KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Numerical algorithms for tensor decompositions

Michiel Vandecappelle

Supervisor:
Prof. dr. ir. L. De Lathauwer

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2021

Numerical algorithms for tensor decompositions

Michiel VANDECAPPELLE

Examination committee:
Prof. dr. ir. P. Verbaeten, chair
Prof. dr. ir. L. De Lathauwer, supervisor
Prof. dr. ir. G. Samaey
Prof. dr. ir. P. Patrinos
Prof. dr. N. Gillis
(Université de Mons)
Univ.-Prof. Dr.-Ing M. Haardt
(Technische Universität Ilmenau)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2021

© 2021 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Michiel Vandecappelle, Kasteelpark Arenberg 10, box 2446, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

I still remember the first time I heard about tensors. It was May 2015 and I was reading the list of thesis proposals on the train between Leuven and Kortrijk. Their conceptual simplicity, yet deep mathematical foundation inspired me to choose one of the tensor topics proposed by prof. Lieven De Lathauwer. Little did I know that this simple choice would take me on a journey that would culminate with this text, more than six years later. As with any PhD, there were many ups and downs, but I have been very fortunate to have so many people that supported me along the way.

First of all, I would like to thank my promoter, Lieven, for his initial and continued trust in me, first as a master's thesis student and later as a PhD student. Your drive for good fundamental research and clear and detailed writing was truly inspiring. Many times, my reports and drafts led to late-night frustrations for you and, after getting them back with hardly any white space left on the pages, to disillusionment for me. Despite this, you never stopped giving detailed feedback and pushed me to become a better researcher. Lieven, it has been an honor to work with you.

Next, I would like to thank the members of my examination committee for carefully reading trough this lengthy PhD text, for asking insightful questions and for providing helpful suggestions to improve the thesis. Thank you, prof. Giovanni Samaey and prof. Panos Patrinos for agreeing to be part of my supervisory committee and for attending my intermediate presentations. Several times, you offered a different viewpoint and challenged me to look beyond the field of tensor research. Prof. Nicolas Gillis, thank you for the discussions on nonnegative factorizations and alternative cost functions during the SeLMA meetings and research days. They were always inspiring. Thank you, prof. Martin Haardt, for the detailed feedback on the PhD text and for attending almost all of my conference presentations. It was great to always see a familiar face and to discuss tensors over a beer or two on the other side of the planet. Prof. Pierre Verbaeten, thank you for chairing this examination committee.

I am grateful to the Research Foundation Flanders — FWO for supporting me through a grant for strategic basic research. Their support allowed me the freedom to choose my own research interests and to work on extending and improving our software package Tensorlab such that anyone in the world can use our algorithms.

I would like to thank my colleagues and friends, both at ESAT in Leuven and Kulak in Kortrijk, for the many successful collaborations, for sharing their knowledge and for making my time at KU Leuven much more enjoyable. Nico, thank you for teaching me all the ins and outs of Tensorlab and for coauthoring many of my articles. I am sure that Tensorlab 4.0 will release one day (although I will not make the same mistake as you did by offering a release date.) Martijn, thank you for sharing your passion for good presentations, both in papers and in slides; Jean-luc would certainly be proud. You and Frederik were there at the very start, emptying your red pens on the first draft of my master's thesis, and it has been a pleasure to work together during all these years. Stijn, thank you for the collaboration on Tensorlab+. It has been inspiring to see how much you have contributed in such a short time. Ignat, thank you for being my office buddy at Kulak and for making me question the weird things about the Dutch language. A few more of our intense table tennis breaks and we would have qualified for the Olympics. Eric, thank you for the close collaboration on the algebraic algorithms during the past few years and for reminding me how rich the mathematics behind tensors are. You may always let me know which restaurants are 'Eric-approved' as well. Patrick, thank you for taking me to the KVK soccer games. In just a few months, you became a real citizen of Kortrijk. Muzaffer, thank you for spicing up our lunch by suddenly asking a deep question about life or the universe.

I would also like to thank the other team members, both in Leuven and Kortrijk, with whom I had the pleasure to collaborate at some point during my PhD. Thank you, Nithin, Robin, Frederik, Otto, Rob, Bernhard, Chuan, Mikael, Alwin and Stijn. I am also grateful to the colleagues in the Biotensors project for offering ideas for concrete applications for my methods and for inviting me to lunches, parties and curve fever tournaments. Thank you, Sabine, Christos, Alexander, Simon, Ofelie, Bori, Griet, Bharath, Siba, Yissel and Claudio. I would also like to thank my Biomed team mates at the ESAT Sports day. In our pink shirts, we went undefeated in advanced volleyball and finally managed to win the advanced soccer tournament as well. Thank you Nico, Stijn, Christos, Jonathan, Rob, Dries, Simon, Simon, Mario and Jasper for letting me join your team. I would also like to thank the colleagues from mathematics and physics at Kulak, who invited me for lunch and coffee breaks early on, when our Kulak tensor team was still much smaller. Finally, I would like to thank the badminton squad at Kulak for our weekly practice. Thank you, Eef, for organizing the practices and Jonas, Charlotte, Dirk, Wim, Pieter, Pieter, Louise, Lore, Maarten and the others for the many intense games.

During my PhD, I have had the opportunity to travel to several international conferences in Europe and North America. I would like to thank everyone who made these trips unforgettable. Thank you Simon and Yissel, for accompanying

me to Kos Island in Greece. I will cherish our lunches at Romeo's or our 'no stress' road trip. Thank you, Jasper, for showing me your favorite curling spot in Calgary and for discussing volleyball over a selection of local beers. Thank you, Nico, for the late-night drive to Angel Fire and the days of hiking, disc golf and downhill mountain biking; special thanks to Jesper for being braver than Nico on the downhills. Thank you, Jesper for paying for my bus in Guildford when only NFC bank cards were accepted and for watching the soccer world championship together. Finally, thank you, Simon, Simon and Martijn for the pilgrimage to Santiago de Compostela on our way from A Coruña and the late-night discussions about controversial topics. Martijn, you still owe me a guided tour in Vigo, though.

I would also like to thank everyone outside of the academic world who attended the defense or simply helped me forget about tensors every once in a while, especially during the long months of lock-down and working from home. I would like to thank my teammates from Balti Kortrijk, with whom I have made great memories, both on and off the court. Special thanks to Michiel, Frederick, Maikel and Louis for the many late-night discussions and the genuine interest in my research and to Arthur, Ewout, Wout and Willem, who were always up for a game of beach volleyball, even in the middle of the Covid-winter. I would also like to thank the guys from 'Casa Rob' and the 'Homies' from Kuurne for the old school gaming nights and my friends from my student days at Kulak for the online board game sessions. I would like to thank Bert in particular, whom I shared a house with for two and a half years, including the lock-down. It was great to have a fellow PhD student to vent with, even if that was often in the middle of the night due to our busy schedules.

Lastly, I am grateful to my family for the support during all these years. *Mama en papa*, thank you for giving me the opportunity to study and to pursue this PhD, for providing a warm home to come back to, and for the many family dinners at home or at the seaside. I would also like to thank Koen and Heidi for making me feel welcome in their family, be it in Kortrijk or in France. Finally, I am forever indebted to my wonderful girlfriend, Ine. She celebrated with me when papers were accepted, she comforted me when drafts returned with hundreds of remarks and she kept me going when it seemed like the finish line only drifted further away. Ine, without you, this journey would have been a thousand times tougher.

Michiel Vandecappelle
September 2021

Abstract

Over the years, many algorithms have been designed to decompose a matrix into a product of other matrices. These matrix decompositions can be used to compress data with a minimal loss of information or for extracting meaningful components. More recently, tensor decompositions such as the canonical polyadic decomposition (CPD) and the low multilinear rank approximation (LMLRA) have been designed as higher-order generalizations of these matrix decompositions leading to better results in certain applications. While matrix methods offer a natural way to process data involving two variables, they often fail to recognize higher-order patterns. Tensors, in contrast, can have any number of modes and can preserve the higher-order structure of the data, which makes them highly useful to analyze multi-variable data. Second, tensor decompositions are often unique in cases where their matrix counterparts are not. As fewer (artificial) restrictions such as orthogonality or triangle structure have to be imposed on the terms to enforce their uniqueness, this can lead to a decomposition in more meaningful terms.

Current tensor decomposition methods commonly make two assumptions about the data supplied by the user. First, they assume that all data is available at once and second, they consider noise on the data to be additive and Gaussian distributed. In many applications, these assumptions are justified and one can decompose the tensor using standard batch methods that minimize the least-squares distance between the tensor and the low-rank model. However, as tensor methods are applied to more and more real-world problems, the number of cases where these assumptions are clearly violated increases as well. Any real-time application, for instance, violates the assumption of having all data available at the start. In such applications, such as process control or patient monitoring, new data arrive at certain time intervals. This data should be included in the tensor model without recomputing the full decomposition from scratch, as such approaches are not time- nor resource-efficient. Similarly, it is not hard to find applications where the assumption of additive Gaussian noise is not satisfied. Audio data, for instance, is generally modelled using a non-Gaussian noise distribution while low-light imaging assumes Poisson distributed pixel intensities. When the real and imaginary parts of a signal are collected independently, as is typically the case in MRI imaging, the noise distribution of the signal magnitude is Rician instead of Gaussian. For these applications, fitting a low-rank model to the tensor with the least-squares

distance as cost function will lead to suboptimal solutions while a suitable choice of an alternative cost function can provide a model that corresponds better to the underlying components.

In this thesis, we provide methods that can still efficiently compute a suitable tensor decomposition when either of the aforementioned assumptions is not satisfied. First, we propose updating methods for both the CPD and the LMLRA. These methods start from an existing tensor decomposition and efficiently update this decomposition when new data arrives. By exploiting the multilinear structure of the tensor models, these methods are both efficient and accurate in tracking low-rank representations of the data. When desired, one can even use a weighted least-squares approach to weight certain tensor entries more heavily than others, e.g., when the uncertainty on the collected tensor entries varies between sensors. The performance of the CPD method is demonstrated for the monitoring of brain hemodynamics coupling in neonates. A second contribution is that we extend the existing least-squares CPD algorithms to other cost functions. This makes it possible to use a dedicated cost function for the application at hand and to obtain better solutions than when standard least-squares approaches are used. We focus in particular on the class of β-divergence cost functions, of which the least-squares distance and the Kullback—Leibler and Itakura—Saito divergences are special cases, but we also supply a dedicated method for the Rician cost function. Additionally, any twice-differentiable entry-wise cost function is supported. We demonstrate the improved effectiveness of the method over least-squares approaches by blindly separating a series of fMRI signals. Finally, we propose an improved algebraic algorithm to compute the CPD of a tensor in the least-squares sense. By recursively splitting the generalized eigenspace into successively smaller subspaces instead of determining all generalized eigenvalues at once, the decomposition is more accurate, yet can still be obtained efficiently. The algebraic solution can be used as an initialization for more costly optimization algorithms and significantly reduce the number of iterations and, hence, also the runtime of these methods.

Beknopte samenvatting

In de loop der jaren zijn er honderden algoritmen ontworpen om een matrix op te splitsen als een product van andere matrices. Deze matrixontbindingen kunnen worden gebruikt om gegevens te comprimeren met een minimaal verlies aan informatie of om zinvolle componenten uit de matrix te extraheren. Meer recent zijn tensorontbindingen zoals de canonieke polyadische ontbinding (CPO) en de lage-multilineaire-rangbenadering (LMLRB) ontworpen als hogereorde-veralgemeningen van deze matrixontbindingen die in bepaalde toepassingen tot betere resultaten leiden. Hoewel matrixmethodes een natuurlijke manier bieden om gegevens met twee variabelen te verwerken, herkennen ze vaak hogereordepatronen in de data niet. Tensoren kunnen daarentegen een willekeurig aantal modes hebben en kunnen de hogereordestructuur van de gegevens wel behouden, waardoor ze zeer nuttig zijn voor het analyseren van gegevens met meerdere variabelen. Ook zijn tensorontbindingen veel gemakkelijker uniek dan hun matrixtegenhangers. Omdat er minder (artificiële) beperkingen zoals orthogonaliteit of driehoeksstructuur aan de termen moeten worden opgelegd om hun uniciteit af te dwingen, kan dit leiden tot een ontbinding met componenten die beter overeenkomen met de realiteit.

Tensorontbindingsmethodes maken gewoonlijk twee aannames over de gegevens die door de gebruiker worden aangeleverd. Ten eerste gaan ze ervan uit dat alle gegevens tegelijk beschikbaar zijn en ten tweede beschouwen ze ruis op de gegevens als additief en Gaussiaans. In veel toepassingen zijn deze aannames gerechtvaardigd en kan men de tensor ontbinden met behulp van standaardmethodes die de kleinstekwadratenafstand tussen de tensor en het lagerangmodel minimaliseren. Naarmate tensormethodes echter op steeds meer reële problemen worden toegepast, neemt ook het aantal gevallen toe waarin deze aannames duidelijk worden geschonden. Elke *real-time* toepassing is bijvoorbeeld in strijd met de veronderstelling dat alle gegevens van bij de start beschikbaar zijn. In dergelijke toepassingen, zoals procescontrole of patiëntmonitoring, komen nieuwe gegevens met bepaalde tijdsintervallen binnen. Deze gegevens moeten in het tensormodel worden opgenomen zonder de volledige ontbinding vanaf nul te herberekenen, aangezien het gebruik van dergelijke methodes niet tijds- of geheugenefficiënt is. Ook is het niet moeilijk om toepassingen te vinden waarbij niet wordt voldaan aan de aanname van additieve Gaussiaanse ruis. Audiogegevens worden bijvoorbeeld over het algemeen gemodelleerd met behulp van een niet-Gaussiaanse ruisverdeling, terwijl men bij fotografische beelden

met zwakke belichting uitgaat van Poissonverdeelde pixelintensiteiten. Wanneer de reële en imaginaire delen van een signaal onafhankelijk worden verzameld, zoals typisch het geval is bij MRI-beeldvorming, is de ruisverdeling van de signaalmagnitude Riciaans in plaats van Gaussiaans. Voor deze toepassingen leidt het gebruik van de kleinstekwadratenafstand als kostfunctie tot suboptimale ontbindingen, terwijl een geschikte keuze van alternatieve kostfunctie een model kan opleveren dat beter overeenkomt met de onderliggende fysische componenten.

In dit proefschrift stellen we een reeks methodes voor die nog steeds efficiënt een accurate tensorontbinding kunnen berekenen wanneer aan één van de bovengenoemde aannames niet wordt voldaan. Ten eerste stellen we voor zowel de CPO als de LMLRB updatingsmethodes voor. Deze methodes starten vanuit een bestaande tensorontbinding en werken deze ontbinding efficiënt bij wanneer er nieuwe gegevens binnenkomen. Door gebruik te maken van de multilineaire structuur van de tensormodellen, zijn deze methodes zowel efficiënt als accuraat voor het volgen van lagerangmodellen van de gegevens. Indien gewenst, kan men zelfs een gewogen kleinstekwadratenmethode gebruiken om bepaalde tensorelementen zwaarder te wegen dan andere, bijvoorbeeld wanneer de kwaliteit van de verzamelde data varieert van sensor tot sensor. De prestaties van de CPO-updatingsmethode worden gedemonstreerd voor het monitoren van de hemodynamische koppeling in de hersenen bij pasgeborenen. Een tweede bijdrage is dat we de bestaande kleinstekwadraten-CPO-algoritmen uitbreiden naar andere kostfuncties. Dit maakt het mogelijk om voor bepaalde toepassingen een meer gepaste kostfunctie te gebruiken en om bijgevolg betere modellen te verkrijgen dan wanneer de standaard kleinstekwadratenkostfunctie wordt gebruikt. We richten ons in het bijzonder op de klasse van β-divergentiekostfuncties, waarvan de kleinstekwadratenafstand en de Kullback–Leibler- en Itakura–Saitodivergenties speciale gevallen zijn. We voorzien echter ook een gespecialiseerde methode voor de Riciaanse kostfunctie en in principe wordt zelfs elke tweemaal afleidbare kostfunctie ondersteund. We demonstreren de verbeterde prestaties van de gespecialiseerde methodes ten opzichte van ontbindingen die de kleinstekwadratenafstand minimaliseren door een reeks fMRI-signalen blindelings van elkaar te scheiden uit een opgemeten gemengd signaal. Ten slotte stellen we een verbeterd algebraïsch algoritme voor om de CPO van een tensor te berekenen met de kleinstekwadratenafstand als kostfunctie. Door de veralgemeende eigenruimte van de tensor voorzichtig recursief op te splitsen in steeds kleinere deelruimten in plaats van alle veralgemeende eigenwaarden in één keer te bepalen, is de ontbinding nauwkeuriger, maar toch efficiënt te verkrijgen. De algebraïsche oplossing kan worden gebruikt als een initialisatie voor duurdere optimalisatiealgoritmen en kan bijgevolg het aantal iteraties en dus ook de looptijd van deze methodes aanzienlijk verminderen.

Contents

Abstract v

Beknopte samenvatting vii

Contents ix

List of Figures xv

List of Tables xxi

List of Algorithms xxiii

List of Abbreviations xxv

List of Symbols xxx

I Tensors 1

1 Introduction 3
1.1 From tables to matrices . . . 4
1.2 From Flatland to Space . . . 5
1.3 Velocity and variety . . . 8
1.4 Research objectives and thesis overview . . . 9
1.4.1 Part 1: Tensors . . . 10
1.4.2 Part 2: Velocity . . . 11
1.4.3 Part 3: Variety . . . 13
1.4.4 Conclusion and appendices . . . 15

2 Tensor decompositions 17
2.1 Introduction . . . 17
2.1.1 Notation and terminology . . . 18
2.2 Decompositions . . . 21
2.2.1 Canonical polyadic decomposition (CPD) . . . 22
2.2.2 Low multilinear rank approximation (LMLRA) . . . 24
2.2.3 Other tensor decompositions . . . 25
2.3 Computing tensor decompositions . . . 26

2.3.1 Algebraic algorithms . . . 27
2.3.2 Optimization-based algorithms . . . 32

II Velocity 41

3 Nonlinear least squares updating of the canonical polyadic decomposition 43
3.1 Introduction . . . 44
3.2 Notation and definitions . . . 45
3.3 NLS updating . . . 46
3.3.1 Objective function . . . 47
3.3.2 Gradient and Gramian . . . 48
3.3.3 Windowing and dynamic rank . . . 49
3.3.4 Complexity analysis . . . 51
3.4 Experiments . . . 52
3.5 Conclusion . . . 53
3.A Other updating schemes . . . 55
3.A.1 Objective function . . . 56
3.A.2 Gradient and Gramian . . . 57
3.A.3 Fill-in experiment . . . 58
3.A.4 Conclusion . . . 59

4 CPD updating using low-rank weights 61
4.1 Introduction . . . 62
4.1.1 Notation, definitions and identities . . . 63
4.2 NLS CPD updating using low-rank weights . . . 64
4.2.1 Objective function . . . 65
4.2.2 Gradient . . . 66
4.2.3 Gramian-vector product and preconditioner . . . 67
4.2.4 Initialization and windowing . . . 68
4.2.5 Complexity analysis . . . 68
4.3 Direction of arrival estimation (DOA) . . . 69
4.4 Conclusion and further work . . . 72

5 Tracking the low multilinear rank approximation 73
5.1 Introduction . . . 73
5.1.1 Notation and terminology . . . 76
5.2 Multilinear UTV decomposition . . . 78
5.3 MLUTV Updating . . . 79
5.3.1 Subspace tracking of the non-evolving modes . . . 82
5.3.2 Subspace tracking of the evolving mode . . . 84
5.3.3 Algorithm complexity . . . 88

5.3.4 Choosing the threshold values gamma_i . . . 91
5.4 Error bounds for the TMLUTVD . . . 91
5.4.1 Frobenius norm . . . 91
5.4.2 Spectral norm . . . 94
5.5 Strategies to reduce the complexity . . . 95
5.5.1 Exploiting small window lengths . . . 96
5.5.2 Compressing the new slice . . . 97
5.5.3 Sequential truncation . . . 98
5.6 Experiments . . . 101
5.6.1 Updating with abrupt subspace changes . . . 101
5.6.2 Direction-of-arrival estimation . . . 104
5.7 Conclusion . . . 105
5.A Proof of approximation bound . . . 107
5.B Update of the core tensor . . . 108

III Variety 111

6 A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost 113
6.1 Introduction . . . 114
6.1.1 Notation . . . 115
6.2 General cost functions for the CPD . . . 116
6.2.1 Gradient . . . 117
6.2.2 Hessian approximation . . . 118
6.2.3 Computing JTZJ . . . 119
6.3 Beta-divergences . . . 121
6.3.1 Indefiniteness of the Hessian approximation . . . 124
6.4 Experiments . . . 127
6.4.1 Optimal value of beta . . . 128
6.4.2 Handling indefiniteness . . . 128
6.4.3 Other beta-divergence CPD algorithms . . . 130
6.4.4 GGN compared to GN . . . 132
6.4.5 Hyperspectral image compression . . . 135
6.5 Conclusion . . . 137
6.A Proofs of Sec. 6.2.3 . . . 137

7 Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions 141
7.1 Introduction . . . 142
7.1.1 Notation . . . 145
7.2 CPD of tensors with general cost functions . . . 147
7.2.1 GGN for non-LS cost functions . . . 148

7.2.2 Computing JTDJx efficiently . . . 151
7.2.3 Approximating D . . . 153
7.2.4 Preconditioners . . . 154
7.3 Alternative cost functions . . . 157
7.4 Experiments . . . 159
7.4.1 Preconditioning . . . 160
7.4.2 Approximating D . . . 161
7.4.3 Other beta-divergence CPD algorithms . . . 163
7.4.4 Large-scale tensors . . . 168
7.4.5 Separating fMRI sources . . . 172
7.5 Conclusion . . . 175

8 Rank-one tensor approximation with beta-divergence cost functions 176
8.1 Introduction . . . 177
8.1.1 Notation . . . 178
8.2 beta-divergence rank-R approximation . . . 178
8.3 Rank-1 approximations . . . 180
8.3.1 Cost function . . . 181
8.3.2 Gradient . . . 182
8.3.3 Hessian approximation . . . 182
8.4 Experiments . . . 184
8.4.1 Experiment 1 . . . 184
8.4.2 Experiment 2 . . . 185
8.5 Conclusion . . . 186

9 Tensor-based blind fMRI source separation without the Gaussian noise assumption 187
9.1 Introduction . . . 188
9.1.1 Notation . . . 189
9.2 Noise Model . . . 189
9.3 CPD with beta-divergence . . . 190
9.4 Results . . . 192
9.5 Conclusion . . . 196

10 A recursive eigenspace computation for the canonical polyadic decomposition 197
10.1 Introduction . . . 197
10.1.1 Notation and definitions . . . 199
10.1.2 Joint generalized eigenvalues and eigenvectors . . . 201
10.1.3 Guide to the reader . . . 204
10.2 The GESD algorithm . . . 204
10.2.1 Rank-1 terms . . . 206
10.2.2 Choice of slices . . . 206

10.2.3 Handling of complex eigenvectors when working over $\mathbb{R}$ 207
10.3 A deterministic bound for distinct eigenvalue clusters 210
10.3.1 Proof of Theorem 10.3.1 216
10.3.2 Distinct eigenvalue clusters for subpencils of a tensor . . 218
10.3.3 Computation of the bounds appearing in Theorem 10.3.1 and Corollary 10.3.1 219
10.4 GESD vs. GEVD accuracy and timing 220
10.4.1 Tensor dimension exceeds tensor rank 221
10.4.2 Factor matrices with correlated columns 222
10.4.3 Comparison with optimization 223
10.4.4 GESD and GEVD vs. noiseless problems 228
10.5 Accuracy of GESD versus deterministic bound 230
10.6 Conclusion 231

11 Conclusion **234**
11.1 Contributions 234
11.2 Perspectives 242

A Monitoring of Brain Hemodynamics Coupling in Neonates using Updated Tensor Decompositions **245**
A.1 Introduction 246
A.2 Methods 247
A.2.1 Data 247
A.2.2 Preprocessing 247
A.2.3 Algorithm 248
A.3 Results 249
A.4 Discussion 251
A.5 Conclusion 252

B Supplementary materials: Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions **253**

C Tensorlab 4.0 – A preview **257**
C.1 Tensorlab 3.0 257
C.2 New features in Tensorlab 4.0 258
C.2.1 Beta-divergence cost functions for the CPD 258
C.2.2 Low-rank WLS cost functions for the CPD 258
C.2.3 CPD updating 259
C.2.4 LS-CPD 259
C.3 Conclusion 260

Bibliography **261**

List of Publications **283**

List of Figures

1.1 Vector and matrices are first- and second-order, respectively, while tensors are of order three or higher. 7
1.2 The CPD of a tensor decomposes a tensor as a sum of R rank-1 terms, which allows for the extraction of meaningful components from higher-order data. 7
1.3 Visual representation of an LMLRA of a third-order tensor. The tensor is represented as a core tensor with smaller dimensions, holding the interactions between the different modes, multiplied in each mode with a factor matrix. . . . 8
1.4 Schematic overview of the interconnections between chapters. Chapters that focus on applications are shown in a darker shade of blue. 16

2.1 A CPD represents the tensor as a sum of R rank-1 terms, leading to a compact decomposition that is unique under mild conditions. 22
2.2 LMLRA of a third-order tensor. The tensor is represented as a core tensor with smaller dimensions, multiplied in each mode with a factor matrix. 24
2.3 For more difficult problems, ALS methods can take many iteration steps with limited progress before eventually converging to the solution, while NLS methods converge in much fewer iterations. 34
2.4 The Gauss–Newton Hessian approximation for the CPD optimization problem has a grid-like structure. 37
2.5 Using the preconditioned CG algorithm, the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ can be solved by repeatedly performing multiplications of the form $\mathbf{y} = \hat{\mathbf{H}}\mathbf{x}$, wherein both $\mathbf{x}$ and $\mathbf{y}$ can be partitioned according to the factor matrix that each part corresponds to. 38

