kressner and C. Tobler. Algorithm 941: `htucker`—a Matlab toolbox for tensors in hierarchical tucker format. *ACM Trans. Math. Softw.*, 40(3):22:1–22:22, 2014.

[23] N. Lee and A. Cichocki. Fundamental tensor operations for large-scale data analysis in tensor train formats, 2016.

[24] S. Ragnarsson. *Structured tensor computations: Blocking, symmetries and Kronecker factorizations*. PhD thesis, Cornell University, 2012.

[25] S. Ragnarsson and C. F. Van Loan. Block tensor unfoldings. *SIAM J. Matrix Anal. Appl.*, 33(1):149–169, 2012.

[26] R. Schneider and A. Uschmajew. Approximation rates for the hierarchical tensor format in periodic Sobolev spaces. *J. Complexity*, 30(2):56–71, 2014.

[27] H. D. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM J. Sci. Comput.*, 21(6):2257–2274, 2000.

[28] G. W. Stewart. *Matrix Algorithms. Vol. II*. SIAM, Philadelphia, PA, 2001. Eigensystems.

[29] L. N. Trefethen and D. Bau, III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[30] L. R. Tucker. Implications of factor analysis of three-way matrices for measurement of change. In C. W. Harris, editor, *Problems in measuring change.*, pages 122–137. University of Wisconsin Press, Madison WI, 1963.

[31] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal.*, 25(3):335–366, 2008.

Curriculum Vitae

Lana Periša was born in Split, Croatia on March 7th 1986, where she finished her basic formal education. She started studying at the Department of Mathematics of the University of Zagreb in 2005, got her B.Sc. in Mathematics in 2008 and M.Sc. in Applied mathematics in 2010. In 2011 she started working as an associate on a project run by Prof. Ivan Slapničar at the Faculty of Electrical and Mechanical Engineering and Naval Architecture (FESB), University of Split, and in the same year started Croatian doctoral program in Mathematics. In 2012 she became a research and teaching assistant at FESB, University of Split, where she continues working since.

She visited École Polytechnique Fédérale de Lausanne twice, in 2015 and 2016, where she worked with Prof. Daniel Kressner on tensor decomposition problems. This resulted in her doctoral thesis and a new Julia package for tensors and tensors in Tucker format called *TensorToolbox*. This work is published in SIAM Journal on Scientific Computing as

- D. Kressner and L. Periša. Recompression of Hadamard Products of Tensors in Tucker Format, *SIAM J. Sci. Comput.*, 39(5), A1879–A1902, 2017,

and was presented by Lana on *7th Workshop on Matrix Equations and Tensor Techniques* in Pisa, Italy in 2017.

In 2014 she was a local organizer of the summer school *Advanced Scientific Programming in Python* at University of Split, and during the years attended several international schools. In 2015 she finished online course *Machine Learning* from Stanford University on coursera.org.

She is a member of Split mathematical society, Society for Industrial and Applied Mathematics and European Women in Mathematics.

IZJAVA O IZVORNOSTI RADA

Ja, _____, student/ica Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu, s prebivalištem na adresi _____,OIB _____, JMBAG _____, ovim putem izjavljujem pod materijalnom i kaznenom odgovornošću da je moj završni/diplomski/doktorski rad pod naslovom:

_____

_____

_____, isključivo moje autorsko djelo, koje je u potpunosti samostalno napisano uz naznaku izvora drugih autora i dokumenata korištenih u radu.

U Zagrebu, _____

_____

Potpis