3.1 Example of the tensor updating procedure. 47
3.2 Weighted mean errors of the new updating method, ALS and NLS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods for $R = 6$ and $\mathrm{SNR} = 50\,\mathrm{dB}$ on a decibel scale. 54
3.3 The tensor $\mathcal{T}^{(\mathrm{up})}$ is decomposed as $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] + \mathcal{S}^{+}$. 56

3.4 The relative factor matrix error of the updating method gradually decreases as more entries are sampled. 60
3.5 The updating method is significantly faster than both `cpdi_nls` and `cpd_nls` and its computation time remains constant over the updates. 60

4.1 WLS updating using low-rank weights enables efficient CPD updates by exploiting the low-rank structure of *both* the weight and data tensor. 65
4.2 A low-rank weight tensor can improve DOA estimation by giving lower importance to values obtained by broken sensors. 71

5.1 LMLRA of a third-order tensor. 76
5.2 Update of the LMLRA of a third-order tensor. 76
5.3 The MLUTVD can be truncated to obtain an LMLRA of a tensor by only using the columns of the factor matrices that correspond to the principal mode-n subspaces and the corresponding entries of the core tensor, marked in red. . . 80
5.4 Low-rank approximation of a matrix and a tensor. 81
5.5 Update of the LMLRA for mode 1. The new columns of the mode-1 unfolding are included in the subspace $\mathbf{U}_1^{(n)}$. 82
5.6 Appending a new slice to the tensor in the evolving mode 3 (right) corresponds to adding a new row to the mode-3 unfolding. 85
5.7 The squared Frobenius norm of the TMLUTVD error is upper-bounded by the sum of the truncation errors in the different modes. 93
5.8 The updating method tracks an LMLRA of a tensor with high accuracy, even when abrupt changes in the subspaces occur. 103
5.9 By compressing the new slices (Section 5.5.2) and using sequential truncation (Section 5.5.3), the method can be sped up significantly, without negatively impacting the accuracy. 103
5.10 The TMLUTVD updating method tracks the DOAs of several sources with high accuracy and correctly estimates the number of sources. 106
5.11 With compression (Section 5.5.2) and sequential truncation (Section 5.5.3), the TMLUTVD tracking method becomes faster than the batch TMLSVD algorithm, with minimal loss in accuracy for the estimation of the subspaces. 106

6.1 Plot of $d_\beta(1, y)$ as a function of y for different values of β. . 122
6.2 The WLS cost function gives a better local approximation to $d_{0.5}(3, y)$ than the LS cost function. 126

6.3 The errors for tensors perturbed by Gamma noise are about 29.4% larger when using $\beta = 2$ compared to $\beta = 0$, while for Gaussian noise, using $\beta = 2$ is optimal. 129
6.4 The cost function with $\beta = 4$ performs better for *additive Gaussian noise* than the cost function with $\beta = 0$. 129
6.5 For *multiplicative Gamma noise*, the cost function with $\beta = 0$ performs better. 130
6.6 The proposed GGN method requires considerably fewer iterations than the other two methods. 133
6.7 The GGN method converges quadratically (until machine precision is reached) close to the solution, while the other methods converge linearly. 134
6.8 The GGN method (dashed) requires about twice as much time to compute the Hessian approximation as the GN method (solid). Second, compared to LS, the β-divergence cost function and gradient evaluations are much more costly. 134
6.9 Spectral band 120 of a (80×80) aerial image perturbed by Poisson noise with an SNR of 10 dB. 136

7.1 When a tensor is perturbed by multiplicative Gamma noise, fitting a CPD with the IS-divergence cost function consistently leads to a better solution than when the LS cost function is used. 143
7.2 Thanks to the structure of each $\mathbf{G}^{(n,n)}$-block, applying the block-Jacobi preconditioner involves solving I independent linear systems. 157
7.3 When $\mathcal{D}$ is compressed, the computation time for an evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ (without the compression time) remains relatively constant for increasing I, while it increases exponentially without compression. 164
7.4 The inexact GGN method with compression is faster than the exact GGN method for sufficiently large tensors. 168
7.5 The inexact GGN method with compression scales much better than the exact GGN method and the L-BFGS and Adam methods for increasing tensor dimensions, while the solution accuracy is at least as good. 169
7.6 The inexact GGN method with compression scales much better than the exact GGN method for increasing rank R. Still, for $R \geq 16$, the first-order method L-BFGS is more efficient. 170
7.7 Spatial (left) and temporal (right) modes of eight simulated fMRI sources. 173

7.8 Unmixed fMRI sources when a CPD with a standard LS cost function is used. . . . 174
7.9 Unmixed fMRI sources when a CPD with a Rician-dedicated cost function is used. . . . 175

8.1 The rank-1 method offers a large speedup, except for the case $\beta = 2$ (LS cost function). . . . 185

9.1 Magnitude spatial maps and time courses of the complex-valued sources used. . . . 191
9.2 First simulation with the 8 sources used in [25]. . . . 194
9.3 Second simulation with the 8 sources used in [25] and different noise variance in the real and imaginary parts of the complex-valued source signals. . . . 194
9.4 Third simulation with 9 sources, where Source 9 with high overlap is included in the scenario of Figure 9.1. . . . 195
9.5 Histograms of the observed values at one spatial point. Left: real and imaginary noise variances are equal. Right: the noise variance of the imaginary part is five times that of the real part. . . . 195

10.1 For normally distributed factor matrices, the GESD and GEVD methods have similar performance with respect to the accuracy of the computed factor matrices. . . . 225
10.2 For factor matrices with entries sampled from the uniform distribution, the GESD method obtains a significantly higher accuracy for the estimated factor matrices than the GEVD method. . . . 225
10.3 For factor matrices with correlated columns, the GESD method performs significantly better than the GEVD method with respect to the factor accuracy. . . . 226
10.4 The relative time difference between the GESD and GEVD methods remains about the same as tensor rank increases. . . 226
10.5 Initializing an optimization algorithm with the GESD or GEVD solution is faster than starting from a random initialization. . . . 227
10.6 For factor matrices with correlated columns, the GESD solution is more accurate than the GEVD solution. . . . 227
10.7 For sufficiently difficult problems, initialization with GESD leads to both a more accurate solution and lower computation time of the optimization algorithm compared to initialization with GEVD. . . . 228

10.8 The angle between $\tilde{\mathbf{E}}_1$ and $\tilde{\mathbf{E}}_2$ stays close to $\pi/2$ until the bound is crossed, while the angle between $\tilde{\mathbf{Z}}_i$ and columns of $\mathbf{B}^{-\mathrm{T}}$ stays close to zero until this bound is crossed. Trying more unitary matrices shifts the bound sightly to the left. . 232

A.1 Concomitant measurements from a characteristic neonate after propofol administration. 248
A.2 Third mode of the decomposed tensor represented as the median value, and the 25 and 75 percentile. 250
A.3 Third mode of the decomposed tensor for the three different groups. 251

List of Tables

3.1 Medians of the CPU-time (in ms) for a single update using the new updating method, NLS and ALS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods. . . 53
3.2 Weighted mean errors for the new updating method, NLS and ALS batch algorithms and PARAFAC-SDT and PARAFAC-RLST. Medians over the fifty updating steps. 54

4.1 Per-iteration complexity of the WLS CPD updating algorithm for an $(I \times I \times I)$-tensor. 69
4.2 The median absolute errors of the azimuths and elevations are much smaller if a low-rank weight tensor is applied. 70
4.3 The updated WLS-method requires less CPU-time (s) than its batch counterpart. 72

5.1 Memory and computational complexity for the batch TMLSVD method and the proposed updating method. 90
5.2 With the strategies from Section 5.5, TMLUTVD updating requires less storage and is faster than the batch algorithm. . . . 101

6.1 The LS and WLS initializations enable faster CPD computations, with WLS performing better for small β. For random initializations, shifting does not influence the computation time much, while it does assure convergence. 131
6.2 When starting from an LS or WLS initialization, the number of remaining (costly) GGN iterations until convergence is significantly lower than for random initialization. Shifting does not change the number of iterations when starting from a random initialization, while it does assure that the method converges. . . 131
6.3 Comparison of the PSNR (left) and the UIQI (right) for a hyperspectral image approximated by a CPD with either a LS or KL-divergence cost function. 137

7.1 Preconditioners can reduce the number of CG or MINRES steps in the GGN method. 160
7.2 Compressing $\mathcal{D}$ decreases the computation time and number of iterations for sufficiently large tensors. 165

7.3 Proper compression of $\mathcal{D}$ decreases the computation time for tensors of large enough dimensions. 166
7.4 Using the RBS framework, the GGN method can be applied to obtain CPDs of large tensors with non-LS cost functions. 171
7.5 The recovered spatial sources are slightly more accurate with the Rician-dedicated cost function compared to the LS cost function. 174

8.1 The NLS algorithm is slightly faster than the MU algorithm and requires less storage space. 186

List of Algorithms

2.1 CPD via GEVD 30
2.2 Truncated MLSVD 31
2.3 Sequentially truncated MLSVD 31

3.1 NLS updating for CPD 50

4.1 WLS CPD updating 65

5.1 SURV updating of the non-evolving modes 1 and 2 85
5.2 RRURV tracking for the evolving mode 89

10.1 GESD recursive step 208
10.2 Splitting eigenspaces (over $\mathbb{R}$)[1] 209

List of Abbreviations

ALS	alternating least squares
BCD	block coordinate descent
BOLD	blood oxygenation level dependent
BSS	blind source separation
BFGS	Broyden–Fletcher–Goldfarb–Shanno
BTD	block term decomposition
CANDELINC	canonical decomposition with linear constraints
CG	conjugate gradients
CPD	canonical polyadic decomposition
CPDI	CPD for incomplete tensors
CPU	central processing unit
CR	Cramér–Rao
CRB	CR bound
CRIB	CR induced bound
CRLB	CR lower bound
DOA	direction of arrival
EEG	electroencephalography
EVD	eigenvalue decomposition
FAPI	fast approximated power iteration

fMRI	functional MRI
FT	Fourier transform
GCPD	generalized CPD
GESD	generalized eigenspace decomposition
GEVD	generalized EVD
GGN	generalized Gauss–Newton
GN	Gauss–Newton
HOSVD	higher-order SVD
HR	heart rate
IFT	inverse Fourier transform
ICA	independent component analysis
INSURE	intubation, surfactant and extubation
IS	Itakura–Saito
JGE	joint generalized eigen
KL	Kullback–Leibler
LBFGS	limited memory BFGS
LL1	decomposition in multilinear rank-$(L_r, L_r, 1)$ terms
LM	Levenberg–Marquardt
LMLRA	low multilinear rank approximation
LS	least squares
LS-CPD	linear system with CPD constrained solution
LVA	latent variable analysis
MABP	mean arterial blood pressure

MINRES	minimum residual method
MLE	maximum likelihood estimator
MLSVD	multilinear SVD
MLUTVD	multilinear UTV decomposition
MR	magnetic resonance
MRI	MR imaging
MTKRPROD	matricized tensor times Khatri–Rao product
MTKRONPROD	matricized tensor times Kronecker product
MU	multiplicative update
NIRS	near-infrared spectroscopy
NLS	nonlinear LS
NMF	nonnegative matrix factorization
PARAFAC	parallel factor analysis
PAST	projection approximation subspace tracking
PC	preconditioner
PCG	preconditioned conjugate gradients
PK	pharmacokinetics
PMA	postmenstrual age
PSD	positive semidefinite
PSNR	peak SNR
QN	quasi-Newton
RAM	random access memory
RBF	radial basis function
RBS	randomized block sampling
RMS	root mean square

RMSE	RMS error
RRULV	rank-revealing ULV
RRURV	rank-revealing URV
SDF	structured data fusion
SNR	signal-to-noise ratio
SSI	structural similarity index
STMLSVD	sequentially truncated MLSVD
STMLUTVD	sequentially truncated MLUTVD
SURV	signed URV
SVD	singular value decomposition
THOSVD	truncated HOSVD
TMLSVD	truncated MLSVD
TMLUTVD	truncated MLUTVD
TTD	tensor train decomposition
UIQI	universal image quality index
URA	uniform rectangular array
WLS	weighted LS

List of Symbols

Unary

a	Scalar
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
$\mathcal{A}$	Tensor
$.^{(n)}$	Nth entry in a set
$\mathbf{T}_{[i_{m_1},\ldots,i_{m_k};i_{m_{k+1}},i_{m_N}]}$	Matricization of a tensor with row modes $i_{m_1},\ldots,i_{m_k}$ and column modes $i_{m_{k+1}},\ldots,i_{m_N}$
$\mathbf{T}_{(n)}$	Mode n unfolding of $\mathcal{T}$, equal to $\mathbf{T}_{[n;i_N,\ldots,i_{n+1},i_{n-1},\ldots,i_1]}$
$\operatorname{vec}(\mathcal{T})$	Vectorization of $\mathcal{T}$, equal to $\mathbf{T}_{[;i_N,\ldots,i_1]}$
$\operatorname{unvec}(\mathbf{t})$	Inverse operation of the vectorization
$\mathbb{N}$	Field of natural numbers
$\mathbb{R}$	Field of real numbers
$\mathbb{C}$	Field of complex numbers
$\mathbb{K}$	Field of real or complex numbers
$\lvert\cdot\rvert$	Absolute value or modulus
$\lVert\cdot\rVert$	Norm
$.^{\mathrm{T}}$	Transpose
$.^{\mathrm{H}}$	Hermitian transpose
$.^{-1}$	Inverse
$.^{\dagger}$	Moore–Penrose pseudoinverse
$\bar{.}$	Complex conjugate
$.^{\bullet a}$	Entry-wise power
$\mathbf{I}_n$ or $\mathbf{I}$	$n \times n$-identity matrix
$\mathbf{1}_{m\times n}$ or $\mathbf{1}$	Matrix consisting of only ones
$\mathbf{0}_{m\times n}$ or $\mathbf{0}$	Matrix consisting of only zeros
$\Pi^{(n)}$	Permutation matrix that brings the nth mode to the first position
d	Derivative
∂	Partial derivative
$\mathcal{O}$	Big-O notation for complexities
I_k	Modified kth-order Bessel function of the first kind
σ_n	nth largest singular value of a matrix
λ_n	Eigenvalue of a matrix
$\emptyset$	Empty set

$\mathrm{BF}_{\epsilon}^{\mathbb{K},0}$	Bauer-Fike ball of radius ϵ over the field $\mathbb{K}$
sgn	Sign function

Binary

$\cdot$	Standard matrix product
$\otimes$	Kronecker product
$\odot$	Column-wise Khatri–Rao product
$\odot^{\mathrm{T}}$	Row-wise Khatri–Rao product
$*$	Hadamard product
$\langle \cdot, \cdot \rangle$	Inner product
$\otimes$	Outer product of vectors
$\cdot_n$	Mode-n tensor-matrix product
$:=$ or $=:$	Left or right definition
$\chi(\cdot, \cdot)$	Chordal metric between two vectors

Other

$[\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$	Canonical polyadic decomposition with N factors $\mathbf{U}^{(n)}$
$[\![\mathcal{S}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$	Low multilinear rank approximation with N factors $\mathbf{U}^{(n)}$ and core tensor $\mathcal{S}$
$\mathrm{diag}(\mathbf{X})$	Diagonal of $\mathbf{X}$
$\mathrm{Diag}(\mathbf{x})$	Square matrix consisting of zeros except for $\mathbf{x}$ on its diagonal
$[\mathbf{A}, \mathbf{B}]$	Horizontal concatenation of matrices
$[\mathbf{A}; \mathbf{B}]$	Vertical concatenation of matrices
min	Minimum

Part I: Tensors

1 Introduction

The last decades have seen an explosion of the amount of data that is collected, stored and analyzed. While household appliances used to have a few buttons to run predefined programs, they are now often 'smart' and collect and analyze data about usage patterns themselves. Marketing used to target specific demographics through magazines and posters, but each individual now receives ads that are personalized with the use of detailed user data. The enormous economical value of data can be seen through the fact that for an increasing number of companies, selling their users' data is their only source of income. As the amount of data that is collected continues to increase, the demand for data analysis tools has grown at the same rate. Indeed, by extracting and storing only the useful parts of the data, the same information can be stored and accessed for a much lower cost.

With increasing volume, the velocity of the data collection has quickly grown as well. For example, social media platforms process millions of user requests per second and for self-driving cars, data analysis has to be performed in milliseconds to avoid crashes at high speeds. The increased speed of applications poses tremendous challenges for data scientists. Instead of looking for the best solution, one now needs to find the optimal solution given the available time, which can require significantly different techniques. There is also no time to manually check the computed results, so one needs a deep understanding of the applied techniques to ensure trustworthiness and safety.

Data variety is a third component that has seen large increases over the past decades. Data used to consist of tabulations of incomes and expenses or simple physical measurements, but are now collected in almost any field and for almost any application. Each application generates different types of data and there is no method that can optimally handle all of them. On the contrary, in fact, as highly specialized application-specific tools are highly sought after to create an edge compared to competitors.

Data have become a huge part of our everyday lives and the demand for new data analysis tools shows no signs of slowing down. The ever-increasing data volume has spurred the development of many interesting new methods to tackle huge data sets with minimal cost. In the same spirit, one of the goals of this thesis is to try and address some of the challenges that are caused by the increasing velocity and variety of data.

1.1 From tables to matrices

Tables offer a natural way to store information. For thousands of years, people have been drawing tables setting out one variable in function of another and have looked for patterns that explain the underlying numbers. Some of the earliest tables were, for instance, drawn on the walls of caves, on the ground, on clay tablets... Over time, these drawings have been replaced by more modern variants: first by tables drawn with ink and later by tables consisting of computer pixels. Still, the main concept has remained the same. In 1850, J. J. Sylvester introduced the term 'matrix' and since then, countless days, years and lives have been dedicated to the study of matrices, which is still ongoing. Because of their simplicity and power, matrices are one of the cornerstones of the field of data science. The data are usually stored in the form of a (large) matrix and matrix methods are applied to extract the relevant information. As an example, a matrix $\mathbf{X}$ holding some mixture of unknown source signals is often decomposed into a low-rank model of the form

$$\mathbf{X} = \mathbf{MS}, \tag{1.1}$$

wherein $\mathbf{S}$ is a wide matrix (more columns than rows), holding a limited number of source signals and $\mathbf{M}$ is a tall matrix (more rows than columns) that holds the specific mixing variables. Computing the matrices $\mathbf{M}$ and $\mathbf{S}$ from $\mathbf{X}$ is an example of a matrix decomposition that has, for instance, found applications in signal processing [37, 39, 45].

Matrix decompositions can be powerful tools to explain patterns in data, but they have two important drawbacks. The first is that they require additional constraints to be unique. In (1.1), one can choose any invertible matrix $\mathbf{W}$ to obtain a different low-rank decomposition

$$\mathbf{X} = (\mathbf{MW})(\mathbf{W}^{-1}\mathbf{S}) = \tilde{\mathbf{M}}\tilde{\mathbf{S}},$$

that is equally valid. Using constraints such as orthogonality, triangle structure or nonnegativity, one can enforce uniqueness and obtain many of the most-used matrix decompositions. Among many others, the SVD, the QR decomposition and the NMF can be constructed in this way [87]: the SVD factorizes a matrix $\mathbf{X}$ as $\mathbf{U\Sigma V}^{\mathrm{T}}$ with $\mathbf{U}$ and $\mathbf{V}$ orthogonal and $\mathbf{\Sigma}$ diagonal and sorted in descending order, while the QR decomposition decomposes $\mathbf{X}$ as $\mathbf{QR}$ with $\mathbf{Q}$ orthogonal and $\mathbf{R}$ upper-triangular and the NMF factorizes $\mathbf{X}$ as $\mathbf{AB}$ with both $\mathbf{A}$ and $\mathbf{B}$ nonnegative. These constrained matrix decompositions have been proven effective for a wide range of use cases. However, for many other applications, these constraints do not correspond to the reality and can lead to incorrect interpretations. For instance, imposing orthogonality or independence

of components [39, 49] when the sources do not have these properties, makes it impossible to recover the original signals.

The second major drawback of matrix methods is that they cannot exploit all structure that exists in higher-order data. Even if the data depend on three or more different variables, storing the information in a matrix offers only two modes to represent it. This generally leads to a loss of important higher-order interactions that cannot be recovered through matrix decompositions. For example, consider a table tracking the mini golf scores of a group of five friends that play a different course each week: after their game, they collect the scores of all players on all eighteen holes of the course in a (*player* × *hole*)-table and add the weekly table to their spreadsheet, below the previous weeks' scores. After a year, they have a large table (260 rows by 18 columns) and would like to analyze each player's performance over the year on the first hole (one of them boasts about the superior warm-up method he adopted over the summer). Matrix methods will struggle to perform this analysis, as the row dimension is a mix of different players and courses: after one row of the warmed up player's scores are four rows of scores for the other players. Reordering the table does not help either, as this results in the holes and/or the courses no longer matching up. No matter how you organize the scores in a matrix, some part of the information will be lost. This drawback makes the use of matrix methods suboptimal to represent multiway data, data that depend on three or more variables. Fortunately, there exists a way to store and decompose data without losing its higher-order structure and without the need to impose artificial constraints: through the use of tensors and tensor decompositions.

1.2 From Flatland to Space

> *I call our world Flatland, not because we call it so, but to make its nature clearer to you, my happy readers, who are privileged to live in Space.*
>
> Edwin A. Abbott, Flatland: A Romance of Many Dimensions [1]

If you were to live in Flatland, the two-way world from the 1884 novel with the same name, every object would look either as a point or a line. Even when actually looking at a sphere or a cube, inhabitants of Flatland still see nothing but a line, as they only see the part of the object that happens to intersect with their world. These people have never heard of Spaceland, the three-way world that encompasses theirs and wherein spheres, cubes and other three-way objects live. Instead, they try to explain everything through their limited view wherein everything is 'linear'.

The relationship between Flatland and Spaceland can be used as a metaphor for the relationship between matrix and tensor methods. For many years, engineers and data scientists have lived in 'Matrixland', explaining everything through two-way models and linear algebra. Only in the last couple of decades, have some of them been convinced by mathematicians to wander into 'Tensorland', where N-way models and multilinear algebra can be found and applied to concrete applications. After discovering that they had been looking at linearizations of a much richer reality for their whole lives, most of them took permanent residence in Tensorland, only making occasional trips to their old reality.

Tensors [35, 174] are higher-order extensions of vectors and matrices, see Figure 1.1, that provide a natural and convenient way to represent higher-order data without destroying its multilinear structure. These multi-indexed arrays can have any number of modes, even if visualizing them on paper or a computer screen is close to impossible for orders higher than three. Finding situations where tensors are useful is not hard, as many applications generate tensor data out of their own, for instance by collecting results while varying three or more parameters. As an example, consider the mini golf scores from Section 1.1: the yearly overview of scores forms a third-order (*player* $\times$ *hole* $\times$ *course*)-tensor. In contrast to matrix methods, tensor methods enable the kind of detailed warm-up analysis that the players crave. Even if the data are not higher-order by nature, they can still be *tensorized* [52] to leverage the useful properties of tensor methods and obtain better results.

Although tensors offer a natural way to store multiway data, they have one major drawback: for an Nth-order tensor with I entries in every mode, the number of entries grows as $\mathcal{O}(I^N)$ as the dimensions of the modes are increased. This *curse of dimensionality* [63] means that tensors can easily become impractically large, and storage and computations can put a strain on the available resources. Fortunately, the information in the data can often be represented by much fewer parameters than there are entries in the tensor. Using a suitable tensor decomposition, one can recover such a representation that is much more efficient to store and manipulate than the full tensor.

Like matrix decompositions, tensor decompositions provide a way to extract meaningful components from tensors or to compress large tensors while retaining the most important features of the data. The canonical polyadic decomposition [102] (CPD) is one example of a tensor decomposition that is widely used in practice. It decomposes a tensor as a sum of R rank-1 terms (with R minimal). The CPD of a third-order tensor $\mathcal{T}$ has the following form:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r,$$



Figure 1.1: Vector and matrices are first- and second-order, respectively, while tensors are of order three or higher.



Figure 1.2: The CPD of a tensor decomposes a tensor as a sum of R rank-1 terms, which allows for the extraction of meaningful components from higher-order data.

wherein the symbol $\otimes$ denotes the outer product of vectors. In contrast to the SVD, constraints are not needed to make the CPD unique: up to scaling within the individual rank-1 terms and permutation of the terms themselves, the decomposition is unique under mild conditions on the dimensions and number of rank-1 terms R [57, 58, 123]. Another very important property of the CPD is that it represents the $\mathcal{O}(I^N)$ entries of an Nth-order tensor with $\mathcal{O}(NR)$ parameters, thus turning an exponential dependence on the order N into a linear dependence. Consequently, for large orders N, using the CPD of the tensor instead of the tensor itself can make extremely time- and memory-heavy computations almost trivially easy to handle. Of course, this is under the assumption that the data permits such a low-rank representation. For a lot of large-scale data, this assumption can confidently be made, but for other types, the exponential dependence on the order cannot be avoided.

A second type of tensor decomposition that has found numerous applications in practice is the low multilinear rank approximation (LMLRA) [208], as visualized in Figure 1.3. Unlike the CPD, this decomposition does not remove the exponential dependence on N, but the smaller core tensor compared to the full tensor can still offer tremendous reductions in storage and computational cost with minor loss of information. The LMLRA allows the interactions between the different modes to be represented in the core tensor. By restricting

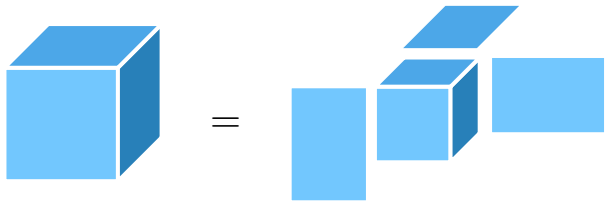

Figure 1.3: Visual representation of an LMLRA of a third-order tensor. The tensor is represented as a core tensor with smaller dimensions, holding the interactions between the different modes, multiplied in each mode with a factor matrix.

the factor matrices to be orthogonal and the core tensor to be 'all-orthogonal', one obtains an MLSVD [47].

Aside from the CPD and the LMLRA, other types of decompositions such as the block term decomposition (BTD) [44], the tensor-train decomposition (TTD) [151] or the hierarchical Tucker decomposition [89] have been developed and applied in applications. An overview of tensor decompositions and algebraic and optimization-based algorithms to compute the CPD and LMLRA are given in Chapter 2.

1.3 Velocity and variety

Because of the enormous number of entries that a tensor can have, the volume of the data has always been a very important consideration when designing tensor algorithms. Expertise from the field of scientific computing has been extensively applied to enable computations on huge data sets and many stochastic methods that sample only small parts of the data have been designed [42, 96, 112, 120, 154, 157, 175, 180, 222]. Only in recent years has the focus shifted to development of algorithms that can also handle the increasing velocity and variety of the data.

Streaming data that requires high-speed analysis has become ubiquitous since the wide-spread use of the internet and the explosion of real-time applications in surveillance, manufacturing, traffic, etc. Tensor methods should not only be able to decompose a tensor that is fully available at the start, but should be able to track changes and additions to the tensor as more data are collected. To tackle this issue, updating methods have been developed that include the new information in the tensor decomposition when new data arrives or old data becomes outdated [113, 136, 137, 139, 149, 179, 182, 199, 244]. These methods prioritize speed by limiting the amount of computations and data accesses,

at the cost of a lower accuracy of the resulting decomposition; a trade-off which is impossible to avoid when time is limited. In Part II of this thesis, we contribute to the quest for tensor updating methods by designing algorithms that can track the CPD and the LMLRA in real-time, while still providing a good approximation to the tensor.

Standard tensor decomposition methods minimize the least-squares (LS) distance between the tensor and the low-rank approximation and thus compute an approximation to the tensor under the implicit assumption that the tensor is perturbed by additive Gaussian noise. As tensors are applied in an increasing number of fields, however, more and more applications have been found where these standard methods do not provide great results. Usually, the main issue is that assumptions about the data and noise distribution are not satisfied for the application at hand. Count data, for instance, is modeled using the Poisson distribution [194], while the magnitude of fMRI signals follows a Rician distribution [125]. As using the LS distance to evaluate the approximation quality leads to less accurate results for these types of data than when a more suitable distance metric is used, recent work has focused on tailoring tensor methods to specific data distributions [34, 38, 74, 94, 101, 105, 240]. This leads to more involved, but significantly more flexible methods that can give more accurate decompositions for a wide variety of applications. In Part III, we develop such methods for the CPD.

1.4 Research objectives and thesis overview

In this thesis, we aim to design algorithms that enable the use of tensors for a wider range of applications. We focus in particular on real-time applications and applications for which using the standard LS cost function is suboptimal due to the data and/or noise distribution. We identify three main objectives:

- *Designing updating algorithms.* Recomputing a tensor decomposition whenever tensor entries change or new data is added, is prohibitively expensive in real-time applications. Our aim is to design algebraic and optimization-based algorithms that reuse the previously computed decomposition and efficiently include the new information, making them fast enough for real-time applications.
- *Designing distribution-specific algorithms.* Using the standard LS cost function in optimization algorithms can be suboptimal for applications with certain specific data and/or noise distributions. We aim to develop algorithms that support the use of more targeted cost functions and

consequently lead to more accurate decompositions. Additionally, we aim to improve existing algebraic CPD algorithms by making them more stable and consequently their computed CPDs more accurate.

- *Applying the developed methods to concrete applications.* The end goal of the development of algorithms is using them on real-life data. We therefore aim to apply the methods that are designed in this thesis to real-life applications that require either the use of tensor updating methods or more specialized decomposition algorithms than are currently available.

Following these objectives, this thesis consists of three major parts:

- *Part I: Tensors.* In the first part, we introduce tensors and give an overview of the most common types of tensor decompositions. We also discuss some relevant algebraic algorithms for their computation, as well as the state-of-the-art optimization framework [183, 223] that is used in the MATLAB toolbox Tensorlab [228] and as basis for the optimization algorithms that are developed throughout the thesis.
- *Part II: Velocity.* In the second part, we develop updating algorithms that are fast enough so that they can be applied to track tensor decompositions in real-time applications. We design an optimization algorithm for the CPD and an algebraic algorithm for the LMLRA.
- *Part III: Variety.* In the third part, we design algorithms that can be used to obtain more accurate CPDs for specific data and noise distributions. An optimization method to compute the CPD with non-LS cost functions is described that can be applied in applications for which the standard LS cost function provides suboptimal solutions due to the distribution of the data and/or noise. We also develop an algebraic algorithm for the CPD that counters stability issues in existing pencil-based algorithms and consequently leads to significantly more accurate CPDs.

Below, a brief introduction is given for each chapter of the thesis. The interconnections between the main chapters are shown in Figure 1.4.

1.4.1 Part I: Tensors

In the remainder of this part, Chapter 2, we explain the philosophy behind the use of tensor decompositions and introduce basic concepts such as the rank of a tensor. We also give an overview of the most commonly used decompositions.

We discuss the CPD and the LMLRA in detail and also briefly introduce other alternatives such as the BTD and the TTD.

Afterwards, we explain the difference between algebraic and optimization-based algorithms to compute tensor decompositions: the former methods are restricted to using tools from the field of linear algebra, while the latter are not. For both the CPD and the LMLRA, we discuss one algebraic method in detail that is highly relevant for the other thesis chapters: the GEVD algorithm for the CPD [62, 168] and the MLSVD [47] algorithm for the LMLRA.

Finally, we introduce optimization methods and explain how they can be tailored to the computation of tensor decompositions by exploiting the multilinear structure of the specific decomposition. We discuss the Gauss–Newton (GN) algorithm, both the exact and inexact variant, for the CPD in detail, as this method is applied in several chapters of the thesis. More advanced optimization schemes are covered as well that can handle structured, bounded or coupled decompositions and decompositions that use alternatives to the standard LS distance as cost function.

1.4.2 Part II: Velocity

The tensor decomposition methods that are discussed in Chapter 2 all make one, seemingly innocent, assumption about the tensor data: they assume that all data is available and final at the start of the decomposition algorithm. In many cases this assumption is valid and the analysis of the tensor data starts only after data collection is complete. However, as tensor methods are increasingly applied to real-life applications, one often encounters cases where this assumption is clearly violated: analysis can start before all data is available or some parts of data change as later readings come in. In such cases, computing the decomposition whenever new data becomes available can be incredibly wasteful, both with respect to storage and computational resources. In Chapter 3, we therefore propose a GN algorithm to compute the CPD of a tensor that does not adhere to the aforementioned assumptions. The method replaces all old data by its earlier computed CPD and computes an updated CPD that takes both the old CPD and the newly added data into account. The base method allows for the simultaneous addition of any number of $(N-1)$th-order tensors to an Nth-order tensor. By using the CPD of the old tensor data and exploiting its multilinear structure in the computation of gradients, Hessian approximations and preconditioners for optimization method, the storage and computational requirements of the method can be orders of magnitude lower than when batch methods are used. Consequently, tracking the CPD in real-time applications becomes feasible. We additionally discuss windowing strategies to remove the

dependence on outdated data and we supply an algebraic initialization to reduce the required number of optimization steps to convergence. Finally, we generalize the method to a much wider class of updating problems. Instead of adding $(N-1)$th-order tensors, the new data can arrive in any form and can even replace earlier data, opening up a whole new avenue of applications. As an example, the CPD is tracked for a tensor of which a random subset of entries becomes available at specific time intervals.

In Chapter 4 the method from Chapter 3 is extended to low-rank weighted LS (WLS) cost functions. WLS cost functions allow different tensor entries to be weighted differently, for instance when the quality of the collected data differs between sensors. If the weight tensor for the CPD is a low-rank tensor, its multilinear structure can be exploited to make optimization methods, such as the GN method much more efficient [16]. By combining the low-rank WLS framework with the CPD updating method from Chapter 4 we develop an algorithm that can track the CPD of a tensor under varying environmental conditions. For instance, sensors can break down or the ambient temperature can influence the quality of the measurements. We update both the weight tensor and the data tensor in parallel and again supply efficient expressions for the computations of the GN algorithm. We also adapt the windowing and initialization strategies to the WLS context. The method is demonstrated on a direction-of-arrival (DOA) experiment wherein a sensor breakdown occurs during the tracking. In contrast to the LS method, the WLS method manages to keep tracking the sources despite the inaccurate readings from the broken sensors.

Where Chapters 3 and 4 focus on the CPD, the LMLRA is targeted in Chapter 5. We first introduce the MLUTVD as an alternative to the MLSVD [47] to obtain an LMLRA of a tensor. Compared to the MLSVD, the MLUTVD also allows a near-optimal truncation of the tensor to its principal subspaces in each mode, but MLUTVD can be obtained more efficiently. The price that is to be paid for the lower computation time is that, unlike the MLSVD, truncation of the MLUTVD to arbitrary multilinear rank does not necessarily provide an LMLRA with a low approximation error. In practice, however, truncation to the principal subspace is very common, making the MLUTVD very useful for applications. Next, we show that the MLUTVD can be updated more efficiently than the MLSVD when new data arrives in the form of $(N-1)$th-order tensors for an Nth-order tensor. The updating is performed by independently tracking the principal subspaces of the different modes and efficiently updating the core tensor. For the nonevolving modes, the new data arrives in the form of new columns and an SURV [243] decomposition is tracked, which offers good numerical stability. For the evolving mode, the new data arrives in the form of new rows, and we track a rank-revealing URV decomposition [73]. By combining

error bounds for the different modes, we supply and prove global upper bounds for the error of the tracked MLUTVD for both the Frobenius and spectral norms. Finally, we propose some strategies that can speed up the updates considerably, but come at the cost of a slightly higher approximation error.

1.4.3 Part III: Variety

For optimization methods, the LS distance is the optimal cost function when the data can be presumed to be perturbed by additive Gaussian noise, either through random fluctuations or as a cause of the data acquisition method. In many applications, this is (approximately) the case, but there are also important classes of applications, such as musical source separation [79] and low-light image analysis [194] wherein the data and noise have significantly different distributions and the LS distance is no longer optimal. In Chapter 6, we therefore develop a generalized GN (GGN) algorithm that can compute the CPD of a tensor with non-LS cost functions, as briefly introduced in Chapter 2. The use of an appropriate alternative cost function in lieu of the LS distance can lead to more meaningful CPDs that correspond better to the underlying physical components in the data. We derive expressions for the gradient and Hessian approximation and exploit the multilinear structure of the CPD to obtain a GGN method that shows superlinear convergence close to the solution. We demonstrate that, despite having more costly iterations, the method can compete with existing first-order methods [37, 105] due to its significantly lower number of iterations. We tailor the method to β-divergence [100] cost functions in particular, a class of cost functions that is parameterized by the value of β and includes the LS distance, the Kullback–Leibler divergence and the Itukura–Saito divergence. We additionally suggest a number of effective strategies to apply the method to the nonconvex cost functions in this class. Finally, we demonstrate the effectiveness of the algorithm on hyperspectral images.

While the GGN method from Chapter 6 is very effective for smaller tensors, it does not scale well to larger tensor dimensions. To handle larger tensors, we develop an inexact variant of the algorithm in Chapter 7. Instead of computing and inverting the full Hessian approximation method, a conjugate gradients (CG) [195] approach is used which only requires the efficient computation of Gramian-vector products. We derive expressions that allow these products to be evaluated efficiently and combine them with effective preconditioners to limit the number of CG iterations that the method requires. This leads to an algorithm to compute the CPD of a tensor with non-LS cost functions that scales much better than the exact GGN method and can compete with first-order methods for tensors with much larger dimensions. As a specific example, we tailor the method to Rician distributed data, which often arises

in functional magnetic resonance imaging (fMRI) [125]. To further accelerate the method, we propose to compress the second-order information with the use of an LMLRA. The multilinear structure of the LMLRA can be leveraged to obtain more efficient iterations without sacrificing the fast convergence of the method. For very large tensor dimensions, the inexact GGN algorithm can even be embedded in the randomized block sampling framework [222], to convert it into a stochastic method that outperforms existing methods such as ADAM [120]. The good performance of the method is demonstrated on tensors with varying dimensions and ranks and on simulated fMRI data.

It can be useful to compute a rank-1 approximation of a tensor to extract one prominent component from the data. When a rank-1 CPD of a tensor is required with a β-divergence cost function, the GGN method of Chapter 6 can be made more efficient. In Chapter 8, we combine the properties of the rank-1 structure and the β-divergence cost function to obtain a more efficient GGN algorithm. We carefully derive improved expressions for the gradient, Hessian approximation and preconditioners and show that the method can outperform first-order methods for specific values of β.

For specific applications such as fMRI analysis, the collected data does not abide to the assumptions for which the LS cost function is optimal [125]. In Chapter 9, we instead use β-divergence cost functions to compute a CPD of a mixture of fMRI sources wherein the components of the CPD correspond to the original unmixed sources. We show that a higher correlation can be obtained between the original and the recovered sources when a suitable β-divergence cost function is used compared to the LS cost function.

Even though non-LS cost functions can provide better solutions for particular applications, the LS cost function is still a suitable choice in many applications. Because of its simplicity and versatility, many algebraic [43, 60, 62, 168, 186] and optimization-based [66, 95, 152, 159, 162, 183, 185] methods have been developed to compute CPDs with the LS cost function. Algebraic methods are usually applied to obtain good initializations for the slower, but more accurate, optimization methods. The GEVD algorithm [62, 168] and other algebraic algorithms are generally pencil-based, i.e., they only look at two slices of a third-order tensor and compute the factors of the CPD from it, even though this has been shown to be numerically unstable [13]. In Chapter 10, we drastically improve the stability of the GEVD algorithm by combining information from multiple pencils. From each pencil, we only split off the generalized eigenspaces corresponding to sets of well-separated generalized eigenvalues instead of trying to split off all generalized eigenvectors at once. We then reduce the problem size, construct a new pencil and continue iteratively splitting off generalized eigenspaces until all generalized eigenvectors have been separated. Because generalized eigenvectors are only split in pencils wherein their generalized eigenvalues

are well-separated, this dramatically increases the stability of the resulting GESD algorithm and consequently the accuracy of the CPD. We derive both probabilistic and deterministic bounds that express the maximal noise level for which the generalized eigenspaces can still be expected to be accurately split.

1.4.4 Conclusion and appendices

In Chapter 11, we summarize the results of this thesis per chapter, highlight the contributions of the doctoral candidate and give perspectives for future work.

We demonstrate the CPD updating method from Chapter 3 in Appendix A on real-life data through the analysis of the coupling of brain hemodynamics in neonates. For different measured signals, smilarity matrices for different segments are concatenated to form a tensor. The CPD updating algorithm is then applied for the incremental analysis of the impact of propofol, a sedative, in the coupling between the different signals. The updating method is successful in showing that a drop in the interaction between signals can be distinguished due to the administration of propofol.

In Appendix B, detailed expressions are given for the higher-order case of the method from Chapter 7. Using these expressions, the gradient, Gramian-vector product and preconditioners can be evaluated efficiently.

Finally, in Appendix C, a preview is given of some of the algorithms that will be included in the upcoming release of Tensorlab [228] 4.0. The aforementioned low-rank WLS algorithm [16] and the methods from this thesis will be included, as well as the LS-CPD method [17] for computing the CPD of a tensor that is only implicitly available as the solution of an underdetermined linear system. Aside from these, several other new algorithms will be included, alongside numerous improvements under the hood.

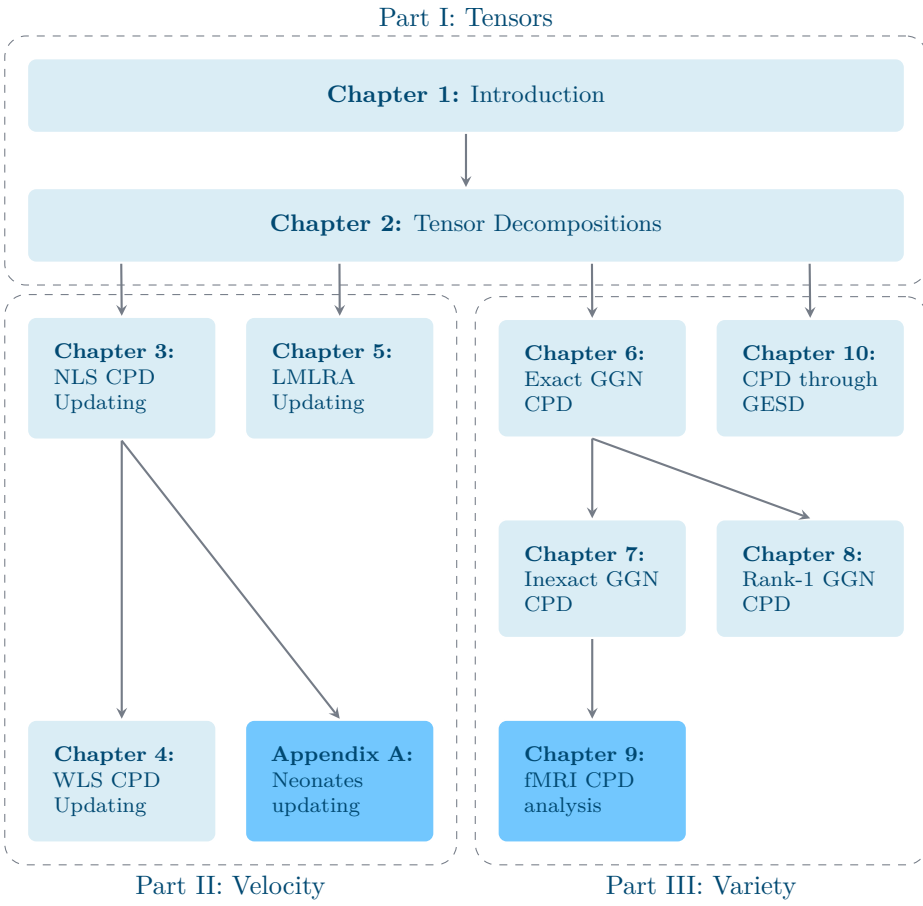

Figure 1.4: Schematic overview of the interconnections between chapters. Chapters that focus on applications are shown in a darker shade of blue.

Tensor decompositions

2

2.1 Introduction

Over the years, people have become accustomed to storing data in tables and spreadsheets. This makes sense, as both paper and computer screens are two-way by nature. Data, however, do not adhere to these physical limitations. When storing temperature data of a room, for instance, this is generally done using three coordinates, making the data three-way. If the temperature changes in time, the data become four-way. Many other applications have an even higher number of variables. While storing such data in two-way tables, or matrices, is possible, this cannot be done without sacrificing the higher-order structure of the data and consequently, losing information. Fortunately, tensors, higher-order extensions of vectors and matrices, alleviate this problem. Tensor data can be of any order and can easily be stored in a computer while maintaining its higher-order structure. In some cases, it can even be beneficial to (explicitly or implicitly) *tensorize* a matrix into a higher-order tensor to better extract its useful information [52].

The more modes a tensor has, the faster the number of entries grows when its dimensions are increased. For instance, doubling all dimensions of a matrix makes it four times larger, while doubling all dimensions of a tenth-order tensor makes it 1024 times larger. As another example, a fortieth-order tensor with all dimensions equal to two has over one trillion entries. This means that using (high-order) tensors can easily put a strain on the available resources, as they require large amounts of memory to store and are computationally expensive to manipulate. This phenomenon is often called the *curse of dimensionality*. This curse can be alleviated in several ways, but they all boil down to the same fundamental idea: trying to store the data with fewer variables by exploiting its intrinsic structure without losing too much information. In analogy to matrix decompositions, this desire for compact and/or interpretable representations has fueled the search for meaningful tensor decompositions [35, 174]. Such decompositions express the data tensor as a combination of vectors, matrices and other (smaller) tensors in such a way that storage requirements are reduced and operations on/with the tensor are less computationally expensive. For real-life data, the different components of the tensor decomposition often allow a physical interpretation as well. In tensor-based blind source separation

(BSS) [37, 45], for instance, each component corresponds to a specific source. Consequently, tensor decompositions can be used for a wide range of applications such as data compression, data analysis, learning and modeling.

In the rest of this chapter, we introduce notation and basic terminology for tensor decompositions. Next, we give an overview of several well-known decompositions. We focus in particular on the canonical polyadic decomposition (CPD) and the low multilinear rank approximation (LMLRA), as these two decompositions are discussed extensively in this thesis. Finally, we cover the two main types of methods to compute a tensor decomposition: algebraic methods that only use approaches from the field of linear algebra and optimization-based methods, which do not have this limitation. The latter can be more accurate, but usually come at the cost of a more involved and computationally expensive algorithm.

2.1.1 Notation and terminology

We denote scalars, vectors, matrices and tensors by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase letters ($\mathbf{A}$) and calligraphic letters ($\mathcal{A}$), respectively. The transpose and Hermitian transpose of a matrix $\mathbf{X}$ are written as $\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{\mathrm{H}}$, respectively. We write $\mathrm{diag}(\mathbf{X})$ for the column vector holding the diagonal entries of the matrix $\mathbf{X}$ and $\mathrm{Diag}(\mathbf{x})$ forms a square matrix that has all entries equal to zeros, except for the vector $\mathbf{x}$ on its diagonal. The $M \times M$-identity matrix is written as $\mathbf{I}_M$. The horizontal concatenation of two matrices $\mathbf{A}$ and $\mathbf{B}$ is written as $[\,\mathbf{A}, \mathbf{B}\,]$, while their vertical concatenation is written as $[\,\mathbf{A}; \mathbf{B}\,]$. When using subscripts to specify entries, a colon refers to all entries of a specific mode, e.g., $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. When formatting the entries of an Nth-order tensor $\mathcal{T}$ in matrix format, the notation $\mathbf{T}_{[i_{m_1}, i_{m_k}; i_{m_{k+1}}, i_{m_N}]}$ is used, with $[i_{m_1}, i_{m_k}; i_{m_{k+1}}, i_{m_N}]$ providing the processing order of the different tensor modes, in the row and column dimensions, i.e., the entries before the semicolon (;) specify the order in which the different modes are processed (from the mode that runs the slowest to the mode that runs the fastest) in the rows of the matrix, while those after the semicolon specify the mode order for the columns. For instance, for a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$ one has $\mathbf{a}_{i_1,i_2,i_3} = (\mathbf{A}_{[1;2,3]})_{i_1,(i_2-1)I_3+i_3}$. If all modes appear in the column dimension, the end result is a vector. In this case, the semicolon is dropped and a lowercase letter is used: for a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$, $\mathbf{t}_{[3,2,1]} \in \mathbb{C}^{I_1 I_2 I_3 \times 1}$ refers to the standard mode order for the vectorization of a third-order tensor. This operation is alternatively written as $\mathrm{vec}(\mathcal{T})$.

Subtensors

There are many ways to extract subtensors from a given tensor. Some common subtensors are defined below.

Definition 2.1.1: Mode-(m,n) slice
A mode-(m,n) slice of a tensor is a matrix obtained by fixing all indices but the mth and the nth.

For example, for a third-order tensor, the mode-$(1,2)$ slice $\mathbf{T}_{::1}$ is a frontal slice, the mode-$(1,3)$ slice $\mathbf{T}_{:1:}$ is a lateral slice and the mode-$(2,3)$ slice $\mathbf{T}_{1::}$ is a horizontal slice.

Definition 2.1.2: Mode-n vector
A mode-n vector of a tensor is a vector obtained by fixing all indices but the nth.

For instance, $\mathbf{t}_{11:}$ is the mode-3 vector in the upper left corner of a third-order tensor $\mathcal{T}$. Mode-n vectors can be used to define the mode-n unfolding of a tensor.

Definition 2.1.3: Mode-n unfolding
The mode-n unfolding $\mathbf{T}_{[(n;N,N-1,\ldots,n+1,n-1,n-2,\ldots,1]}$ of an Nth-order tensor $\mathcal{T}$ is constructed by stacking all its mode-n vectors as columns of a matrix in the specified mode order (the right-most index runs the fastest). The mode-n unfolding is also written as $\mathbf{T}_{(n)}$.

For example, in computing the mode-2 unfolding of a third-order tensor $\mathcal{T}$, $t_{ijk} \in \mathbb{C}^{I\times J\times K}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{j,q}$, with $q = i + (k-1)I$.

Products

Aside from the standard matrix product, written as $\mathbf{AB}$ or $\mathbf{A}\cdot\mathbf{B} \in \mathbb{C}^{I\times K}$ for matrices $\mathbf{A} \in \mathbb{C}^{I\times J}$ and $\mathbf{B} \in \mathbb{C}^{J\times K}$, other useful products of vectors, matrices and tensors exist.

Definition 2.1.4: Kronecker product
The Kronecker product of matrices $\mathbf{A} \in \mathbb{C}^{I\times J}$ and $\mathbf{B} \in \mathbb{C}^{K\times L}$ is defined as

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{C}^{IK\times JL}.$$

Definition 2.1.5: Khatri–Rao product
For $\mathbf{A} \in \mathbb{C}^{I \times K}$ and $\mathbf{B} \in \mathbb{C}^{J \times K}$, the column-wise Khatri–Rao product is defined as $\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_{:1} \otimes \mathbf{b}_{:1} & \cdots & \mathbf{a}_{:K} \otimes \mathbf{b}_{:K} \end{bmatrix}$, while the row-wise Khatri–Rao product of $\mathbf{A} \in \mathbb{C}^{K \times I}$ and $\mathbf{B} \in \mathbb{C}^{K \times J}$ is $\mathbf{A} \odot^{\mathrm{T}} \mathbf{B} = \left(\mathbf{A}^{\mathrm{T}} \odot \mathbf{B}^{\mathrm{T}}\right)^{\mathrm{T}}$.

Definition 2.1.6: Hadamard product
The Hadamard, or entry-wise, product of $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{I \times J}$ is written as $\mathbf{A} * \mathbf{B}$.

Definition 2.1.7: Outer product of vectors
The outer product of vectors $\mathbf{a} \in \mathbb{C}^{I}$ and $\mathbf{b} \in \mathbb{C}^{J}$ is denoted by $\mathbf{C} = \mathbf{a} \otimes \mathbf{b} \in \mathbb{C}^{I \times J}$, with $c_{ij} = a_i b_j$.

The outer product of N vectors: $\mathbf{a}^{(1)} \otimes \mathbf{a}^{(2)} \otimes \cdots \otimes \mathbf{a}^{(N)}$, with $\mathbf{a}^{(n)} \in \mathbb{C}^{I_n}$, forms an Nth-order tensor $\mathcal{C} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$, with $c_{i_1 i_2 \ldots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N}$.

Definition 2.1.8: Mode-n tensor-matrix product
The mode-n tensor-matrix product of $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathbf{X} \in \mathbb{C}^{J \times I_n}$ is written as $\mathcal{T} \cdot_n \mathbf{X}$. This product replaces every mode-n vector $\mathbf{t}$ of $\mathcal{T}$ by $\mathbf{X}\mathbf{t}$.

For example, $\mathcal{T} \cdot_1 \mathbf{X} \in \mathbb{C}^{J \times I_2 \times I_3}$ is obtained by replacing every mode-1 vector $\mathbf{t}_{:i_2 i_3}$ of $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times I_3}$ by $\mathbf{X}\mathbf{t}_{:i_2 i_3}$. In general, for $\mathcal{S} = \mathcal{T} \cdot_n \mathbf{X}$, one has $\mathbf{S}_{(n)} = \mathbf{X}\mathbf{T}_{(n)}$ in unfolded form.

There are several useful identities that relate the aforementioned products [132]:

$$(\mathbf{A} \odot \mathbf{B})^{\mathrm{T}} \cdot (\mathbf{C} \odot \mathbf{D}) = (\mathbf{A}^{\mathrm{T}} \cdot \mathbf{C}) * (\mathbf{B}^{\mathrm{T}} \cdot \mathbf{D})$$

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \otimes (\mathbf{B} \cdot \mathbf{D})$$

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \odot \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \odot (\mathbf{B} \cdot \mathbf{D})$$

$$\mathrm{vec}(\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C}) = (\mathbf{C}^{\mathrm{T}} \otimes \mathbf{A}) \cdot \mathrm{vec}(\mathbf{B}).$$

Rank

For matrices, the column and row rank are defined as the number of linearly independent columns and rows of the matrix, respectively. As they are always equal, they are generally simply called the rank of the matrix. For tensors, the

situation is different. Not only are there several meaningful ways to define the rank of a tensor, their values are also not equal in general. Below, we list some of the most commonly used notions of rank.

Definition 2.1.9: Rank
A Nth-order tensor is a rank-1 tensor if it is the outer product of N nonzero vectors. The minimal number of terms needed to write the tensor as a sum of rank-1 tensors, is the rank R.

The tensor rank generalizes the matrix property that states that the rank of a matrix is equal to the minimal number of terms required to write the matrix as a sum of rank-1 matrices.

Definition 2.1.10: Mode-n rank
The mode-n rank R_n of a tensor $\mathcal{T}$ is the (matrix) rank of its mode-n unfolding $\mathbf{T}_{(n)}$.

Definition 2.1.11: Multilinear rank
The multilinear rank of an Nth-order tensor $\mathcal{T}$ is the tuple $(R_1, R_2, \ldots, R_N)$ holding all mode-n rank values R_n.

Unlike the column rank and row rank of a matrix, the mode-n rank values are not equal in general and they do not have to equal the rank of the tensor.

Example 2.1.1:
For the tensor $\mathcal{T} \in \mathbb{R}^{2\times 2\times 2}$ with $\mathbf{T}_{::1} = \mathbf{T}_{::2} = \mathbf{I}_2$, the rank is 2, while the multilinear rank is equal to $(2, 2, 1)$.

2.2 Decompositions

While storing and manipulating tensor data can be very costly, the useful information in the tensor can often be represented more efficiently using a suitable tensor decomposition. The benefit of such a representation is two-fold. On the one hand, storage and manipulation are usually much less expensive for a tensor decomposition compared to the full tensor. On the other hand, the individual components of the tensor decomposition can also directly correspond to physical components that one wants to extract from the data. The CPD and the LMLRA are two well-known decompositions that have been used in one form or another for decades, but other decompositions such as the block-term decomposition and the tensor train decomposition have gained popularity in recent years as well.

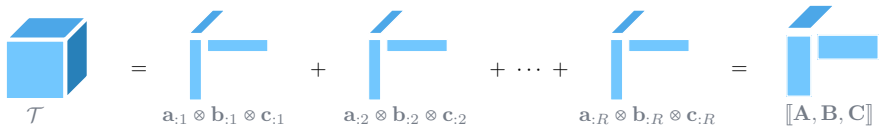

Figure 2.1: A CPD represents the tensor as a sum of R rank-1 terms, leading to a compact decomposition that is unique under mild conditions.

2.2.1 Canonical polyadic decomposition (CPD)

Definition 2.1.9 tells us that the rank of a tensor is equal to the minimal number of terms that are required to write the tensor as a sum of rank-1 terms. By explicitly constructing a decomposition of this form, one obtains a CPD of the tensor, wherein the term *canonical* specifies that the number of terms is minimal, in contrast to other polyadic decompositions.

Definition 2.2.1: CPD
A canonical polyadic decomposition (CPD) [20,35,102,174] of an Nth-order tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2, \times \cdots \times I_N}$ is defined as:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \otimes \mathbf{u}_r^{(2)} \otimes \cdots \otimes \mathbf{u}_r^{(N)} =: \left[\!\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\!\right].$$

The column vectors $\mathbf{u}_r^{(n)} \in \mathbb{C}^{I_n}$, for $n \in [1, 2, \ldots, N]$ are collected as columns of factor matrices $\mathbf{U}^{(n)} \in \mathbb{C}^{I_n \times R}$.

In Figure 2.1, the CPD of a third-order tensor is represented schematically. For a CPD $\mathcal{T} = [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$, its mode-$n$ unfolding is as follows:

$$\mathbf{U}^{(n)} \left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \mathbf{U}^{(n-2)} \odot \mathbf{U}^{(1)} \right)^{\mathrm{T}}.$$

In practice, one usually wants to compute a rank-R CPD of a tensor $\hat{\mathcal{T}} = \mathcal{T} + \mathcal{N}$, wherein $\mathcal{T}$ is a rank-R tensor and $\mathcal{N}$ consists of noise. Because of the noise, a rank-R polyadic decomposition can only be an approximation of the unperturbed rank-R tensor $\mathcal{T}$. Still, we will use the term CPD to refer to such an approximation. An important caveat is that the problem of finding the best rank-R approximation with $R > 1$ can be ill-posed for tensors of order three and above [51], i.e., the best rank-R approximation of a tensor does not necessarily exist. A sequence of low-rank tensors can converge to a tensor of higher rank with some of the components becoming arbitrarily large and (almost) canceling each other out. When computing rank-R approximations of tensors, one should thus be careful to avoid this phenomenon of diverging components.

By computing a rank-R CPD, the information in an Nth-order tensor $\mathcal{T} \in \mathbb{C}^{I \times I \times \cdots \times I}$ is represented with only $\mathcal{O}(NIR)$ variables instead of I^N. Especially for high-order tensors, this can greatly reduce the storage requirements of the tensor and make operations with the tensor much less computationally costly. A wide range of methods have been developed to compute the CPD of a tensor. Algebraic methods, see [43, 60, 62, 69, 95, 168, 186], which use tools from linear algebra to find the decomposition, can be very fast, but are often less accurate than optimization methods for noisy tensors. Because of this, the former are often used as an initialization for the latter. See Section 2.3 for a more extensive discussion.

The CPD is essentially unique under mild conditions on the tensor dimensions, the rank of the CPD and the structure of the factor matrices. Essential uniqueness refers to the fact that the rank-1 terms can be permuted and that each factor matrix column can be scaled and counter-scaled within the same rank-1 term, without fundamentally changing the decomposition. An example of a uniqueness condition for third-order tensors is given below. The generalized eigenvalue decomposition of two frontal slices of the tensor offers both a proof for this condition and a way to compute the CPD, see Section 2.3.1.

Property 2.2.1: Essential uniqueness of the CPD
The CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of a third-order tensor $\mathcal{T}$ is essentially unique if two factor matrices, e.g., $\mathbf{A}$ and $\mathbf{B}$, have full column rank (column rank equal to R) and the third factor matrix, e.g., $\mathbf{C}$ does not have proportional columns.

The mild uniqueness conditions of the CPD make this decomposition highly useful for applications where multiple components are to be extracted from the data, as one wants the recovered components to correspond uniquely to their physical counterparts. For other decompositions, uniqueness is only obtained by putting restrictions such as sparsity or orthogonality on the factors of the decomposition, while these constraints do not necessarily correspond to the reality. For example, the matrix singular value decomposition (SVD) is only unique because of its orthogonal factor matrices and diagonal matrix of singular values. Without these constraints, a matrix $\mathbf{X}$ has, aside from the decomposition $\mathbf{X} = \mathbf{A}\mathbf{B}^{\mathrm{T}} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r$, with $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{B} \in \mathbb{R}^{J \times R}$, for any choice of orthogonal $\mathcal{W} \in \mathbb{R}^{R \times R}$, another valid decomposition $\mathbf{X} = (\mathbf{A}\mathbf{W})(\mathbf{B}\mathbf{W})^{\mathrm{T}}$. For an extensive overview of uniqueness conditions for the CPD, see [57–60, 123, 176].

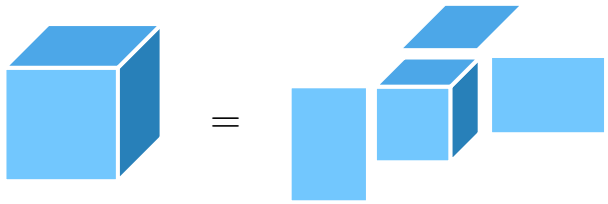

Figure 2.2: LMLRA of a third-order tensor. The tensor is represented as a core tensor with smaller dimensions, multiplied in each mode with a factor matrix.

2.2.2 Low multilinear rank approximation (LMLRA)

Definition 2.2.2: LMLRA
A low multilinear rank approximation (LMLRA) [35, 174, 208] of a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ is defined as

$$\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot_2 \mathbf{U}^{(2)} \cdots \cdot_N \mathbf{U}^{(N)} =: \left[\!\left[\mathcal{S}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\right],$$

with $\mathcal{S} \in \mathbb{C}^{R_1 \times R_2 \times \cdots \times R_N}$, $\mathbf{U}^{(N)} \in \mathbb{C}^{I_n \times R_n}$ and $R_n \leq I_n$, for $1 \leq n \leq N$.

In Figure 2.2 an LMLRA is shown schematically.

A multilinear rank-$(R, R, \ldots, R)$ LMLRA of an Nth-order $I \times I \times \cdots \times I$-tensor reduces the number of variables from I^N to $R^N + NIR$. If R is significantly smaller than I, the LMLRA structure allows tensor computations to be performed more efficiently than when the full tensor is used and also offers a large reduction in storage cost. In general, the mode-n ranks can be different between the modes. The LMLRA has an Nth-order core tensor that reflects the inter-mode interactions. This decomposition is often used to compress the information in a tensor and for subspace analysis in the different tensor modes. Ideally, the approximation increases the SNR of the data, making analysis of the resulting core tensor easier compared to the full tensor. The optimal LMLRA for the given multilinear rank is not unique in the strict sense: the multiplication of a factor matrix with a nonsingular matrix does not change the fit as long as the inverse multiplication is applied to the core tensor. Still, this indeterminacy can simply be seen as a more general form of the scaling and counterscaling that is possible for the CPD. As for matrix factorizations, strict uniqueness is often enforced by imposing additional constraints on the factor matrices and/or core tensor.

The (truncated) multilinear singular value decomposition (MLSVD) [47] is a specific form of LMLRA with structure on its factors that can be computed

algebraically. The MLSVD has orthogonal factor matrices and the core tensor is *all-orthogonal*, which for an Nth-order tensor means that the inner product of any two $(N-1)$th-order mode-n subtensors is zero. For example, for a third-order tensor, all-orthogonality means that the inner product of any two frontal slices, any two horizontal slices and any two lateral slices is zero. Generally, the columns of the N factor matrices are sorted according to the norms of their accompanying $(N-1)$th-order subtensors of the core tensor, as this allows easy truncation of the MLSVD. These norms are called the mode-n multilinear singular values. Analogous to the matrix case, keeping only the R_n factor matrix columns that correspond to the principal subspace in mode-n and discarding the other columns, along with the corresponding core tensor entries, provides an LMLRA of the tensor. Note that the Eckart–Young theorem [65] for the SVD cannot be straightforwardly generalized to tensors, so this truncated MLSVD does not necessarily provide the LMLRA with the best fit. In practice, however, the difference is often very small for tensors that arise in typical applications (although exceptions exist). Because of this, the MLSVD can serve as a great initialization for optimization methods that compute the optimal LMLRA [48, 110]. See Section 2.3 for a more detailed discussion of LMLRA computation.

Both the CPD and the LMRA can be used to represent tensor data with fewer variables, but they achieve this in a different way. A CPD is particularly effective when there is no special interaction between the different modes and the extracted rank-1 components correspond to physical components. In such cases, the mild uniqueness properties of the CPD allow for highly interpretable models. Because of this, the CPD has been successfully applied for BSS [12, 39, 49, 191], but also for array processing [142, 173, 187] and in many other applications. Compared to a CPD, an LMLRA offers significantly more flexibility, as interactions between the different modes can be represented in the core tensor. The higher flexibility often negatively influences the interpretability of the decomposition, however. For each mode, the LMLRA represents the information in the tensor in a lower-dimensional space and can thus be seen as a higher-order alternative of principal component analysis (PCA). The LMLRA is useful in cases where one is mainly interested in the mode-n principal subspaces or when compression is more important than explicitly extracting unique components, as can be the case in classification, feature extraction, and subspace-based harmonic retrieval [91, 135, 156, 220].

2.2.3 Other tensor decompositions

Aside from the CPD and the LMLRA, other types of decompositions exist. Depending on the specific application, some are more useful than others. The

block-term decompostion [44,50] (BTD) generalizes the LMLRA by decomposing the tensor as a sum of LMLRA-type terms with possibly different multilinear rank. This type of decomposition allows for more flexibility in the factors and is often used when the rank-1 requirements of the CPD components cannot be satisfied by the data. An important special case of the BTD is the decomposition in multilinear rank-$(L_r, L_r, 1)$ terms [46, 61] (LL1), for which the LMLRA-type terms have multilinear rank $(L, L, 1)$. Note that the CPD is an LL1-decomposition with L equal to 1. Other noteworthy tensor decompositions are the tensor train decomposition [151] and the hierarchical Tucker decomposition [88, 92]. The former decomposes a tensor into a train of order-3 tensors, with each tensor connected to its left and right neighbor through a summation. It can be highly effective in decreasing the storage and computational costs of high-order tensors. The latter repeatedly splits the different modes of the tensor into increasingly smaller subsets. In the end, a tree-like structure of lower-order tensors is obtained that, as for the tensor train decomposition, enables compact storage and efficient computations for high-order tensors.

Further adaptations of existing tensor decompositions can be constructed as well. For instance, one can impose nonnegativity on one or more factors of a CPD or LMLRA when decomposing a nonnegative tensor [34, 37, 129, 145, 152, 166, 184, 236, 241]. This is closely related to the field of nonnegative matrix factorization (NMF) [84], which has been successfully applied to image compression and topic modeling, among others. Other restrictions are possible as well, ranging from orthogonality of certain factor matrices to imposing Hankel- or Löwner structure on the factors or the tensor slices [53, 189, 209].

Tensor decompositions can also be coupled [3, 155, 188, 190], which means that several decompositions are computed simultaneously, while (parts of) factor matrices or 'generators' are shared between the different decompositions. Coupled decompositions are particularly useful when different types of (tensor) data are available for the same observation. In that case, one knows that the decompositions are inherently linked. A prime example of the use of coupled tensor decompositions lies in coupled fMRI-EEG analysis [4, 109].

2.3 Computing tensor decompositions

Tensor data can easily take up large amounts of memory because of its higher-order nature. Consequently, a lot of work has been done to compute tensor decompositions as efficiently as possible, ideally without sacrificing accuracy. Below, we make a distinction between algebraic and optimization-

based algorithms. Algebraic methods use tools such as SVDs and elementary rotations from the field of linear algebra to compute the decomposition. Such methods are often very fast and theoretically find the optimal solution in noiseless cases. Unfortunately, they can have trouble finding a good solution when a lot of noise is present. Also, in some cases, the algebraic solution does not coincide with the approximation with the lowest error, e.g., an algebraically computed truncated MLSVD does not offer the best LMLRA for the given ranks. Because of these reasons, algebraic solutions are often used as initialization of an optimization-based algorithm. Optimization-based methods are generally more costly than algebraic methods, but achieve a higher accuracy. Starting from a good initialization can greatly reduce the number of iterations that the optimization algorithm needs to find the solution. If the initialization is computed efficiently, this also reduces the total computation time. In Section 2.3.1, we discuss algebraic methods for both the CPD and the LMLRA. In Section 2.3.2, we explain the general framework of optimization-based algorithms for the computation of tensor decompositions.

2.3.1 Algebraic algorithms

Often, there are several ways to derive an algebraic method for the computation of a specific tensor decomposition. Below, we will focus on two particular algorithms that are relevant for this thesis: the generalized eigenvalue decomposition (GEVD) approach [62] to compute the CPD of a tensor and the truncated MLSVD approach [47] to compute an LMLRA.

Computing the CPD with a GEVD

We discuss the computation of a rank-R CPD of a third-order tensor $\mathcal{T} \in \mathbb{C}^{I \times J \times K}$. The method can be generalized to Nth-order tensors, with $N > 3$, by rearranging the higher-order tensor into a third-order tensor for the computation of the first factor matrix. To ensure uniqueness, we assume $I, J \geq R$ and $K \geq 2$ and require that the first two factor matrices have full column rank and that the third factor matrix has no proportional columns, following [168]. At the cost of a more complex algorithm, more relaxed uniqueness conditions than the ones above can be derived, see [62].

We first define generalized eigenvalues and eigenvectors.

Definition 2.3.1: Generalized eigenvalues and eigenvectors
A generalized eigenvalue of a pair of square matrices $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n \times n}$, is a λ such that $\mathbf{A}\mathbf{v} = \lambda \mathbf{B}\mathbf{v}$. The vector $\mathbf{v} \in \mathbb{C}^n$ is an associated (right) generalized eigenvector.

Left generalized eigenvectors can be similarly defined through the equation

$$\mathbf{v}^{\mathrm{H}}\mathbf{A} = \lambda \mathbf{v}^{\mathrm{H}}\mathbf{B}.$$

The GEVD of a matrix pair can generally be obtained by finding all generalized eigenvalues and associated eigenvectors (requiring $\mathcal{O}(n^3)$ flop). The generalized eigenvectors are usually collected as columns of a matrix.

We show how the GEVD relates to the computation of a rank-R CPD $[\![\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]\!]$ of a tensor $\mathcal{T} \in \mathbb{C}^{R \times R \times K}$ with $K = 2$. First, unfolding the CPD in the third mode gives us:

$$\mathbf{U}_1 \mathrm{Diag}(\mathbf{u}_{3_{1,:}}) \mathbf{U}_2^{\mathrm{H}} = \mathbf{T}_{::1}$$

$$\mathbf{U}_1 \mathrm{Diag}(\mathbf{u}_{3_{2,:}}) \mathbf{U}_2^{\mathrm{H}} = \mathbf{T}_{::2}.$$

Then, we can multiply the first equation with the inverse of the second:

$$\mathbf{U}_1 \mathrm{Diag}(\mathbf{u}_{3_{1,:}}) \mathbf{U}_2^{\mathrm{H}} \left(\mathbf{U}_1 \mathrm{Diag}(\mathbf{u}_{3_{2,:}}) \mathbf{U}_2^{\mathrm{H}}\right)^{-1} = \mathbf{T}_{::1}\mathbf{T}_{::2}^{-1}$$

$$\Leftrightarrow \mathbf{U}_1 \mathrm{Diag}(\mathbf{u}_{3_{1,:}}) \mathrm{Diag}(\mathbf{u}_{3_{2,:}})^{-1} \mathbf{U}_1^{-1} = \mathbf{T}_{::1}\mathbf{T}_{::2}^{-1}.$$

Writing $\mathrm{Diag}(\mathbf{d}) = \mathrm{Diag}(\mathbf{u}_{3_{1,:}}) \mathrm{Diag}(\mathbf{u}_{3_{2,:}})^{-1}$, we obtain:

$$\mathbf{U}_1 \mathrm{Diag}(\mathbf{d}) \mathbf{U}_1^{-1} = \mathbf{T}_{::1}\mathbf{T}_{::2}^{-1}$$

$$\Leftrightarrow \mathrm{Diag}(\mathbf{d}) \mathbf{U}_1^{-1} \mathbf{T}_{::2} = \mathbf{U}_1^{-1} \mathbf{T}_{::1}, \tag{2.1}$$

which has the form of a (transposed) GEVD. Solving (2.1) gives us $\mathbf{U}_1^{-1}$, from which the other two factor matrices can be derived. Instead of $\mathbf{U}_1^{-1}$, one could find $\mathbf{U}_2^{-1}$ through a similar procedure.

In practice, the algorithm to compute a CPD of a tensor $\mathcal{T} \in \mathbb{C}^{R \times R \times K}$ with $K = 2$ through a GEVD goes as follows. First, compute a GEVD of the two frontal slices $\mathbf{T}_{::1}$ and $\mathbf{T}_{::2}$ to obtain a matrix $\mathbf{M} \in \mathbb{C}^{R \times R}$ of generalized eigenvectors, which is equal to ${\mathbf{U}^{(1)}}^{-1}$. Using $\mathbf{M}$, the factor matrices $\mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$ can be obtained as follows. Construct the matrix $\mathbf{X} = \mathbf{T}_{(1)}^{\mathrm{T}} \mathbf{M} \in \mathbb{C}^{RK \times R}$ and compute a rank-1 approximation of the form $\mathbf{u}_{:r}^{(3)} \otimes \mathbf{u}_{:r}^{(2)}$ for each column

$\mathbf{x}_{:r}$ of $\mathbf{X}$. Combining the $\mathbf{u}_{:r}^{(2)}$ into $\mathbf{U}^{(2)}$ and the $\mathbf{u}_{:r}^{(3)}$ into $\mathbf{U}^{(3)}$ then gives the two remaining factor matrices. If desired, solving the system

$$\mathbf{U}^{(1)}(\mathbf{U}^{(3)} \odot \mathbf{U}^{(2)})^{\mathrm{T}} = \mathbf{T}_{(1)}$$

for $\mathbf{U}^{(1)}$ provides another estimate of the first factor matrix. If the third mode has dimension $K > 2$, one can simply take two linear combinations of frontal slices of the tensor and compute the GEVD. As we derive $\mathbf{U}^{(1)}$ from just two slices of the tensor, it is often beneficial to maximize the amount of information in those slices to keep the approximation error of the CPD as low as possible. The most common choice is to take the first two frontal slices of the truncated MLSVD core. Alternatively, it is also possible to take more than two slices into account, see Chapter 10.

For the computation of a rank-R CPD of a tensor $\mathcal{T} \in \mathbb{C}^{I \times J \times K}$, with $I, J \geq R$ and mode-3 rank R_3, we start by compressing $\mathcal{T}$ to a tensor $\mathcal{S} \in \mathbb{C}^{R \times R \times \min(R, R_3)}$ by computing a decomposition $\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{V}^{(1)} \cdot_2 \mathbf{V}^{(2)} \cdot_3 \mathbf{V}^{(3)}$, with $\mathbf{V}^{(1)} \in \mathbb{C}^{I \times R}$, $\mathbf{V}^{(2)} \in \mathbb{C}^{J \times R}$ and $\mathbf{V}^{(3)} \in \mathbb{C}^{K \times \min(R, R_3)}$ column-wise orthogonal. The truncated MLSVD is often a good choice in practice. We now compute a rank-R CPD of $\mathcal{S}$ using the GEVD method for the case $R \times R \times K$, with $K = \min(R, R_3)$, as outlined above, and multiply the resulting factor matrices $\mathbf{U}^{(i)}$ with their respective $\mathbf{V}^{(i)}$ on the left to obtain a CPD of $\mathcal{T}$ [27]. Pseudocode for the GEVD method can be found in Algorithm 2.1.

The GEVD approach also allows for a generalization to higher orders. First reshape the tensor into a third-order tensor and compute the inverse of the first factor matrix through the GEVD method for third-order tensors. Then compute a rank-1 approximation with $(N-1)$ modes instead of 2 to obtain the remaining $(N-1)$ factor matrices and solve a linear system to obtain $\mathbf{U}^{(1)}$.

Computing an LMLRA with the MLSVD

The MLSVD [47] of a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ can be computed algebraically by performing N SVDs. For every mode n, compute the SVD $\mathbf{U}^{(n)}\mathbf{\Sigma}^{(n)}\mathbf{V}^{(n)\mathrm{H}}$ of $\mathbf{T}_{(n)}$. The mode-n factor matrix of the requested MLSVD is then simply equal to $\mathbf{U}^{(n)}$. After finding all factor matrices in this manner, the core tensor $\mathcal{S}$ can be computed as $\mathcal{T} \cdot_1 \mathbf{U}^{(1)\mathrm{H}} \cdot_2 \mathbf{U}^{(2)\mathrm{H}} \cdots_N \mathbf{U}^{(N)\mathrm{H}}$. To obtain a multilinear rank-$(R_1, R_2, \ldots, R_N)$ truncated MLSVD, one truncates all but the first R_n columns of $\mathbf{U}^{(n)}$ before computing the core tensor $\mathcal{S}$. Pseudocode for the computation of the truncated MLSVD is given in Algorithm 2.2.

The N SVDs are independent from each other and can thus easily be parallellized. If support for parallel implementations is not available, the computation of a

Algorithm 2.1: CPD via GEVD

function $[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}] = \underline{\text{cpd_gevd}}(\mathcal{T}, R)$
Input : Tensor $\mathcal{T} \in \mathbb{C}^{I \times J \times K}$, rank R
Output : CPD $[\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$

% Compute compression to dimensions $R \times R \times K$
$[\![\mathcal{S}; \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \mathbf{V}^{(3)}]\!] = \texttt{mlsvd}(\mathcal{T}, [R, R, K])$
% Compute GEVD of first two frontal slices to find $\mathbf{A}$
$\mathbf{A} = \texttt{gevd}(\mathbf{S}_{::1}^{\mathrm{T}}, \mathbf{S}_{::2}^{\mathrm{T}})$
% Find $\mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$ through rank-1 approximations
$\mathbf{X} = \mathbf{S}_{(1)}^{\mathrm{T}} \mathbf{A}$
for *r=1:R* **do**
  Solve $\mathbf{u}_{:r}^{(3)} \otimes \mathbf{u}_{:r}^{(2)} = \mathbf{x}_{:r}$ for $\mathbf{u}_{:r}^{(3)}$ and $\mathbf{u}_{:r}^{(2)}$
end
% Solve system to find $\mathbf{U}^{(1)}$
$\mathbf{U}^{(1)} (\mathbf{U}^{(3)} \odot \mathbf{U}^{(2)})^{\mathrm{T}} = \mathbf{S}_{(1)}$
% Expand to full tensor
$\mathbf{U}^{(1)} \leftarrow \mathbf{V}^{(1)} \mathbf{U}^{(1)}$
$\mathbf{U}^{(2)} \leftarrow \mathbf{V}^{(2)} \mathbf{U}^{(2)}$
$\mathbf{U}^{(3)} \leftarrow \mathbf{V}^{(3)} \mathbf{U}^{(3)}$

truncated MLSVD computation can be sped up by using sequential truncation instead. A sequentially truncated MLSVD (STMLSVD) is obtained by setting $\mathcal{T} \leftarrow \mathcal{T} \cdot_n \mathbf{U}^{(n)^{\mathrm{H}}}$ immediately after computing $\mathbf{U}^{(n)}$ for each mode n and continuing with this new $\mathcal{T}$. The tensor $\mathcal{T}$ thus gradually decreases in size, which can offer a significant speedup for the SVDs of the modes that are processed later. This approach can lead to a slightly larger approximation error though, see the analysis in [217]. Pseudocode for the (STMLSVD) is given in Algorithm 2.3.

Neither the truncated MLSVD nor the sequentially truncated MLSVD necessarily provide the best multilinear rank-$(R_1, R_2, \ldots, R_N)$ LMLRA, so the Eckart–Young theorem for the SVD cannot be straightforwardly generalized to tensors. In many practical applications, however, they are very close to optimal. For both approximations, error bounds for the approximation of the tensor can be derived in function of the size of the truncated multilinear singular values, see [47, 217].

Algorithm 2.2: Truncated MLSVD

function $[\mathcal{S}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}] = \underline{\text{mlsvd}}(\mathcal{T})$
Input : Tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$
Output : MLSVD $[\![\mathcal{S}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$

for *n=1:N* **do**
    % Compute SVD of $\mathbf{T}_{(n)}$
    $[\mathbf{U}^{(n)}, \mathbf{S}^{(n)}, \mathbf{V}^{(n)}] = \texttt{svd}(\mathbf{T}_{(n)})$
    Look for gap between singular values in $\mathbf{S}^{(n)}$
    Set R_n equal to the number of singular values above gap
    % Extract R_n left-most singular vectors from $\mathbf{U}^{(n)}$
    $\mathbf{U}^{(n)} \leftarrow \mathbf{U}^{(n)}_{:,1:R_n}$
end
% Compute core
$\mathcal{S} = \mathcal{T} \cdot_1 \mathbf{U}^{(1)^{\mathrm{T}}} \cdot_2 \mathbf{U}^{(2)^{\mathrm{T}}} \cdots_N \mathbf{U}^{(N)^{\mathrm{T}}}$

Algorithm 2.3: Sequentially truncated MLSVD

function $[\mathcal{S}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}] = \underline{\text{stmlsvd}}(\mathcal{T})$
Input : Tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$
Output : MLSVD $[\![\mathcal{S}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$

for *n=1:N* **do**
    % Compute SVD of $\mathbf{T}_{(n)}$
    $[\mathbf{U}^{(n)}, \mathbf{S}^{(n)}, \mathbf{V}^{(n)}] = \texttt{svd}(\mathbf{T}_{(n)})$
    Look for gap between singular values in $\mathbf{S}^{(n)}$
    Set R_n equal to the number of singular values above gap
    % Extract R_n left-most singular vectors from $\mathbf{U}^{(n)}$
    $\mathbf{U}^{(n)} \leftarrow \mathbf{U}^{(n)}_{:,1:R_n}$
    % Compress $\mathcal{T}$ in mode n
    $\mathcal{T} \leftarrow \mathcal{T} \cdot_n \mathbf{U}^{(n)^{\mathrm{T}}}$
end
$\mathcal{S} = \mathcal{T}$

2.3.2 Optimization-based algorithms

Computing a tensor decomposition can also be seen as an optimization problem. The advantage of optimization methods over algebraic methods lies in the fact that they use all information in the tensor and are more robust to noise, while also allowing multilinear structure to be exploited. Naturally, the solution of an algebraic method can be used to initialize the optimization method, as the algebraic solution can typically be obtained efficiently. The goal of an optimization method is to minimize a certain metric, often referred to as the cost function or the objective function, for the difference between the tensor $\mathcal{T}$ and its approximation by a tensor decomposition $\hat{\mathcal{T}}$. Most of the time, the squared Frobenius norm of $\hat{\mathcal{T}} - \mathcal{T}$, denoted by $\|\hat{\mathcal{T}} - \mathcal{T}\|_{\mathrm{F}}^2$ is used, as this cost function corresponds to the least-squares (LS) solution and is easy to handle.

Definition 2.3.2: Frobenius norm of a tensor
The Frobenius norm $\|\mathcal{T}\|_{\mathrm{F}}$ of a tensor $\mathcal{T} \in \mathbb{C}^{I_1 \times I_2 \times \cdots I_N}$ is equal to

$$\sqrt{\mathrm{vec}(\mathcal{T})^{\mathrm{H}}\mathrm{vec}(\mathcal{T})},$$

or, alternatively, the square root of the sum of the magnitude squared tensor entries.

Aside from the squared Frobenius norm, or LS cost function, other cost functions can be used as well. While such alternative cost functions can provide better solutions in specific cases, they are generally more computationally costly to apply, see Section 2.3.2. In the rest of this section, we will use the LS cost function, unless specified otherwise.

We focus on the real CPD of a third-order tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, but the optimization framework can also be applied to complex tensors, higher-order tensors and to the computation of LMLRAs, BTDs or other tensor decompositions. For a rank-R CPD, the optimization problem has the following form:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f(\mathbf{A},\mathbf{B},\mathbf{C}) = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}\|_{\mathrm{F}}^2, \tag{2.2}$$

with $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$ and $\mathbf{C} \in \mathbb{R}^{K \times R}$. The cost function can be nonconvex and may have spurious local minima. In some cases, a global minimum does not even exist. Consequently, computing the CPD of a tensor is a difficult problem in general. Many general-purpose optimization methods, see for example [150], can be applied to solve (2.2). Especially for large tensors, however, naively applying an optimization method without exploiting the CPD structure of the approximation, can make the computational complexity impractically high.

Because of this, specialized optimization methods have been designed for the CPD, see, for instance, [2, 20, 152, 157, 183, 223].

A well-known method to compute a CPD is called alternating least-squares [20] (ALS). It is a block-coordinate descent (BCD) method that successively updates each factor matrix, while keeping the others fixed. This is attractive from a computational point of view, as one can find the new iterate for $\mathbf{A}$ through solving the linear system $\mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} = \mathbf{T}_{(1)}$ for $\mathbf{A}$. One can show that this is equivalent to computing $\mathbf{A} = \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}) \left((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})\right)^{-1}$, wherein $\left((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})\right)$ is an $(R \times R)$-matrix. New iterates for $\mathbf{B}$ and $\mathbf{C}$ can analogously be found by computing $\mathbf{B} = \mathbf{T}_{(2)}(\mathbf{C} \odot \mathbf{A}) \left((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A})\right)^{-1}$ and $\mathbf{C} = \mathbf{T}_{(3)}(\mathbf{B} \odot \mathbf{A}) \left((\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A})\right)^{-1}$, respectively. This process is iterated until it converges to a stationary point up to a requested precision level or a different stopping criterion, e.g., a maximal iteration limit, is reached. ALS is widely used and is effective for many real-life problems. Unfortunately, the method can exhibit slow convergence for harder problems, such as for problems with correlated factor matrix columns. Some adaptations of ALS have been proposed to improve its convergence behavior, with varying levels of success [95, 162, 185].

As an alternative to ALS methods, nonlinear least-squares (NLS) methods have been applied to the computation of the CPD [66, 152, 159, 183] as well. NLS methods do not alternate between the factor matrices, but optimize all factors at once. Compared to ALS, they often require much fewer iterations to obtain the solution, see Figure 2.3. For the most effective methods, though, this generally comes at a cost of more expensive iterations. An excellent overview of NLS methods for the computation of CPDs can be found in [223] and several of these methods are implemented in the MATLAB toolbox Tensorlab [228]. Below, we discuss the aspects that are relevant for this thesis.

NLS methods for the CPD

We write $\mathbf{z} = [\mathrm{vec}(\mathbf{A}); \mathrm{vec}(\mathbf{B}); \mathrm{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$, for the vector holding all optimization variables, so the optimization problem simply becomes:

$$\min_{\mathbf{z}} f(\mathbf{z}).$$

NLS methods find a series of iterates $\mathbf{z}_k$ which, when successful, eventually converge to the solution. To find the next iterate $\mathbf{z}_{k+1}$, a step from the previous iterate $\mathbf{z}_k$ is taken as follows:

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha \mathbf{p},$$

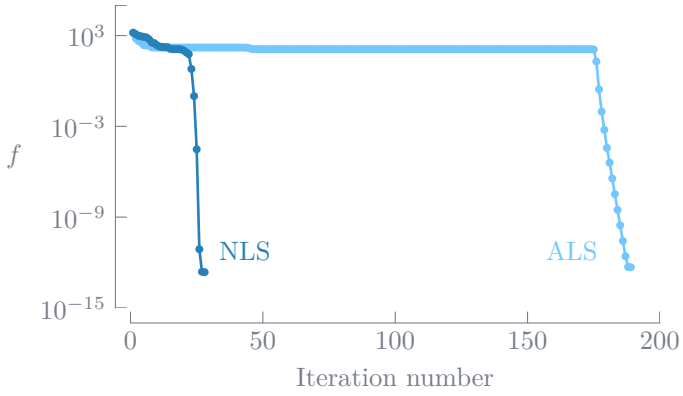

Figure 2.3: For more difficult problems, ALS methods can take many iteration steps with limited progress before eventually converging to the solution, while NLS methods converge in much fewer iterations. Computation of a CPD of a $9 \times 9 \times 9 \times 9 \times 9$ tensor of rank 11 [223].

with $\mathbf{p}$ the step direction and α the step size. For NLS methods, the step direction is determined by constructing a local second-order approximation of the cost function around the previous iterate. One can first determine $\mathbf{p}$ and then find α through a line search [150, 185]. Alternatively, one can find both at the same time using a trust-region approach [150]. In this case, one looks for the next iterate within a ball of radius Δ wherein the local cost function approximation can be trusted. As finding the optimal step can be prohibitively expensive, a dogleg or plane search [150, 185] is often used to find a suboptimal, but still adequate step instead.

The local NLS approximation of f around $\mathbf{z}_k$ has the following form:

$$f(\mathbf{z}_k + \mathbf{p}) \approx f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}}\mathbf{g}_k + \frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{H}_k\mathbf{p}, \tag{2.3}$$

with $\mathbf{g}_k \in \mathbb{R}^{R(I+J+K)}$ the gradient of f at $\mathbf{z}_k$ and $\mathbf{H}_k \in \mathbb{R}^{R(I+J+K)\times R(I+J+K)}$ the Hessian of f at $\mathbf{z}_k$. The goal is then to find the step direction $\mathbf{p}$ that minimizes (2.3). It can easily be shown that $\mathbf{p}$ can be computed as the solution of the linear system

$$\mathbf{H}_k\mathbf{p} = -\mathbf{g}_k.$$

Using this $\mathbf{p}$ corresponds to the Newton method [150] and with it, quadratic convergence can be achieved close to the solution.

In practice, computing the full Hessian matrix is too expensive for large problems. Additionally, this matrix can be indefinite. For these reasons, a positive

semi-definite approximation $\hat{\mathbf{H}}_k$ of $\mathbf{H}_k$ is often used instead. The cruder the approximation, the cheaper it becomes to compute $\mathbf{p}$, but the worse the quality of this step direction is. Using $\hat{\mathbf{H}}_k = \mathbf{I}_{R(I+J+K)}$, for example, one obtains the classical gradient descent method. The step direction $\mathbf{p}$ is simply equal to the negative of the gradient and can be computed very efficiently, but convergence of the method is at most linear. As a compromise between accuracy and speed, one can instead use either the Gauss–Newton (GN) approximation $\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k$ or the Levenberg–Marquardt (LM) approximation $\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k + \lambda\mathbf{I}_{R(I+J+K)}$, wherein $\mathbf{J}_k \in \mathbb{R}^{IJK \times R(I+J+K)}$ is the Jacobian matrix of f, to achieve (near-)quadratic convergence close to the solution with iterations that are significantly less costly than those of the Newton method. Careful exploitation of the CPD structure in the Jacobian matrix, however, can significantly accelerate of these methods and make them effective options in practice. Below, we explain how this exploitation of the structure can be performed for the GN method. We drop the index k for ease of notation.

Gradient computation The gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g}_{\mathbf{A}}; \mathbf{g}_{\mathbf{B}}; \mathbf{g}_{\mathbf{C}}]$. The subgradients $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be found by applying the chain rule. This gives us the following property.

Property 2.3.1:
For the CPD fitting problem, the gradient terms corresponding to the different tensor modes can be computed as:

$$\mathbf{g}_{\mathbf{A}} = \operatorname{vec}\left(\mathbf{A}((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})) - \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\right)$$

$$\mathbf{g}_{\mathbf{B}} = \operatorname{vec}\left(\mathbf{B}((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A})) - \mathbf{T}_{(2)}(\mathbf{C} \odot \mathbf{A})\right)$$

$$\mathbf{g}_{\mathbf{C}} = \operatorname{vec}\left(\mathbf{C}((\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A})) - \mathbf{T}_{(3)}(\mathbf{B} \odot \mathbf{A})\right).$$

The matricized tensor times Khatri–Rao products (`MTKRPROD`) in $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be computed efficiently using specialized routines [218, 223, 226].

Hessian approximation Similar to the gradient, the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ can also be split according to the different factor matrices, but now in both the row and column modes. This gives a grid-like layout, see Figure 2.4, with three blocks in every mode. We can exploit the CPD structure of the Jacobian to compute these blocks much more efficiently than when a naive implementation is used. The diagonal blocks of the Hessian approximation are

of the form $\mathbf{J}_m^{\mathrm{T}}\mathbf{J}_m$, where $\mathbf{J}_m$ refers to the Jacobian of the mth factor matrix.

Property 2.3.2:
For the Hessian of the CPD fitting problem, the diagonal block $\mathbf{G}^{(1,1)} \in \mathbb{R}^{IR\times IR}$ is given by:

$$\begin{aligned}\mathbf{G}^{(1,1)} = \mathbf{J}_1^{\mathrm{T}}\mathbf{J}_1 &= \left((\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right)^{\mathrm{T}}\left((\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right)\\ &= \left((\mathbf{C}^{\mathrm{T}}\mathbf{C})*(\mathbf{B}^{\mathrm{T}}\mathbf{B})\right)\otimes\mathbf{I}_I,\end{aligned}$$

while the other diagonal blocks can be found analogously:

$$\mathbf{G}^{(2,2)} = \mathbf{J}_2^{\mathrm{T}}\mathbf{J}_2 = \left((\mathbf{C}^{\mathrm{T}}\mathbf{C})*(\mathbf{A}^{\mathrm{T}}\mathbf{A})\right)\otimes\mathbf{I}_J$$

$$\mathbf{G}^{(3,3)} = \mathbf{J}_3^{\mathrm{T}}\mathbf{J}_3 = \left((\mathbf{B}^{\mathrm{T}}\mathbf{B})*(\mathbf{A}^{\mathrm{T}}\mathbf{A})\right)\otimes\mathbf{I}_K.$$

The off-diagonal blocks of the Hessian approximation are of the form $\mathbf{J}_m^{\mathrm{T}}\mathbf{J}_n$.

Property 2.3.3:
The $(1,2)$-off-diagonal block $\mathbf{G}^{(1,2)} \in \mathbb{R}^{IR\times JR}$ of the Hessian approximation for the CPD fitting problem can be computed as follows:

$$\mathbf{G}^{(1,2)} = \mathbf{J}_1^{\mathrm{T}}\mathbf{J}_2 = \begin{bmatrix}\mathbf{a}_{:1}\mathbf{b}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:1}^{\mathrm{T}}\\ \vdots & \ddots & \vdots\\ \mathbf{a}_{:1}\mathbf{b}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:R}^{\mathrm{T}}\end{bmatrix} * \mathbf{W}^{(1,2)},$$

where the matrix $\mathbf{W}^{(1,2)}$ is equal to $(\mathbf{C}^{\mathrm{T}}\mathbf{C}) \in \mathbb{R}^{R\times R}$. The other off-diagonal blocks can be found analogously.

The Hessian approximation is symmetric, so only half of the off-diagonal blocks need to be computed.

Preconditioned conjugate gradients For small tensors, the full Hessian approximation can be used to explicitly solve the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ for the step direction $\mathbf{p}$. For large tensors, however, this becomes too computationally costly. In that case, the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ can be solved using preconditioned conjugate gradients (PCG) instead [195]. This iterative method only uses multiplications of the form $\hat{\mathbf{H}}\mathbf{x}$ to solve the system up to the required precision, so no costly matrix inversions are needed. We can, however, exploit the structure of the Jacobian matrix to compute these $\hat{\mathbf{H}}\mathbf{x}$-multiplications efficiently.

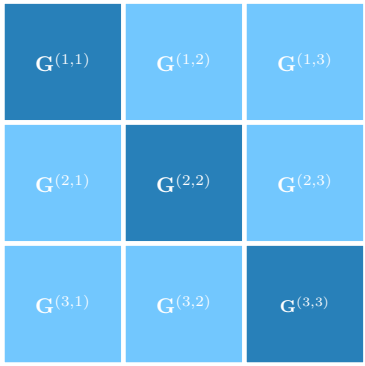

Figure 2.4: The Gauss–Newton Hessian approximation for the CPD optimization problem has a grid-like structure. The diagonal and off-diagonal blocks can both be computed efficiently by exploiting the multilinear structure through the expressions from Properties 2.3.2 and 2.3.3, respectively. The Hessian approximation is symmetric, so $\mathbf{G}^{(m,n)} = \mathbf{G}^{(n,m)^{\mathrm{T}}}$.

For the CPD, the vectors $\mathbf{x}$ and $\mathbf{y}$ always have the same structure as $\mathbf{z}$. This means that we can also split these vectors into three parts corresponding to the factor matrices: $\mathbf{x} = [\mathbf{x_A}; \mathbf{x_B}; \mathbf{x_C}] = [\mathrm{vec}(\mathbf{X_A}); \mathrm{vec}(\mathbf{X_B}); \mathrm{vec}(\mathbf{X_C})]$ and $\mathbf{y} = [\mathbf{y_A}; \mathbf{y_B}; \mathbf{y_C}]$. Using these expressions, we have

$$\mathbf{y_A} = \mathbf{G}^{(1,1)}\mathbf{x_A} + \mathbf{G}^{(1,2)}\mathbf{x_B} + \mathbf{G}^{(1,3)}\mathbf{x_C},$$

with similar expressions for $\mathbf{y_B}$ and $\mathbf{y_C}$, see Figure 2.5. The products with the diagonal blocks can be computed as

$$\mathbf{G}^{(1,1)}\mathbf{x_A} = \mathrm{vec}\left(\mathbf{X_A}((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B}))\right),$$

while the products with the off-diagonal blocks can be computed as

$$\mathbf{G}^{(1,2)}\mathbf{x_B} = \mathrm{vec}\left(\mathbf{A}((\mathbf{X_B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C}))\right).$$

The products with other blocks can be obtained analogously. Using these expressions, the multiplications of the form $\hat{\mathbf{H}}\mathbf{x} = \mathbf{y}$ can be performed very efficiently, making the CG approach significantly faster for large tensors than forming the full Hessian approximation matrix and explicitly solving the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$.

To further decrease the computational cost of solving $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$, preconditioning can be used. Preconditioning clusters the eigenvalues of the coefficient matrix by solving, instead of $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$, a modified system

$$\mathbf{M}^{-1}\hat{\mathbf{H}}\mathbf{p} = -\mathbf{M}^{-1}\mathbf{g}.$$



Figure 2.5: Using the preconditioned CG algorithm, the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ can be solved by repeatedly performing multiplications of the the form $\mathbf{y} = \hat{\mathbf{H}}\mathbf{x}$, wherein both $\mathbf{x}$ and $\mathbf{y}$ can be partitioned according to the factor matrix that each part corresponds to. On the right side, submatrices are only multiplied with vectors in the same shade of blue, leading to expressions with exploitable multilinear structure, e.g., $\mathbf{y}_\mathbf{A} = \mathbf{G}^{(1,1)}\mathbf{x}_\mathbf{A} + \mathbf{G}^{(1,2)}\mathbf{x}_\mathbf{B} + \mathbf{G}^{(1,3)}\mathbf{x}_\mathbf{C}$ can be computed efficiently without ever forming the $\mathbf{G}^{(1,m)}$-matrices in full.

The clustering of the eigenvalues can significantly reduce the number of iterations that the CG method requires to find $\mathbf{p}$ up to the requested precision. Ideally, the matrix $\mathbf{M}$ is sparse and structured, such that it can be inverted efficiently, but at the same time closely resembles $\hat{\mathbf{H}}$ such that the eigenvalues of $\mathbf{M}^{-1}\hat{\mathbf{H}}$ are more clustered than those of $\hat{\mathbf{H}}$. For the CPD, a block-Jacobi preconditioner is often effective. This preconditioner is block-diagonal and consists of the diagonal blocks of $\hat{\mathbf{H}}$. Recall that these blocks can be computed efficiently following (2.3.2). Additionally, they can be inverted cheaply because of the Kronecker structure:

$$\left(\left((\mathbf{C}^\mathrm{T}\mathbf{C}) * (\mathbf{B}^\mathrm{T}\mathbf{B})\right) \otimes \mathbf{I}_I\right)^{-1} = \left((\mathbf{C}^\mathrm{T}\mathbf{C}) * (\mathbf{B}^\mathrm{T}\mathbf{B})\right)^{-1} \otimes \mathbf{I}_I.$$

Beyond basic CPDs As mentioned above, the LS cost function is only one particular choice of cost function. By using this cost function, one inherently assumes that the noise entries are identically and independently distributed and (approximately) zero-mean Gaussian. When the noise follows a different distribution, the LS solution can be suboptimal and other cost functions can provide more meaningful results. Some examples of alternative cost functions are weighted LS cost functions [115] and β-divergence [100] cost functions, but other alternatives exist as well.

For non-LS cost functions, the CPD cost function in (2.2) is no longer valid and a more general optimization problem needs to be solved:

$$\min_{\mathbf{z}} f(\mathbf{z}) = \min_{\mathbf{z}} \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} f_{ijk}(m_{ijk}(\mathbf{z})),$$

with $m_{ijk}(\mathbf{z})$ mapping the optimization variables in $\mathbf{z}$ to the tensor elements in the current CPD estimate $\mathcal{M}$ of the optimization algorithm. Each f_{ijk} is a twice differentiable fit function for the corresponding model entry m_{ijk}. For LS, for instance, $f_{ijk} = (m_{ijk} - t_{ijk})^2$. To solve this general CPD fitting problem, we can no longer use the GN method, as this method was designed for the LS cost function specifically. Fortunately, the GN method can be generalized into the generalized GN (GGN) method [169] and this method is capable of solving the CPD fitting problem for general cost functions. The gradient of the GGN method consists of the following subgradients:

$$\mathbf{g}_{\mathbf{A}} = \text{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right),$$

$$\mathbf{g}_{\mathbf{B}} = \text{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right),$$

$$\mathbf{g}_{\mathbf{C}} = \text{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right),$$

where the tensor $\mathcal{R}$ holds the entry-wise first-order derivatives of the functions f_{ijk} at m_{ijk}. The GGN Hessian approximation is

$$\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J},$$

where $\mathbf{Z} = \text{Diag}\left(\text{vec}(\mathcal{D})\right)$ and $\mathcal{D}$ holds the entry-wise second-order derivatives of the functions f_{ijk} at m_{ijk}. As for the LS cost function, it is possible to exploit the CPD structure of the Jacobian matrix in the formation of $\hat{\mathbf{H}}$ and in the computation of multiplications of the form $\hat{\mathbf{H}}\mathbf{x}$ to significantly speed up the solution of the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$. Still, GGN iterations are significantly more expensive than GN iterations.

Non-LS cost functions such as β-divergences are often defined for nonnegative values only. In such cases, it is generally required to restrict the CPD factors so that they are nonnegative as well, leading to a bound-constrained optimization problem. The solution of such problems is more involved than for unconstrained problems, but the same techniques can be applied as for the unconstrained CPD. Aside from nonnegativity, there are many other types of constraints that can be applied to the factors of a CPD. Symmetry, where (parts) of different factor matrices are equal [118], or orthogonal [189] and Hankel-structured [209] factor matrices are only a few examples. These constraints can be implemented efficiently by only optimizing over the free variables. For instance, if two factor

matrices are the same, they only have one set of optimization variables instead of two. The same idea can be applied to enforce orthogonality, Hankel/Löwner-structure or other common types of constraints [223].

The approach for structured factors can also be used to construct coupled decompositions [3, 188, 190], where factor matrices, and thus optimization variables, are shared between multiple tensor decompositions. By only optimizing over the free variables, the coupling between the decompositions can be made explicit. Alternatively, soft coupling can be achieved by adding a penalization term to the cost function that adds a penalty that is proportional to the difference between the coupled factors [4]. Similarly, penalization can also be used to enforce sparsity of the factors through regularization terms.

For very large tensors, the (G)GN method might still be too expensive, even if all available structure is exploited. In such cases, stochastic methods can be used instead. Such methods use only a sample of entries from the tensor to form gradients and Hessian approximations instead of the full tensor. For example, the randomized block sampling method [222] samples small subtensors of the tensor, while other methods use for instance random sampling [120]. The use of stochastic methods decreases the convergence speed, but significantly speeds up individual iterations. Only the sampled entries are required to be stored in memory, and not the full tensor, making it possible to compute CPDs of tensors that are otherwise prohibitively large.

Part II: Velocity

Nonlinear least squares updating of the canonical polyadic decomposition

3

ABSTRACT

Current batch tensor methods often struggle to keep up with fast-arriving data. Even storing the full tensors that have to be decomposed can be problematic. To alleviate these limitations, tensor updating methods modify a tensor decomposition using efficient updates instead of recomputing the entire decomposition when new data becomes available. In this chapter, the structure of the decomposition is exploited to achieve fast updates for the canonical polyadic decomposition whenever new slices are added to the tensor in a certain mode. A batch NLS-algorithm is adapted so that it can be used in an updating context. By only storing the old decomposition and the new slice of the tensor, the algorithm is both time- and memory efficient. Experimental results show that the proposed method is faster than batch ALS and NLS methods, while maintaining a good accuracy for the decomposition.

This chapter is an extended version of M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Nonlinear least squares updating of the canonical polyadic decomposition," in 2017 25th European Signal Processing Conference (EUSIPCO17), Aug. 2017, pp. 693–697. The figures have been updated for consistency and a new section was added at the end that generalizes the proposed method to a wider class of structured updates. The specific contributions of the doctoral candidate are listed in Section 11.1.

3.1 Introduction

Tensor decompositions are powerful tools for various applications in machine learning and signal processing [35, 119, 174]. Tensors are higher-order extensions of vectors and matrices. They allow one to store and analyze large and higher-order datasets with the use of compact and meaningful tensor decompositions. As such, tensor tools are promising for big data applications. Several algebraic and optimization-based algorithms have been developed for tensor decompositions; see for instance [62, 157, 183]. Recently, dedicated methods have been designed for large, sparse, or incomplete tensors; see for example [154, 222, 227] and references therein. Also, support for structured and coupled tensor decompositions has been added to tensor toolboxes such as Tensorlab [184, 228].

Mainly batch methods are used to handle higher-order data, but they make one very important assumption: they assume that the full tensor is available at the start and that it does not change afterwards. As a result, these methods always compute a decomposition for the whole tensor. Yet, in real-time applications, tensors do not have to be immutable: the tensor entries might change gradually (or abruptly) over time or the tensor may grow (or shrink) in one or more modes. A tensor might even be so large that a decomposition for the full tensor cannot be computed at once, but must be built up from the decomposition of its subtensors, e.g., slice by slice. The available time to compute a decomposition may also be limited. In such cases, one would like to update the tensor decomposition using only the data that has changed. Such efficient updating methods for tensor decompositions are currently being investigated [139, 149, 199]. While the decomposition might lose some accuracy, the speed and memory efficiency of updating methods may give them an advantage in practice.

In this chapter, we exploit the structure of the Canonical Polyadic Decomposition (CPD) of a tensor to obtain a Nonlinear Least Squares (NLS) updating algorithm. We apply the framework for the computation of structured tensor decompositions proposed in [224, 226] to modify the CPD when new tensor slices become available and old slices become outdated. This yields a CPD updating method that is more efficient than the existing batch methods, while maintaining a good accuracy. Additionally, the method only has to store the old decomposition and the new tensor slice in every updating step, thus making it both time- and memory-efficient. The method also admits arbitrary windowing strategies and can be used for tensors that have a dynamic tensor rank.

We fix notation and give basic definitions in Section 3.2. We derive our method in Section 3.3, discuss numerical experiments in Section 3.4 and conclude in Section 3.5.

3.2 Notation and definitions

Scalars, vectors and matrices are denoted by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively. We refer to tensors by using letters in calligraphic script ($\mathcal{T}$). An Nth order tensor has N different modes. The outer product of N vectors, denoted by $\mathbf{v}^{(1)} \otimes \cdots \otimes \mathbf{v}^{(N)}$, is a natural extension of the outer product of two vectors. The result is an Nth-order tensor $\mathcal{T}$ of which each entry is defined as follows: $t_{i_1 \ldots i_N} = v^{(1)}_{i_1} v^{(2)}_{i_2} \cdots v^{(N)}_{i_N}$. For simplicity of notation, third-order tensors will be used throughout the rest of the chapter. A mode-n fiber $\mathbf{t}_{ij:}$, $\mathbf{t}_{i:k}$ or $\mathbf{t}_{:jk}$ of a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ is a vector obtained by fixing all indices but the nth. Likewise, a mode-(m, n) slice $\mathbf{T}_{i::}$. $\mathbf{T}_{:j:}$ or $\mathbf{T}_{::k}$ is a matrix obtained by fixing all but the mth and nth index. The Frobenius norm is denoted by $\|.\|_{\mathrm{F}}$ and the Kronecker, Khatri–Rao and Hadamard products of matrices by $\otimes$, $\odot$ and $*$, respectively, where

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1, \ldots, \mathbf{a}_R \otimes \mathbf{b}_R \end{bmatrix},$$

and $(\mathbf{A} * \mathbf{B})_{ij} = a_{ij} b_{ij}$. Herein $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 \ldots, \mathbf{a}_R \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 \ldots, \mathbf{b}_R \end{bmatrix}$. The mode-$n$ product of a tensor and a matrix is denoted by $\cdot_n$ and defined as $(\mathcal{T} \cdot_n \mathbf{G})_{i_1, \ldots, i_{n-1} j i_{n+1} i_N} = \sum_n t_{i_1, \ldots, i_N} g_{j i_n}$, where the nth dimension of $\mathcal{T}$ is equal to the number of columns of $\mathbf{G}$. The transpose of a matrix $\mathbf{M}$ is written as $\mathbf{M}^{\mathrm{T}}$ and its Moore–Penrose pseudoinverse as $\mathbf{M}^{\dagger}$. $\mathrm{vec}(\mathbf{X})$ denotes the vectorization of the matrix $\mathbf{X}$, i.e., putting the columns of $\mathbf{X}$ below each other and $\mathrm{diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its diagonal.

A third-order tensor has rank 1 if it is the outer product of three non-zero vectors. The rank of a tensor is the minimal number of terms that is needed to write the tensor as a linear combination of rank-1 tensors. This leads to the CPD of a tensor, which decomposes a rank-R tensor $\mathcal{T}$ as a linear combination of R rank-1 terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r.$$

The vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ are usually collected into factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, as follows: $\mathbf{A} = [\mathbf{a}_1 \ldots \mathbf{a}_R]$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. The CPD is written as $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_R$, or as $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ if the rank R is clear from the context. If the tensor $\mathcal{T}$ is unfolded to a matrix in the third mode by combining all its mode-3 fibers as columns of a matrix $\mathbf{T}_{(3)}$, the CPD can also be written as $\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}$ [183].

3.3 NLS updating

NLS algorithms have been shown to perform well for the computation of the CPD of a tensor, as they are efficient and very robust for more difficult decompositions [183]. In this section, we convert the batch NLS algorithm to an updating method that maintains most of these nice properties, while being more time- and memory-efficient.

Consider at time k a third order tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, with rank-$R$ CPD $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$. At time $k+1$, a frontal slice $\mathbf{M}$ is added to $\mathcal{T}$ in the third mode, as shown in Figure 3.1, forming a tensor $\mathcal{T}^{(\text{up})}$. Instead of recomputing the entire CPD to include the new slice, we want to perform an efficient update of the decomposition, both in time and memory, to obtain the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathcal{T}^{(\text{up})}$. Assuming the model stays approximately the same, the old factor matrices are used to initialize the algorithm, but $\mathbf{Z}$ is extended with a new row $\mathbf{c}_{\text{new}}^{\text{T}}$: thus $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{Y}$ and $\mathbf{C} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{c}_{\text{new}}^{\text{T}} \end{bmatrix}$. The vector $\mathbf{c}_{\text{new}}$ can be obtained from the new slice by computing the least squares solution of $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\text{new}} = \text{vec}(\mathbf{M})$.

Property 3.3.1:
An algebraic initialization for the vector The vector $\mathbf{c}_{\text{new}}$ corresponding to the new tensor slice can be computed through:

$$\begin{aligned} \mathbf{c}_{\text{new}} &= (\mathbf{Y} \odot \mathbf{X})^{\dagger} \text{vec}(\mathbf{M}) \\ &= \left[(\mathbf{Y} \odot \mathbf{X})^{\text{T}} (\mathbf{Y} \odot \mathbf{X}) \right]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\text{T}} \text{vec}(\mathbf{M}) \\ &= [(\mathbf{Y}^{\text{T}}\mathbf{Y}) * (\mathbf{X}^{\text{T}}\mathbf{X})]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\text{T}} \text{vec}(\mathbf{M}). \end{aligned} \tag{3.1}$$

As R is typically small, computing the inverse of the $R \times R$-matrix $[(\mathbf{Y}^{\text{T}}\mathbf{Y}) * (\mathbf{X}^{\text{T}}\mathbf{X})]$ is not expensive (and singularity of this matrix points to an overestimation of R), while $(\mathbf{Y} \odot \mathbf{X})^{\text{T}} \text{vec}(\mathbf{M})$ can be obtained without explicitly forming the Khatri–Rao product: the rth entry of the resulting vector is equal to $\mathbf{x}_{:r}^{\text{T}} \mathbf{M} \mathbf{y}_{:r}$.

In the remainder of this section, we describe an NLS method that can efficiently update an existing CPD when a new slice is added to the tensor, while only the factor matrices and the new tensor slice are stored. The method builds on the framework for the efficient representation of structured tensors described in [224, 226]. We exploit the structure of the CPD to derive efficient expressions for the objective function, gradient and Gramians that are needed in the NLS method. Windowing strategies are also discussed, as are dynamic tensor ranks.

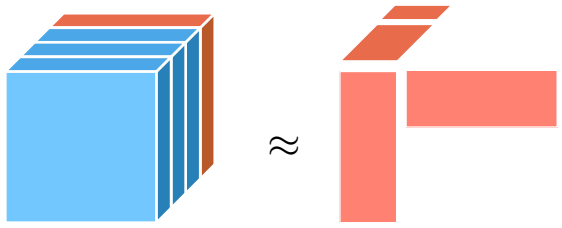

Figure 3.1: Example of the tensor updating procedure. Left: the original tensor is extended with an extra slice (red) in the third mode. Right: the CPD of the tensor is updated by adding a new vector to the factor matrix in the third mode and modifying the existing factor matrices.

The section ends with an analysis of the complexity of the algorithm. The full algorithm is given in Algorithm 3.1.

3.3.1 Objective function

We compute the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the updated tensor $\mathcal{T}^{(\text{up})}$ by minimizing the following objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}^{(\text{up})}\|_{\text{F}}^2.$$

We partition the factor matrix $\mathbf{C}$ as $\begin{bmatrix} \overline{\mathbf{C}} \\ \underline{\mathbf{c}} \end{bmatrix}$, with $\underline{\mathbf{c}}$ the last row of $\mathbf{C}$ and rewrite f as

$$f = \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] - \mathcal{T}\|_{\text{F}}^2 + \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] - \mathbf{M}\|_{\text{F}}^2,$$

which can be expanded to

$$f = \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!]\|_{\text{F}}^2 - \langle [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!], \mathcal{T} \rangle + \frac{1}{2} \|\mathcal{T}\|_{\text{F}}^2 + \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!]\|_{\text{F}}^2 - \langle [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!], \mathbf{M} \rangle + \frac{1}{2} \|\mathbf{M}\|_{\text{F}}^2,$$

wherein the notation $\langle \mathcal{Q}, \mathcal{R} \rangle$ for two tensors $\mathcal{Q}$ and $\mathcal{R}$ of equal dimensions denotes the inner product $\text{vec}(\mathcal{R})^{\text{T}}\text{vec}(\mathcal{Q})$. One can note that $\frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!]\|_{\text{F}}^2 + \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!]\|_{\text{F}}^2 = \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\text{F}}^2$. The full tensor $\mathcal{T}$ is not stored during the updating process for memory efficiency. Instead, we work with its CPD approximation $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$, which is the best available guess of $\mathcal{T}$. Using this

CPD approximation, we obtain the following objective function:

$$f \approx \frac{1}{2}\|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_{\mathrm{F}}^2 - \left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] \right\rangle + \frac{1}{2}\|[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\|_{\mathrm{F}}^2 - \langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\rangle + \frac{1}{2}\|\mathbf{M}\|_{\mathrm{F}}^2. \quad (3.2)$$

Equation (3.2) can be further simplified to avoid the construction of full tensors by exploiting the structure of the CPD. First,

$$\begin{aligned}\|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_{\mathrm{F}}^2 &= \mathrm{vec}([\![\mathbf{A},\mathbf{B},\mathbf{C}]\!])^{\mathrm{T}}\mathrm{vec}([\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]) \\ &= \mathbf{1}^{\mathrm{T}}(\mathbf{C}\odot\mathbf{B}\odot\mathbf{A})^{\mathrm{T}}(\mathbf{C}\odot\mathbf{B}\odot\mathbf{A})\mathbf{1} \\ &= \mathbf{1}^{\mathrm{T}}[(\mathbf{A}^{\mathrm{T}}\mathbf{A})*(\mathbf{B}^{\mathrm{T}}\mathbf{B})*(\mathbf{C}^{\mathrm{T}}\mathbf{C})]\mathbf{1}, \quad (3.3)\end{aligned}$$

where $\mathbf{1}$ is a vector of length R consisting of only ones. The term $\frac{1}{2}\|[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\|_{\mathrm{F}}^2$ can be simplified in the same manner. The inner product $\left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] \right\rangle$ can similarly be written as

$$\left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] \right\rangle = \mathbf{1}^{\mathrm{T}}[(\mathbf{A}^{\mathrm{T}}\mathbf{X})*(\mathbf{B}^{\mathrm{T}}\mathbf{Y})*(\overline{\mathbf{C}}^{\mathrm{T}}\mathbf{Z})]\mathbf{1}. \quad (3.4)$$

Because $[\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!] = \mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^T$, both $[\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!]$ and $\mathbf{M}$ are matrices and their inner product is simply $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^T)*\mathbf{M}]\mathbf{1}$.

3.3.2 Gradient and Gramian

For an NLS algorithm, efficient evaluations of the gradient and Gramian of f are required [226]. Below, we give the derivation of the gradient $\mathbf{g} = \nabla f = \left[\mathrm{vec}\left(\frac{\partial f}{\partial \mathbf{A}}\right); \mathrm{vec}\left(\frac{\partial f}{\partial \mathbf{B}}\right); \mathrm{vec}\left(\frac{\partial f}{\partial \mathbf{C}}\right)\right]$. The expression for the Gramian is identical to the batch algorithm, as shown in Properties 2.3.2 and 2.3.3 in Chapter 2.

We derive the terms $\frac{\partial f}{\partial \mathbf{A}}$ and $\frac{\partial f}{\partial \mathbf{C}}$ of the gradient. The term $\frac{\partial f}{\partial \mathbf{B}}$ is obtained analogously to $\frac{\partial f}{\partial \mathbf{A}}$. Only the first, second and fourth terms of equation (3.2) are non-constant, so we compute their derivatives with respect to $\mathbf{A}$ and $\mathbf{C}$, using the expressions from the previous subsection. For $\frac{\partial f}{\partial \mathbf{A}}$, we find

$$\begin{aligned}\frac{\partial}{\partial \mathbf{A}}\frac{1}{2}\|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_{\mathrm{F}}^2 &= \mathbf{A}[(\mathbf{B}^{\mathrm{T}}\mathbf{B})*(\mathbf{C}^{\mathrm{T}}\mathbf{C})] \\ -\frac{\partial}{\partial \mathbf{A}}\left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] \right\rangle &= -\mathbf{X}[(\mathbf{Y}^{\mathrm{T}}\mathbf{B})*(\mathbf{Z}^{\mathrm{T}}\overline{\mathbf{C}})] \\ -\frac{\partial}{\partial \mathbf{A}}\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\rangle &= -\mathbf{M}\mathbf{B}\mathrm{diag}(\underline{\mathbf{c}}), \quad (3.5)\end{aligned}$$

leading to

$$\frac{\partial f}{\partial \mathbf{A}} = \mathbf{A}[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] - \mathbf{X}[(\mathbf{Y}^{\mathrm{T}}\mathbf{B}) * (\mathbf{Z}^{\mathrm{T}}\overline{\mathbf{C}})] - \mathbf{M}\mathbf{B}\mathrm{diag}(\underline{\mathbf{c}}).$$

Analogously, we have

$$\begin{aligned}\frac{\partial f}{\partial \mathbf{C}} &= \mathbf{C}[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] + \begin{bmatrix} -\frac{\partial f}{\partial \overline{\mathbf{C}}}\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!],[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\rangle \\ -\frac{\partial f}{\partial \underline{\mathbf{c}}}\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!],\mathbf{M}\rangle \end{bmatrix} \\ &= \mathbf{C}[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] + \begin{bmatrix} -\mathbf{Z}[(\mathbf{X}^{\mathrm{T}}\mathbf{A}) * (\mathbf{Y}^{\mathrm{T}}\mathbf{B})] \\ -\mathrm{vec}(\mathbf{M})^{\mathrm{T}}(\mathbf{B} \odot \mathbf{A}) \end{bmatrix},\end{aligned}$$

as $\frac{\partial f}{\partial \mathbf{C}}$ can be partitioned into $\begin{bmatrix} \frac{\partial f}{\partial \overline{\mathbf{C}}} \\ \frac{\partial f}{\partial \underline{\mathbf{c}}} \end{bmatrix}$ and the second and fourth term of Equation (3.2) do not depend on $\underline{\mathbf{c}}$ or $\overline{\mathbf{C}}$, respectively.

In the Gauss–Newton (GN) method, the Hessian of f is approximated by its Gramian $\mathbf{J}^T\mathbf{J}$, where $\mathbf{J}$ is the Jacobian of f. Using a limited number of conjugate gradients (CG) iterations, the speed of the algorithm is increased and explicit evaluation of the Gramian is avoided, as CG only needs the Gramian-vector product $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p}$, which can be obtained efficiently by exploiting the block-structure of the Gramian. Following Sorber et al. [183], if we write

$$\mathbf{p} = \mathrm{vec}(\begin{bmatrix}\mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3\end{bmatrix}),$$

then $(\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p})_{1,1}$, the contribution of the diagonal $1,1$-block of $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ in $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p}$, can be computed as $\mathrm{vec}(\mathbf{P}_1[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})])$. The contribution of the off-diagonal $1,2$-block of $\mathbf{J}^T\mathbf{J}$, $(\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p})_{1,2}$, can be computed as $\mathrm{vec}(\mathbf{A}[(\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{P}_2^{\mathrm{T}}\mathbf{B})]$. The contributions of the other diagonal and off-diagonal blocks can be obtained analogously. In practice, a block-Jacobi preconditioner [14] with diagonal blocks $[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \otimes \mathbf{I}_I$, $[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \otimes \mathbf{I}_J$ and $[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})] \otimes \mathbf{I}_K$, where $\mathbf{I}_n$ is the $n \times n$-identity matrix, is also applied to the system to improve the convergence speed.

3.3.3 Windowing and dynamic rank

Different weighting strategies can be followed to ensure that recent tensor slices influence the decomposition more than older ones. The use of exponential or truncated/rectangular windows are two popular windowing strategies [143, 149]. The first one uses a weighting matrix $\mathbf{L} = \mathrm{diag}([\lambda^k, \lambda^{k-1}, \ldots, \lambda, 1])$, so that every old slice is scaled down by a factor λ whenever a new slice arrives. The second one only considers the last M slices for the update and thus truncates the tensor by removing its outdated slices. Its weighting matrix looks as follows:

$\mathbf{L} = \text{diag}([0, \ldots, 0, 1, \ldots, 1])$, where $\mathbf{L}$ contains M ones. Both types can be combined to obtain a truncated exponential window with weighting matrix $\mathbf{L} = \text{diag}([0, 0, \ldots, 0, \lambda^{M-1}, \lambda^{M-2}, \ldots, \lambda, 1])$.

Windowing can easily be incorporated into the updating algorithm, by modifying the objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|[\![\mathbf{A}, \mathbf{B}, \mathbf{LC}]\!] - \mathcal{T}^{(\text{up})} \cdot_3 \mathbf{L}\|_{\text{F}}^2.$$

The factor matrices $\mathbf{C}$ and $\mathbf{Z}$ are thus replaced by $\mathbf{LC}$ and $\overline{\mathbf{L}}\mathbf{Z}$, respectively, where $\overline{\mathbf{L}}$ is the matrix $\mathbf{L}$ without its last row and column. If required, the factor matrix $\mathbf{C}$ can be recovered from $\mathbf{LC}$ by left multiplication with $\mathbf{L}^\dagger$, where $\mathbf{L}^\dagger$ is obtained by inverting the non-zero entries of $\mathbf{L}$. Otherwise, $\lambda\mathbf{LC}$ can directly be used for the initialization of the next updating step.

The updating method can easily be adapted to admit changes of the tensor rank, as in every update, the rank of the new CPD can be adjusted. To see this, note that the objective function (3.2) and the gradients do not change if $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$ has a different rank than $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$. Determining when the rank should change, is more difficult, however; see [242] for an extended discussion.

Algorithm 3.1: NLS updating for CPD

1: function $[\mathbf{A}, \mathbf{B}, \mathbf{C}] = \underline{\text{cpd_up}}([\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!], \mathbf{M}, \mathbf{L}, P, Q)$

2: **Input:** Old CPD $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]_R$, new slice $\mathbf{M}$, windowing matrix $\mathbf{L}$, max number of GN iterations P, max number of CG iterations Q
3: **Output:** Updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$
4: Solve $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\text{new}} = \text{vec}(\mathbf{M})$ using Equation (3.1)
5: Decide on rank R' of the updated CPD based on the error $\|(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\text{new}} - \text{vec}(\mathbf{M})\|_{\text{F}}$ (default $R' = R$)
6: Concatenate $\mathbf{c}_{\text{new}}$ to $\mathbf{Z}$ to obtain the initialization $\left[\!\!\left[\mathbf{X}, \mathbf{Y}, \mathbf{L}\begin{bmatrix}\mathbf{Z} \\ \mathbf{c}_{\text{new}}^{\text{T}}\end{bmatrix}\right]\!\!\right]_R$
7: Remove column or add random column to the initialization if $R' \neq R$
8: Solve the NLS-problem $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|[\![\mathbf{A}, \mathbf{B}, \mathbf{LC}]\!]_{R'} - \mathcal{T}^{(\text{up})} \cdot_3 \mathbf{L}\|_{\text{F}}^2$ with max P GN iterations and max Q preconditioned CG iterations per GN iteration, using the efficient evaluations in Section 3.3
9: Recover $\mathbf{C}$ from $\mathbf{C} = \mathbf{L}^\dagger(\mathbf{LC})$ or store $\mathbf{LC}$
10: Return the updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$

3.3.4 Complexity analysis

A dogleg trust-region Gauss-Newton method is used to compute the CPD updates. This method performs a maximum of P iterations wherein the optimization step $\mathbf{p}_p$ is computed by solving $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p = -\mathbf{g}_p$, where $\mathbf{J}$ is the Jacobian of f. The latter system is solved by preconditioned CG with Q iterations for which the complexity is dominated by the evaluation of $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$. Each update thus requires a maximum of P evaluations of f and $\mathbf{g}$ and PQ evaluations of $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$ plus an additional number P' of evaluations of f for the trust-region method. P' is typically equal to P.

For the objective function, it can be noted that $\frac{1}{2}\|[\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!]\|_{\mathrm{F}}^2$ and $\frac{1}{2}\|\mathbf{M}\|_{\mathrm{F}}^2$ are constant and only have to be computed once. Assuming $R' = R$, the terms $\frac{1}{2}\|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_{\mathrm{F}}^2$ and $\left\langle [\![\mathbf{A},\mathbf{B},\overline{\mathbf{C}}]\!], [\![\mathbf{X},\mathbf{Y},\mathbf{Z}]\!] \right\rangle$ can be computed in $\mathcal{O}\big(R^2\max(I,J,M)\big)$ flops, with M the length of the window, using the simplification of equation (3.3). The last term $\langle [\![\mathbf{A},\mathbf{B},\underline{\mathbf{c}}]\!], \mathbf{M}\rangle$ can be computed in $\mathcal{O}\big(IJ\big)$ flops, totaling at a complexity of $\mathcal{O}\big(2R^2\max(I,J,M) + IJ\big)$ flops.

The gradient $\mathbf{g}$ consists of three terms, $\frac{\partial f}{\partial \mathbf{A}}$, $\frac{\partial f}{\partial \mathbf{B}}$ and $\frac{\partial f}{\partial \mathbf{C}}$. They can all be computed in $\mathcal{O}\big(2R^2\max(I,J,M) + IJR\big)$ flops, totaling at $\mathcal{O}\big(6R^2\max(I,J,M) + 3IJR\big)$ flops.

The Gramian-vector product $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p$ requires $\mathcal{O}\big(3R^2\max(I,J,M)\big)$ flops per CG iteration [183] and thus $\mathcal{O}\big(3QR^2\max(I,J,M)\big)$ flops per GN iteration. Preconditioning adds three $R \times R$ matrix inversions and Q matrix-vector products per GN iteration, totaling $\mathcal{O}\big(3R^3 + QR(I+J+M)\big)$ flops.

Summing these values for P iterations of the method and adding the P' evaluations of f for the trust-region method, leads to a total time complexity of $\mathcal{O}\big((8P+3QP+2P')R^2\max(I,J,M) + (3RP+P+P')IJ + 3PR^3 + PQR(I+J+M)\big)$, which for low rank tensors and a truncated window is dominated by the term $3RPIJ$. Note that this term only depends on the dimensions of the new slice $\mathbf{M}$ and not on the window length M.

The memory consumption of the proposed method is dominated by the storage of the old and new CPD, which is $\mathcal{O}\big(R(I+J+M)\big)$ and the new tensor slice, which is $\mathcal{O}\big(IJ\big)$. The gradients and Gramian-vector products that are used during the execution of the method both require $\mathcal{O}\big(R(I+J+M)\big)$ memory as well. In contrast, storing the full (windowed) tensor would require $\mathcal{O}\big(IJM\big)$ memory.

3.4 Experiments

We compare the proposed method with batch algorithms for the CPD and with the PARAFAC-SDT and PARAFAC-RLST methods of Nion et al. [149]. All computations are done in Tensorlab [228]. The batch algorithms simply compute the CPD of the tensor formed by the slices in the window. These algorithms are a nonlinear least squares (NLS) algorithm, called `cpd_nls`, and an alternating least squares (ALS) algorithm, called `cpd_als`, both available in Tensorlab. In every step, they are initialized with the decomposition from the previous updating step, with the aforementioned least squares solution $\mathbf{c}_{\text{new}} = (\mathbf{Y} \odot \mathbf{X})^{\dagger}\text{vec}(\mathbf{M})$, corresponding to the new slice, concatenated to the third factor matrix. All optimization methods are limited to $P = 1$ iterations, while $Q = 5$ CG iterations are allowed for the linear systems that are solved during the algorithms. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0.

Initially, a rank-R tensor of dimensions $1000 \times 1000 \times 100$ is generated by sampling two factor matrices from the uniform distribution on $[0, 1]$. The third factor matrix is sampled along a third-degree polynomial with coefficients from the standard normal distribution to obtain a model that varies slowly along its third mode. The different experiments have $R = 2, 4$ and 6, respectively. The tensor is perturbed by zero-mean Gaussian noise so that the signal-to-noise ratio (SNR) is 20 dB. First, a full CPD is computed for the first fifty mode-3 slices of the tensor. The other slices are then added one by one to the tensor, after which the decomposition is updated. A truncated exponential window of length $M = 30$ with forgetting factor $\lambda = 0.9$ is applied. Reported values are medians across ten runs.

In Table 3.1, the median required CPU-time to compute an update using five different methods is shown. The five methods are the batch NLS and batch ALS algorithms using only slices from the truncated exponential window, i.e., using only the last M slices of the tensor, and PARAFAC-SDT, PARAFAC-RLST and the proposed updating method using the same window. For these large tensors, the updating method achieves about the same speed as PARAFAC-SDT and both are a factor 10 to 50 faster than the batch methods.

In Table 3.2, the median fitting errors over the fifty updating steps are shown. The error is here defined as the average error for all tensor entries, where the slices are weighted using the windowing matrix, i.e., the errors of older slices have a smaller weight in the mean than those of newer slices. The accuracy of the updating method is equal to that of the batch methods, while PARAFAC-SDT and PARAFAC-RLST perform a lot worse, especially for larger values of R. As

Table 3.1: Medians of the CPU-time (in ms) for a single update using the new updating method, NLS and ALS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods.

R	2	3	4	5	6
Update 1 P=1,Q=5	27	24	27	28	30
NLS	1355	1800	4814	7662	15980
ALS	891	94	1117	1299	1758
SDT	59	86	98	145	179
RLST	835	859	863	940	968
Update 2 P=5,Q=25	48	80	121	165	177

only one iteration is performed by the updating method during the experiments ($P = 1$ and $Q = 5$), higher accuracy can easily be traded for longer execution times by increasing P and/or Q for more difficult experiments. Results for $P = 5$ and $Q = 25$ are also included in Tables 3.1 and 3.2. It can be seen that the execution time rises linearly with the number of performed iterations. In Figure 3.2, the errors of the different methods are plotted for the case $R = 6$ and SNR $= 50$ dB. The updating method achieves the same results as the batch methods. The error also remains relatively constant over the fifty updates, in contrast to PARAFAC-SDT, for which the errors accumulate.

Summarizing, the error of the updating method is equal to that of the batch methods and it is significantly faster. As only the old CPD and the new slice have to be stored, the memory cost is lower compared to the batch methods, which have to store the last M tensor slices. For tensors with millions of entries or time-sensitive applications, this can make an important difference in the applicability of tensor methods. Although PARAFAC-SDT has the same speed as the updating method for large tensors, it consistently yielded a lower accuracy.

3.5 Conclusion

An NLS updating method was proposed for the CPD that exploits its structure to execute a fast NLS update whenever a new slice arrives. The batch NLS algorithm for the CPD is adapted so that it can be used in an updating context.

Table 3.2: Weighted mean errors for the new updating method, NLS and ALS batch algorithms and PARAFAC-SDT and PARAFAC-RLST. Medians over the fifty updating steps.

R	2	4	6
Update 1 $P{=}1,Q{=}5$	$8.92 \cdot 10^{-3}$	$6.20 \cdot 10^{-3}$	$8.17 \cdot 10^{-3}$
NLS	$8.92 \cdot 10^{-3}$	$6.20 \cdot 10^{-3}$	$8.17 \cdot 10^{-3}$
ALS	$8.92 \cdot 10^{-3}$	$6.20 \cdot 10^{-3}$	$8.17 \cdot 10^{-3}$
SDT	$2.99 \cdot 10^{-2}$	$3.87 \cdot 10^{-2}$	$9.46 \cdot 10^{-2}$
RLST	$9.14 \cdot 10^{-3}$	$2.70 \cdot 10^{-1}$	$6.58 \cdot 10^{0}$
Update 2 $P{=}5,Q{=}25$	$8.92 \cdot 10^{-3}$	$6.20 \cdot 10^{-3}$	$8.17 \cdot 10^{-3}$

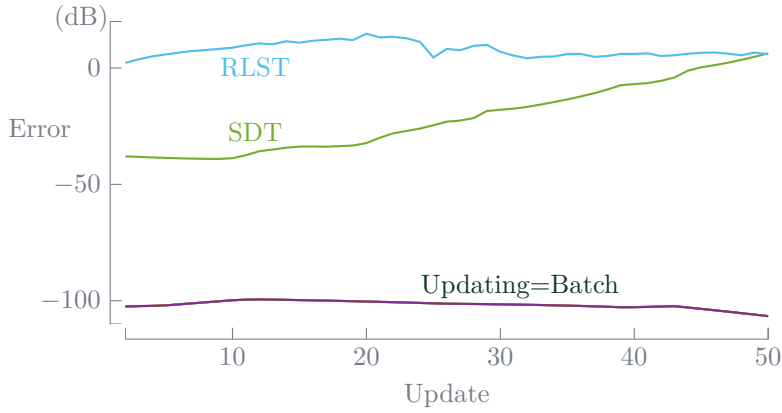

Figure 3.2: Weighted mean errors of the new updating method, ALS and NLS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods for $R = 6$ and SNR $= 50$ dB on a decibel scale.

By only using the previous decomposition and the new tensor slice when it arrives, the updating method becomes both time- and memory-efficient, while maintaining a good accuracy for the decomposition. Efficient expressions are derived for the computation of the objective function, gradient and Gramian. It is also shown that arbitrary windowing strategies and changes of the tensor rank can be handled straightforwardly. Finally, the performance of the method is demonstrated on a large-scale tensor. The algorithm is faster than batch ALS and NLS algorithms for the numerical experiments, while maintaining good

accuracy, especially compared to PARAFAC-SDT and PARAFAC-RLST. As only the new slice and the old factor matrices are needed in the computation of the update, updating is very memory efficient, making it applicable for large-scale problems. A possible drawback of this memory-efficient approach is that small latent trends in the data may be ignored during multiple consecutive updates as these trends are dominated by the current model in every step. This could be mitigated by tracking some additional information, e.g., the previous (few) slice(s) or an extra rank-1 term.

3.A Other updating schemes

The proposed CPD updating method is introduced in the context of slice updates. The core idea, however, can be applied to a much wider range of problems. The only requirement is that the new data affects relatively few entries compared to the number of entries in the full tensor. Together with the exploitation of the CPD structure of the information from the previous update, this can lead to significant speed gains, as shown for the case of slice updates. Other examples of such updating formats include fiber updates, Nth-order subtensor updates and sparse updates.

Below, we generalize the slice updating scheme from Section 3.3. We start from the same optimization problem as before:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}^{(\text{up})}\|_{\text{F}}^2, \tag{3.6}$$

and show that an efficient algorithm can be obtained for a wide class of updating schemes. As for slice updates, rank changes can be introduced straightforwardly and windowing techniques can be applied using an appropriate weight tensor; see also [210]. The complexity analysis in Section 3.3.4 can be extended to the general framework. Naturally, the operation count depends on the number of entries that are affected by the new updates. In general, however, the updating method can be expected to be significantly less costly than recomputing the full CPD using a batch method.

Below, we derive expressions for the objective function, gradient and Gramian of the general updating method. We also apply the method to update the CPD of a tensor that gradually gets filled in using sparse updates.

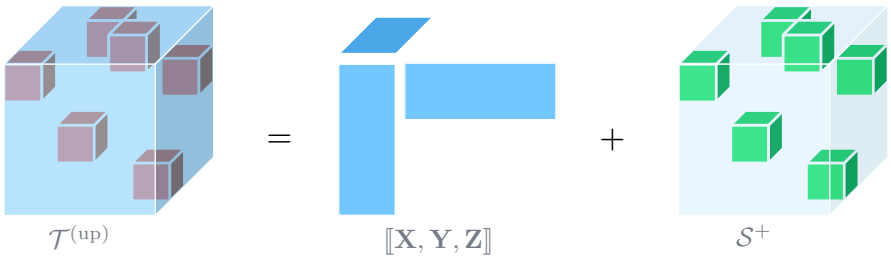

Figure 3.3: The tensor $\mathcal{T}^{(\mathrm{up})}$ is decomposed as $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] + \mathcal{S}^{+}$. Left: new values (red) become available for some entries of the original tensor. Right: decomposition into a sum of the old CPD and a tensor that is zero except for additive updates (green) for the affected entries.

3.A.1 Objective function

The Frobenius norm in (3.6) can be expanded into three terms, giving

$$f = \frac{1}{2}\|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\mathrm{F}}^2 - \left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], \mathcal{T}^{(\mathrm{up})} \right\rangle + \frac{1}{2}\|\mathcal{T}^{(\mathrm{up})}\|_{\mathrm{F}}^2. \tag{3.7}$$

For the first term, $\|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\mathrm{F}}^2$ can be efficiently computed as $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})]\mathbf{1}$ following (3.3).

The computation of the second and third terms of (3.7) is more involved. We never store the full tensor $\mathcal{T}^{(\mathrm{up})}$ during the updating process, so we only have the CPD $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$ of $\mathcal{T}$, computed in the previous update, and some new values for certain tensor entries. We denote the tensor holding the net additive entry updates by $\mathcal{S}^{+}$, i.e, $\mathcal{S}^{+}$ holds the new value minus the old value for each affected tensor entry. These updates can, for instance, be modifications to the data in a single slice, a few fibers, a small subtensor or a set of entries scattered throughout the tensor. The other entries of $\mathcal{S}^{+}$ are set to zero. We can now decompose $\mathcal{T}^{(\mathrm{up})}$ as follows:

$$\mathcal{T}^{(\mathrm{up})} = [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] + \mathcal{S}^{+}.$$

This decomposition is shown schematically in Figure 3.3 for a sparse updating scheme. For (mode-3) slice updates, we assume that a zero row has been added to the bottom of $\mathbf{Z}$ to represent the new slice, i.e., we assume that the previous CPD estimate of the new slice consists of only zeros.

For the second term of (3.7), we then find

$$\left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], \mathcal{T}^{(\mathrm{up})} \right\rangle = \langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \rangle + \left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], \mathcal{S}^{+} \right\rangle,$$

The entries of $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ corresponding to unchanged entries of $\mathcal{T}^{(\text{up})}$ do not play a role in the second inner product because of the zeros in $\mathcal{S}^+$, so we can replace $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ by $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+$, with the $^+$ signaling that the entries on the non-updated positions are set to zero. Of course, both $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+$ depend on the same optimization variables (the entries of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$). This gives us:

$$\left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], \mathcal{T}^{(\text{up})} \right\rangle = \langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \rangle + \left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+, \mathcal{S}^+ \right\rangle.$$

As before, the first term can be efficiently computed as $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}^{\mathrm{T}}\mathbf{X}) * (\mathbf{B}^{\mathrm{T}}\mathbf{Y}) * (\mathbf{C}^{\mathrm{T}}\mathbf{Z})]\mathbf{1}$, using (3.4). The second term requires the computation of the values in $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+$ from the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$. However, for the updates that we consider, the nonzero parts of both $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+$ and $\mathcal{S}^+$ are small relative to the size of the full tensor $\mathcal{T}^{(\text{up})}$: they have the dimensions of a single tensor slice, a fiber, a small subtensor, can be represented as a sparse tensor, etc. So, both terms can be computed efficiently, even if the CPDs needs to be partially evaluated.

For the third term of (3.7), $\mathcal{T}^{(\text{up})}$ can again be written as $\mathcal{T}^{(\text{up})} = [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] + \mathcal{S}^+$:

$$\begin{aligned} \|\mathcal{T}^{(\text{up})}\|_{\text{F}}^2 &= \|[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]\|_{\text{F}}^2 + 2\left\langle [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!], \mathcal{S}^+ \right\rangle + \|\mathcal{S}^+\|_{\text{F}}^2 \\ &= \|[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]\|_{\text{F}}^2 + 2\left\langle [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]^+, \mathcal{S}^+ \right\rangle + \|\mathcal{S}^+\|_{\text{F}}^2. \end{aligned}$$

As before, the first term can be efficiently computed following (3.3) and the number of nonzero entries in the second and third terms is small compared to the full tensor $\mathcal{T}^{(\text{up})}$.

Putting all three terms of the objective function f in (3.7) together yields:

$$\begin{aligned} f = \frac{1}{2}\|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\text{F}}^2 - \langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \rangle - \left\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^+, \mathcal{S}^+ \right\rangle \\ + \frac{1}{2}\|[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]\|_{\text{F}}^2 + \left\langle [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]^+, \mathcal{S}^+ \right\rangle + \frac{1}{2}\|\mathcal{S}^+\|_{\text{F}}^2. \end{aligned} \tag{3.8}$$

Looking back at the objective function for slice updates, one can see that (3.8) is indeed a generalization of (3.2). For slice updates, the nonzero part of the tensor $\mathcal{S}^+$ is equal to $\mathbf{M}$ and the fifth term in (3.8) is equal to zero, i.e., $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]^+$ is zero because the new slice was not modeled by the previous CPD $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$.

3.A.2 Gradient and Gramian

For the gradient, we need to compute derivatives of f with respect to $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. The last three terms in (3.8) are constant in $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, so they do not

appear in the gradient. For $\frac{\partial f}{\partial \mathbf{A}}$, the first two terms can be evaluated as in (3.5):

$$\frac{\partial}{\partial \mathbf{A}} \frac{1}{2} \|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_{\mathrm{F}}^2 = \mathbf{A}[(\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})]$$

$$-\frac{\partial}{\partial \mathbf{A}} \langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \rangle = -\mathbf{X}[(\mathbf{Y}^{\mathrm{T}}\mathbf{B}) * (\mathbf{Z}^{\mathrm{T}}\mathbf{C})].$$

The derivative of the third term depends strongly on the specific structure of the updated tensor entries. If slice updates are performed, the expression in (3.5) can be used. For other structures, alternative efficient expressions can be derived. For sparse updates, for instance, the derivative can be computed by adding the contribution of each new entry individually. The objective function (3.8) is symmetric in $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, so similar expressions can be derived for the derivatives with respect to $\mathbf{B}$ and $\mathbf{C}$.

As for the case of slice updates, the Jacobian of f is equal to the Jacobian of the batch algorithm, i.e., the structure the tensor that we approximate has no influence on the Jacobian. Consequently, the Gramian $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ that is used as approximation for the Hessian matrix in the Gauss–Newton algorithm, is identical to the Gramian of the batch algorithm and its structure can be exploited in the solution of the linear system $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p} = -\mathbf{g}$, see Section 3.3.2.

3.A.3 Fill-in experiment

In the following experiment, we compute an initial CPD of an incomplete tensor and gradually refine the CPD by sampling new tensor entries. The full tensor is generated by first constructing a rank-5 tensor from factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{50\times 5}$ sampled from the uniform distribution on $[0, 1]$ and then adding Gaussian distributed noise with an SNR of 20 dB. The initial CPD is computed by uniformly sampling 8 000 values (out of 125 000) from the tensor and computing a CPD using the `cpdi_nls` [225] algorithm from Tensorlab [228]. This is a specialized Gauss–Newton algorithm for incomplete tensors that takes only known entries into account in the CPD computation. In every update, we sample 50 new values and refine the CPD in three different ways: using the proposed updating method, using the batch `cpdi_nls` algorithm and using the standard batch Gauss–Newton CPD algorithm `cpd_nls` from Tensorlab. All methods take their decomposition from the previous update as initialization. We consider a setting where the tensor entries gradually become available and do not change once they are collected. Therefore, we do not allow the same entries to be sampled multiple times in different updates. Of course, the proposed method can also be applied to run the same experiment while allowing entries to be sampled several times (with slightly different values) in different updates.

We define the relative factor matrix to assess the accuracy of a CPD.

Definition 3.A.1: Relative factor matrix error
The relative factor matrix error between the true factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ and the estimated factor matrices from the CPD $\left[\!\left[\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}\right]\!\right]$ is equal to the vector

$$\left(\frac{\|\mathbf{A} - \hat{\mathbf{A}}\|_{\mathrm{F}}}{\|\mathbf{A}\|_{\mathrm{F}}}, \frac{\|\mathbf{B} - \hat{\mathbf{B}}\|_{\mathrm{F}}}{\|\mathbf{B}\|_{\mathrm{F}}}, \frac{\|\mathbf{C} - \hat{\mathbf{C}}\|_{\mathrm{F}}}{\|\mathbf{C}\|_{\mathrm{F}}} \right).$$

In Figure 3.4, the maximal relative factor matrix error is shown between the true factor matrices and the estimated factor matrices in function of the updating step. The fraction of sampled entries is also shown. It can be seen that the error of the updating method shows a steady decrease as more entries are sampled. The `cpdi_nls` and `cpd_nls` algorithms always obtain the same error. Initially, their error decreases faster than that of the updating method, but as the new samples come in, the progress stagnates and the updating method catches up. In Figure 3.5 the time per update is shown for all three methods. The updating method is clearly the fastest and its computation time remains constant over the updates. The `cpdi_nls` method starts out relatively fast, but gradually becomes slower as more entries are sampled. In the end, the updating method is more than ten times faster per update. The `cpd_nls` method starts out slow, when few values of the tensor are available. As the sampled tensor approaches the full tensor, however, the `cpd_nls` method speeds up until it is only about two times as slow per update as the updating method.

3.A.4 Conclusion

We showed that the CPD slice updating method can be generalized so that it supports a whole class of updating schemes. We derived expressions for the objective function and gradient and showed that the properties of the slice updating method still hold in the general case. We demonstrated the performance of the updating method for sparse updates in an experiment on synthetic data: in each update, a number of tensor entries are sampled randomly and these are used to refine the CPD of the tensor. We showed that the error of the CPD gradually becomes smaller as more entries are sampled and that the accuracy of the updating method is competitive with batch methods, while being significantly faster.

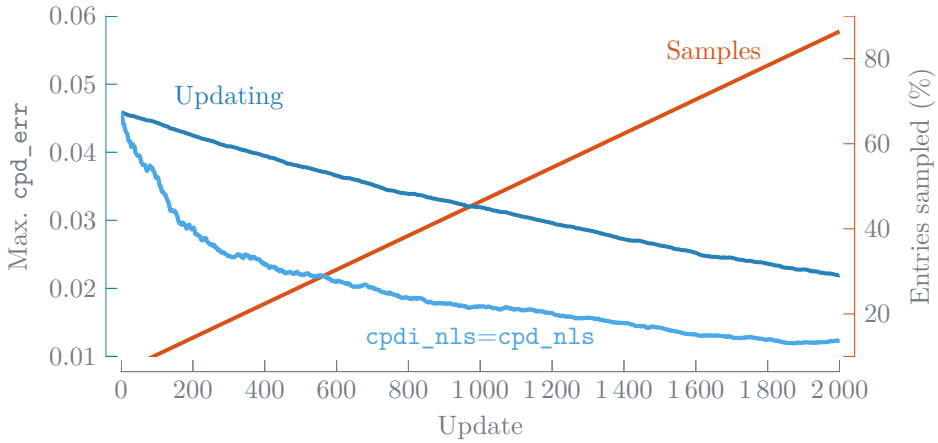

Figure 3.4: The relative factor matrix error of the updating method gradually decreases as more entries are sampled. Initially, the error for `cpdi_nls` and `cpd_nls` drops faster, but as more entries are sampled the error of the updating method catches up. The fraction of entries that have been sampled is shown on the right y-axis.

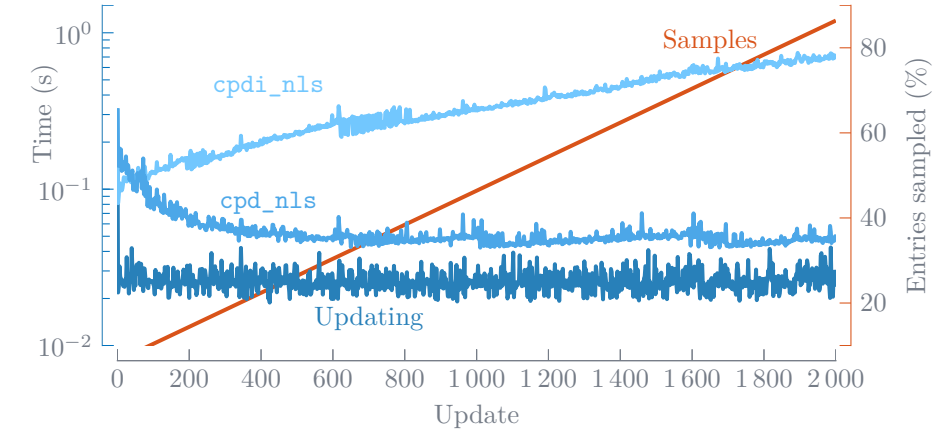

Figure 3.5: The updating method is significantly faster than both `cpdi_nls` and `cpd_nls` and its computation time remains constant over the updates. In contrast, `cpdi_nls` is relatively fast in the beginning and gradually becomes slower, while `cpd_nls` is slow initially and gradually becomes faster as the incomplete tensor approaches a full tensor. The fraction of entries that have been sampled is shown on the right y-axis.

4 CPD updating using low-rank weights

ABSTRACT

Tensor updating methods enable tensor decompositions to adapt quickly when new data is added to the tensor. At present, updating methods for the canonical polyadic decomposition (CPD) give every tensor entry the same weight. In practice, however, data quality or relative importance might differ between tensor entries, which warrants the use of more general weighting schemes. In this chapter, an NLS updating method is developed for the CPD that uses a weighted least squares (WLS) approach with a low-rank weight tensor. This weight tensor itself can also be updated to allow dynamic weighting schemes. By exploiting the CPD structure of both the data and weight tensors, the algorithm obtains better accuracy than the unweighted updating methods, while being more time- and memory efficient than batch WLS methods.

This chapter is based on M. Vandecappelle, M. Boussé, N. Vervliet, and L. De Lathauwer, "CPD updating using low-rank weights," in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018, Calgary, Alberta, Canada), 4 2018, pp. 6368–6372. The figures have been updated for consistency. The specific contributions of the doctoral candidate are listed in Section 11.1.

4.1 Introduction

The popularity of tensor methods has been rising steadily in the past years and they have been applied in a wide variety of fields such as machine learning and signal processing [35, 174]. Tensors are higher-order extensions of vectors (first-order) and matrices (second-order). Tensor decompositions allow the compact storage of large tensors, enabling both compression and analysis of big, higher-order datasets. Apart from the classical alternating least squares (ALS) algorithms, several algebraic and all-at-once optimization-based algorithms for the computation of tensor decompositions have been developed: See, for example, [62, 154, 157, 183, 184, 222, 227] and references therein.

Signals are often perturbed by noise. If prior information about the noise is available, this information can be leveraged in the computation of tensor decompositions such as the canonical polyadic decomposition (CPD) to improve the quality of the model. Different tensor entries can be weighted differently in the least squares (LS) cost function. The weights may take into account the quality of the data of the corresponding tensor entries. As an example, consider an array of sensors with varying signal-to-noise ratios (SNR). In this case, it makes sense to weight entries according to the quality of the sensor they originate from, giving more weight to data from the better sensors. Weighted LS (WLS) algorithms exist that admit general weight tensors [104, 152], and extend the CPD cost function from Equation 2.2 to

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|\mathcal{W} * (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})\|_{\mathrm{F}}^2,$$

wherein $\mathcal{W}$ is a weight tensor of the same dimensions as $\mathcal{T}$. When time is limited, it can be beneficial to use a low-rank weight tensor instead of a full weight tensor $\mathcal{W}$. Writing $\llbracket \mathbf{N}, \mathbf{P}, \mathbf{Q} \rrbracket$ for the CPD of $\mathcal{W}$, the cost function becomes

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|\llbracket \mathbf{N}, \mathbf{P}, \mathbf{Q} \rrbracket * (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})\|_{\mathrm{F}}^2.$$

In the evaluation of this cost function and its derivatives, low-rank structure can be exploited to make the decomposition less demanding, both in storage and computation, compared to using a full weight tensor [16]. Also, approximating the weight tensor is usually good enough, as a high accuracy of the weights is generally unnecessary or even unobtainable. Note that the standard CPD methods can be seen as WLS methods with a specific rank-1 weight tensor (consisting of only ones).

If new data is added to a tensor at regular time-intervals, its storage can quickly become problematic. In contrast, the CPD of this tensor needs only

a fraction of the memory that the full tensor requires, while the important features of the data are maintained. Efficient updating methods for the CPD have been developed in Chapter 3 and in [149, 199], but by combining CPD updating with a low-rank WLS method into a weighted CPD updating method, we can get the best of two worlds. Returning to the sensor array example, it would be useful to leverage the prior information about the sensors in the CPD updating process by incorporating a low-rank weight tensor. Even stronger, one can use information about the current state of the sensors to update the low-rank weight tensor itself, e.g., some types of sensors might have warm-up times or may be influenced by external conditions that affect their performance. Therefore, we develop a weighted nonlinear LS (NLS) CPD updating algorithm that exploits the low-rank structure of both the weight and data tensors, as described in [224, 226]. This makes the algorithm computation- and memory-efficient, properties that are primordial if the method is to be applied in an updating context. For the implementation, we use Tensorlab [228], a MATLAB toolbox for tensor computations.

We fix notation, basic definitions and useful identities in the remainder of this section. Our method is derived in Section 4.2. We illustrate the algorithm on a practical example in Section 4.3.

4.1.1 Notation, definitions and identities

Scalars, vectors, matrices and tensors are denoted by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase letters ($\mathbf{A}$), and letters in calligraphic script ($\mathcal{T}$), respectively. The kth frontal slice $\mathbf{T}_{::k}$ of a tensor is a matrix obtained by fixing the third index of the tensor. A mode-n unfolding of a tensor $\mathcal{T}$ is a matrix $\mathbf{T}_{(n)}$ with the mode-n vectors as its columns following the ordering in [119]. The vectorization operator is denoted by vec$(\cdot)$ and $[\mathbf{a}^{\mathrm{T}}; \mathbf{b}^{\mathrm{T}}]$ refers to row-wise concatenation. Kronecker, Hadamard, column- and row-wise Khatri–Rao products of matrices are written as $\otimes$, $*$, $\odot$, and $\odot^{\mathrm{T}}$, respectively. The Moore–Penrose pseudoinverse of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{\dagger}$. A tensor has rank 1 if it is the outer product of three non-zero vectors. The tensor rank is the minimal number of rank-1 tensors needed to write the tensor as their linear combination. The derivations will focus on the case of third-order real tensors. The CPD of a tensor decomposes a rank-R tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ as a linear combination of R rank-1 terms: $\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_R$, with $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_R] \in \mathbb{R}^{I \times R}$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. The mode-3 unfolding is given by $\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}$. Useful identities are:

$$[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] * [\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] = [\![\mathbf{A} \odot^{\mathrm{T}} \mathbf{D}, \mathbf{B} \odot^{\mathrm{T}} \mathbf{E}, \mathbf{C} \odot^{\mathrm{T}} \mathbf{F}]\!] \ , \tag{4.1}$$

and

$$\langle [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] , [\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] \rangle = \mathbf{1}^{\mathrm{T}} [(\mathbf{D}^{\mathrm{T}} \mathbf{A}) * (\mathbf{E}^{\mathrm{T}} \mathbf{B}) * (\mathbf{F}^{\mathrm{T}} \mathbf{C})] \mathbf{1}. \tag{4.2}$$

4.2 NLS CPD updating using low-rank weights

Assume that one has computed the rank-R CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]_R$ of a third-order tensor $\overline{\mathcal{T}} \in \mathbb{R}^{I \times J \times K}$ using a WLS approach with low-rank weight tensor $\overline{\mathcal{W}} = [\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!]_L$. Then, let frontal slices $\mathbf{M}$ and $\mathbf{V}$ be added to both $\overline{\mathcal{T}}$ and $\overline{\mathcal{W}}$ in the third mode, as shown in Figure 4.1, obtaining the tensors $\mathcal{T}$ and $\mathcal{W}$, respectively. First, we want to update the CPD of the weight tensor to $[\![\mathbf{N}, \mathbf{P}, \mathbf{Q}]\!]_L$, where a new row $\underline{\mathbf{q}}$ is appended to $\overline{\mathbf{Q}}$ to include its new slice: $\mathbf{Q} = [\, \overline{\mathbf{Q}} \,;\, \underline{\mathbf{q}} \,]$. This can be done using the CPD updating method described in Chapter 3. Next, we would like to use this updated weight tensor to update the CPD of $\overline{\mathcal{T}}$ to a CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*}$ of $\mathcal{T}$ using a WLS approach with weight tensor $\mathcal{W}$, where R^* can be different from R.

The algorithm builds upon a dogleg trust-region Gauss-Newton (GN) algorithm, but the expressions are valid for more general NLS and quasi-Newton (QN) algorithms. The GN algorithm solves the following minimization problem:

$$\min_{\mathbf{s}_k} \frac{1}{2} \| \mathrm{vec}(\mathcal{F}_k) + \mathbf{J}_k \mathbf{s}_k \|_{\mathrm{F}}^2 \quad \text{such that} \quad \| \mathbf{s}_k \| \leq \Delta_k,$$

where $\mathcal{F}_k = \mathcal{W} * ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T})$ is the residual, and $\mathbf{s}_k$, $\mathbf{J}_k$ and Δ_k are the step, Jacobian and trust radius in iteration k, respectively. To increase efficiency, preconditioned conjugate gradient (PCG) iterations are used to solve the linear system $\mathbf{J}_k^{\mathrm{T}} \mathbf{J}_k \mathbf{s}_k = -\nabla f_k$ for the step $\mathbf{s}_k$, where ∇ is the gradient operator. Efficient expressions for the objective function, gradient and Gramian-vector products can be derived for the GN algorithm with PCG by exploiting the CPD structure of $\mathcal{W}$ and $\mathcal{T}$. We use a similar approach as that in Chapter 3 to obtain these expressions. They remain valid when the rank of the new CPD differs from that of the old CPD, so that the algorithm enables rank-changes between updates. The use of a block-Jacobi preconditioner and an initialization strategy are also discussed. The section ends with an analysis of the complexity of the algorithm. The full algorithm is given in Algorithm 4.1.

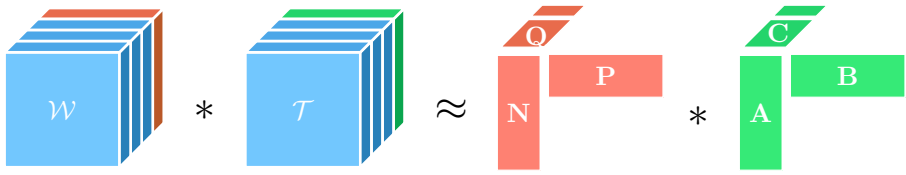

Figure 4.1: WLS updating using low-rank weights enables efficient CPD updates by exploiting the low-rank structure of *both* the weight and data tensor. Left: the weight ($\mathcal{W}$) and the data ($\mathcal{T}$) tensor are extended with an extra slice (red and green, respectively) in the third mode. Right: first, the CPD of the weight tensor is updated by adding a new vector to the factor matrix in the third mode and modifying the existing factor matrices to obtain the CPD $[\![\mathbf{N}, \mathbf{P}, \mathbf{Q}]\!]$. This updated low-rank weight tensor is then used to compute the updated CPD model $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the data in a similar way, this time using a WLS approach.

Algorithm 4.1: WLS CPD updating

1: **Input:** CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]_R$ of old tensor $\overline{\mathcal{T}}$, new slice $\mathbf{M}$, CPD $[\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!]_L$ of old weighting tensor $\overline{\mathcal{W}}$, new weighting slice $\mathbf{V}$, number of GN iterations it_{GN}, number of CG iterations it_{CG}, new rank R^*.
2: **Output :** CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*}$ of $\mathcal{T}$, CPD of $\mathcal{W}$.
3: Compute CPD of $\mathcal{W}$ using (unweighted) CPD updating.
4: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ using (4.3).
5: Solve the NLS-problem $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|\mathcal{W} * ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]_{R^*} - \mathcal{T})\|_{\text{F}}^2$ with it_{GN} GN iterations and it_{CG} preconditioned CG iterations per GN iteration, using the efficient evaluations in Section 4.2.
6: Return the updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.

4.2.1 Objective function

The WLS objective function for the computation of the CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of the updated low-rank tensor $\mathcal{T}$ using the updated weight tensor $\mathcal{W}$ is as follows:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|\mathcal{W} * ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T})\|_{\text{F}}^2 .$$

By splitting $\mathbf{C}$ as $[\,\overline{\mathbf{C}}\,;\,\underline{\mathbf{c}}\,]$, with $\underline{\mathbf{c}}$ the last row of $\mathbf{C}$, and similarly for the factor matrix $\mathbf{Q}$ of $\mathcal{W}$, f can be expanded as

$$f = \frac{1}{2} \left\| [\![\mathbf{N}, \mathbf{P}, \overline{\mathbf{Q}}]\!] * \left([\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!] - \overline{\mathcal{T}}\right) \right\|_{\text{F}}^2 + \frac{1}{2} \left\| [\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * \left([\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!] - \mathbf{M}\right) \right\|_{\text{F}}^2 .$$

To limit memory usage, the tensor $\overline{\mathcal{T}}$ is never stored during the updating process. Its CPD approximation $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]$ is used instead. Thus, using (4.1), the objective function becomes

$$f \approx \frac{1}{2} \left\| \left[\!\!\left[\mathbf{A}', \mathbf{B}', \overline{\mathbf{C}}' \right]\!\!\right] - [\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}}']\!] \right\|_{\mathrm{F}}^2 + \frac{1}{2} \left\| [\![\mathbf{A}', \mathbf{B}', \underline{\mathbf{c}}']\!] - [\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * \mathbf{M} \right\|_{\mathrm{F}}^2,$$

where $\mathbf{A}' = \mathbf{N} \odot^{\mathrm{T}} \mathbf{A}$, $\mathbf{B}' = \mathbf{P} \odot^{\mathrm{T}} \mathbf{B}$, $\overline{\mathbf{C}}' = \overline{\mathbf{Q}} \odot^{\mathrm{T}} \overline{\mathbf{C}}$, $\underline{\mathbf{c}}' = \underline{\mathbf{q}} \odot^{\mathrm{T}} \underline{\mathbf{c}}$ etc. We can expand f to obtain

$$f \approx \frac{1}{2} \|[\![\mathbf{A}', \mathbf{B}', \mathbf{C}']\!]\|_{\mathrm{F}}^2 - \left\langle \left[\!\!\left[\mathbf{A}', \mathbf{B}', \overline{\mathbf{C}}' \right]\!\!\right], [\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}}']\!] \right\rangle + \frac{1}{2} \left\| [\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}}']\!] \right\|_{\mathrm{F}}^2 - \langle [\![\mathbf{A}', \mathbf{B}', \underline{\mathbf{c}}']\!], \mathbf{M}' \rangle + \frac{1}{2} \|\mathbf{M}'\|_{\mathrm{F}}^2.$$

Here, we used that

$$\frac{1}{2} \left\| \left[\!\!\left[\mathbf{A}', \mathbf{B}', \overline{\mathbf{C}}' \right]\!\!\right] \right\|_{\mathrm{F}}^2 + \frac{1}{2} \|[\![\mathbf{A}', \mathbf{B}', \underline{\mathbf{c}}']\!]\|_{\mathrm{F}}^2 = \frac{1}{2} \|[\![\mathbf{A}', \mathbf{B}', \mathbf{C}']\!]\|_{\mathrm{F}}^2,$$

and defined the weighted new slice $\mathbf{M}'$ as $[\![\mathbf{N}, \mathbf{P}, \underline{\mathbf{q}}]\!] * \mathbf{M}$. The CPD structure of f can be exploited to obtain expressions that admit efficient computations. We use (4.2) to simplify the first two terms to: $\mathbf{1}^{\mathrm{T}}[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]\mathbf{1}$ and $\mathbf{1}^{\mathrm{T}}[(\mathbf{X}'^{\mathrm{T}}\mathbf{X}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{Y}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{Z}}')]\mathbf{1}$, respectively, and the third term to $\mathbf{1}^{\mathrm{T}}[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{B}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}}')]\mathbf{1}$. The last two terms are simply inner products of matrices.

4.2.2 Gradient

The gradient $\nabla f = \left[\mathrm{vec}\left(\frac{\partial f}{\partial \mathbf{A}}\right); \mathrm{vec}\left(\frac{\partial f}{\partial \mathbf{B}}\right); \mathrm{vec}\left(\frac{\partial f}{\partial \overline{\mathbf{C}}}\right)\right]$, which we denote by $[\mathbf{g}_A; \mathbf{g}_B; \mathbf{g}_C]$ has the following terms:

$$\mathbf{g}_A = \left(\mathbf{M}^{\mathbf{A}}\right)^{\mathrm{T}} \mathrm{vec}\left[\mathbf{A}'[(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')] - \mathbf{X}'[(\mathbf{Y}'^{\mathrm{T}}\mathbf{B}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}})] - \mathbf{M}'\mathbf{B}'\mathrm{diag}(\underline{\mathbf{c}}')\right],$$

$$\mathbf{g}_B = \left(\mathbf{M}^{\mathbf{B}}\right)^{\mathrm{T}} \mathrm{vec}\left[\mathbf{B}'[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')] - \mathbf{Y}'[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\overline{\mathbf{Z}}'^{\mathrm{T}}\overline{\mathbf{C}}')] - \mathbf{M}'^{\mathrm{T}}\mathbf{A}'\mathrm{diag}(\underline{\mathbf{c}}')\right],$$

$$\mathbf{g}_C = \left(\mathbf{M}^{\mathbf{C}}\right)^{\mathrm{T}} \mathrm{vec}\left[\mathbf{C}'[(\mathbf{A}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{B}'^{\mathrm{T}}\mathbf{B}')] + \begin{bmatrix} -\overline{\mathbf{Z}'}[(\mathbf{X}'^{\mathrm{T}}\mathbf{A}') * (\mathbf{Y}'^{\mathrm{T}}\mathbf{B}')] \\ -\mathrm{vec}(\mathbf{M}')^{\mathrm{T}}(\mathbf{B}' \odot \mathbf{A}') \end{bmatrix}\right].$$

where

$$\mathbf{M}^{\mathbf{A}} = \left[\mathbf{I}_R \otimes \mathrm{diag}(\mathbf{n}_1);\ \ldots\ ;\ \mathbf{I}_R \otimes \mathrm{diag}(\mathbf{n}_L)\right],$$

with $\mathbf{I}_n$ the $(n \times n)$-identity matrix. The matrices $\mathbf{M}^{\mathbf{B}}$ and $\mathbf{M}^{\mathbf{C}}$ are defined analogously using the other factor matrices of the weight tensor, $\mathbf{P}$ and $\mathbf{Q}$. As before, we split $\mathcal{T}'$ into its old part $\overline{\mathcal{T}'} = \overline{\mathcal{W}} * \overline{\mathcal{T}}$ and its new slice $\mathbf{M}'$ and replace $\overline{\mathcal{T}'}$ by its CPD $[\![\mathbf{X}', \mathbf{Y}', \overline{\mathbf{Z}'}]\!]$. By exploiting the CPD structure in this way, we obtain sub-gradients which can be computed efficiently.

4.2.3 Gramian-vector product and preconditioner

In every CG-iteration, we need to efficiently compute a Gramian-vector product $\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\alpha}$. As the Gramian does not depend on $\mathcal{T}$, we can use similar expressions as those in [16] for the Gramian-vector products and preconditioner. We apply a block-Jacobi preconditioner [14] $\mathbf{V}$, consisting of the diagonal blocks $\mathbf{V}^{\mathbf{A}}$, $\mathbf{V}^{\mathbf{B}}$ and $\mathbf{V}^{\mathbf{C}}$, where

$$\mathbf{V}^{\mathbf{A}} = \left(\mathbf{M}^{A}\right)^{\dagger} \left([(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]^{-1} \otimes \mathbf{I}_I\right) \left(\mathbf{M}^{A}\right)^{\dagger \mathrm{T}},$$

and $\mathbf{V}^{\mathbf{B}}$ and $\mathbf{V}^{\mathbf{C}}$ are defined analogously. The inverses of the matrices of the form $[(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]^{-1}$ are inexpensive to compute, as these $(RL \times RL)$-matrices are small. We can exploit the structure of the problem to avoid the computation and storage of the pseudo-inverses. The preconditioner is applied through products of the form $\mathbf{V}\mathrm{vec}(\boldsymbol{\Gamma})$, with $\boldsymbol{\Gamma}$ an arbitrary matrix. This matrix $\boldsymbol{\Gamma}$ can be split according to the different factor matrices into $\boldsymbol{\Gamma} = [\boldsymbol{\Gamma}_{\mathbf{A}}; \boldsymbol{\Gamma}_{\mathbf{B}}; \boldsymbol{\Gamma}_{\mathbf{C}}]$ wherein the notation $\boldsymbol{\Gamma}_{\mathbf{A}}$ indicates that $\boldsymbol{\Gamma}_{\mathbf{A}}$ has the same dimensions as $\mathbf{A}$. We thus only need to compute products of the form $\mathbf{V}^{\mathbf{A}}\mathrm{vec}(\boldsymbol{\Gamma}_{\mathbf{A}})$ to apply the preconditioner. To perform such a product, we normalize the rows of $\mathbf{N}$ to obtain $\hat{\mathbf{N}}$ and then compute

$$(\hat{\mathbf{N}} \odot^{\mathrm{T}} \boldsymbol{\Gamma}_{\mathbf{A}})[(\mathbf{B}'^{\mathrm{T}}\mathbf{B}') * (\mathbf{C}'^{\mathrm{T}}\mathbf{C}')]^{-1}.$$

Next, we tensorize this $(I \times RL)$-matrix into an $(I \times R \times L)$-tensor $\hat{\mathcal{F}}$ and perform entry-wise multiplications of $\hat{\mathbf{N}}$ with every lateral slice of $\hat{\mathcal{F}}$. Finally, we compute the mode-3 sum of this tensor to obtain $\mathbf{V}^{\mathbf{A}}\mathrm{vec}(\boldsymbol{\Gamma}_{\mathbf{A}})$. The products $\mathbf{V}^{\mathbf{B}}\mathrm{vec}(\boldsymbol{\Gamma}_{\mathbf{B}})$, and $\mathbf{V}^{\mathbf{C}}\mathrm{vec}(\boldsymbol{\Gamma}_{\mathbf{C}})$ can be computed analogously, where the $\boldsymbol{\Gamma}_{\mathbf{A}}$ and $\boldsymbol{\Gamma}_{\mathbf{B}}$ have the same dimensions as $\mathbf{B}$ and $\mathbf{C}$, respectively.

4.2.4 Initialization and windowing

Assuming the model does not change too abruptly, the previous CPD $[\![\mathbf{X}, \mathbf{Y}, \overline{\mathbf{Z}}]\!]$ can be used to initialize the algorithm, after extending $\overline{\mathbf{Z}}$ with a new row $\mathbf{c}_{\text{new}}^{\mathrm{T}}$: thus $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{Y}$ and $\mathbf{C} = [\,\overline{\mathbf{z}}\,; \mathbf{c}_{\text{new}}^{\mathrm{T}}\,]$. The vector $\mathbf{c}_{\text{new}}$ can be obtained from the LS solution of the system

$$\min_{\mathbf{c}} \frac{1}{2} \left\| \mathbf{V} * \left([\![\mathbf{A}, \mathbf{B}, \mathbf{c}^{\mathrm{T}}]\!] - \mathbf{M} \right) \right\|_{\mathrm{F}}^2 ,$$

where $\mathbf{A}$ and $\mathbf{B}$ are fixed. We can rewrite this as

$$\min_{\mathbf{c}} \frac{1}{2} \left\| (\mathbf{B}' \odot \mathbf{A}')(\mathbf{c} \otimes \underline{\mathbf{q}}^{\mathrm{T}}) - \mathrm{vec}(\mathbf{M}') \right\|_{\mathrm{F}}^2 .$$

From the LS solution and the chain rule, we obtain after simplification:

$$\mathbf{c} = \mathrm{unvec}\left(\mathrm{vec}(\mathbf{M}')^{\mathrm{T}} \hat{\mathbf{Q}} \hat{\mathbf{W}}^{-1} \right)^{\mathrm{T}} \left(\mathbf{q}^{\mathrm{T}} / \|\mathbf{q}\|_2 \right) , \tag{4.3}$$

where $\hat{\mathbf{Q}} = (\mathbf{B}' \odot \mathbf{A}')$, $\hat{\mathbf{W}} = \hat{\mathbf{Q}}^{\mathrm{T}} \hat{\mathbf{Q}} = \left((\mathbf{A}'^{\mathrm{T}} \mathbf{A}') * (\mathbf{B}'^{\mathrm{T}} \mathbf{B}') \right)$, and $\mathrm{unvec}(\cdot)$ reorders the $(RL \times 1)$-matrix into an $(R \times L)$-matrix. The pseudoinverse of a vector is cheap to compute and the Khatri-Rao structure of $\hat{\mathbf{Q}}$ can be exploited when computing $\mathrm{vec}(\mathbf{M}')^{\mathrm{T}} \hat{\mathbf{Q}}$ [218].

To make the method perform in non-stationary environments, past slices can gradually be forgotten using a sliding window. This can be handled by simply removing the first row of $\overline{\mathbf{Z}}$ before the next updating step. An exponential window can be applied by multiplying the rows of $\mathbf{Q}$ with the appropriate scaling terms.

4.2.5 Complexity analysis

The per-iteration complexity of the algorithm is listed in Table 4.1 for a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, with $I = J = K$. The expressions are similar to those in Chapter 3, but in our case, the factor matrices have RL columns instead of R, the gradients require an extra multiplication and the evaluations of the preconditioned matrix-vector products require $3(RL)^2 \max(I, J, K) + 3(RL)^3$ flops. The weight tensor is updated using the NLS CPD updating method from Chapter 3, so the computation cost is dominated by that of the WLS updating step.

The method requires the storage of the new data and weighting slices ($\mathcal{O}(IJ)$), factor matrices, gradients and Gramian-vector products (all $\mathcal{O}(RL(I+J+K))$). Keeping the $\mathbf{M}^{\mathbf{A}}$-type matrices in memory is not required. In contrast, only storing the full tensor would already require $\mathcal{O}(IJK)$ memory.

Table 4.1: Per-iteration complexity of the WLS CPD updating algorithm for an $(I \times I \times I)$-tensor.

	$\text{Calls/it}_{\text{GN}}$	Complexity
Weighting of factor matrices	1	$\mathcal{O}(3RLI)$
Objective function	$1+\text{it}_{\text{Trust Reg.}}$	$\mathcal{O}(2(RL)^2I + I^2)$
Gradient	1	$\mathcal{O}(6(RL)^2I + 3RLI^2)$
Gramian-vector product	it_{CG}	$\mathcal{O}(3(RL)^2I + 3(RL)^3)$

4.3 Direction of arrival estimation (DOA)

Direction-of-arrival (DOA) estimation problems arise frequently in the field of array processing [122]. The goal of DOA estimation is to retrieve the direction in which several sources are impinging on a sensor array, which is crucial in applications such as radar and sonar. In the case of line-of-sight signals that impinge upon a uniform rectangular array (URA), DOA-retrieval can be performed through the computation of a CPD [128,165,172,173]. Additionally, it has been shown that incorporating prior knowledge about the sensor accuracies by use of a low-rank weight tensor can improve the performance of DOA-methods [16]. We show that our proposed method can achieve a higher accuracy than standard LS updating when sensor conditions are non-constant, e.g., a sensor breaks down unexpectedly or the environment temperature changes, while being more efficient than batch WLS.

Our URA has $M \times M$ sensors, each acquiring K samples from R different moving sources in the far field. The omnidirectional sensors are evenly spaced with inter-sensor spacing Δ. The observed tensor $\mathcal{T} \in \mathbb{C}^{M \times M \times K}$ is perturbed by noise and $\mathbf{t}_{ijk}$ holds the observed signal from sensor (i, j) at time k. The kth frontal slice of $\mathcal{T}$ admits a low rank approximation $\mathcal{T} \approx [\![\mathbf{A}^{(k)}, \mathbf{E}^{(k)}, \mathbf{s}^{(k)}]\!]$, as each source contributes one rank-1 term. Using the matrices of azimuth angles $\overline{\mathbf{Z}} \in \mathbb{R}^{M \times R}$ and elevation angles $\overline{\mathbf{L}} \in \mathbb{R}^{M \times R}$ of the R sources at each time step, one finds that the matrix $\mathbf{A}^{(k)} \in \mathbb{C}^{M \times R}$ has entries $a_{mr}^{(k)} = \exp((m-1)2\pi/\lambda \sin(\overline{z}_{rk}\pi/180)\Delta \mathrm{i})$ and $\mathbf{E}^{(k)} \in \mathbb{C}^{M \times R}$ has entries $e_{mr}^{(k)} = \exp((m-1)2\pi/\lambda \sin(\overline{l}_{rk}\pi/180)\Delta \mathrm{i})$, with i the imaginary unit. Both are Vandermonde matrices due to the evenly spaced sensors. The vector $\mathbf{s}^{(k)} \in \mathbb{C}^{1 \times R}$ contains the sources and λ is the wavelength. With a rank-R CPD of a few frontal slices of $\mathcal{T}$, the factor matrices $\mathbf{A}^{(k)}$ and $\mathbf{E}^{(k)}$ can be recovered, and from them, the positions of the sources [128,165,172,173].

We choose a URA with $M^2 = 400$ sensors and determine the DOA of $R = 3$ sources with oscillating trajectories. $K = 100$ samples are collected and in each time step, the CPD of the observed tensor is updated to incorporate the new data.

Table 4.2: The median absolute errors of the azimuths and elevations are much smaller if a low-rank weight tensor is applied.

	LS			WLS		
Azimuth	10.788	1.228	16.436	0.652	0.340	0.976
Elevation	3.898	2.311	3.548	0.297	0.270	0.250

We assume that the sensors obtain samples with an SNR of 20 dB. We further assume that sensors 26 to 185 break down after 40 samples, which results in their sample SNR dropping to -14 dB. We choose a weight tensor $\mathcal{W}$, where all w_{ijk} are set to 1, except for the entries where $26 \leq 10(i-1)+j \leq 185$ & $k > 40$. These are set to 0.02 to reflect the relative difference in SNR between the sensors after the breakdown. These weights were obtained by a heuristic approach, but optimal weights can be obtained from the Fisher information. Note that $\mathcal{W}$ has rank 3, but the parts before and after the breakdown are rank-1 and rank-2, respectively, so we start with a rank-1 $\mathcal{W}$ and update it to a rank-3 (at slice 40) and finally a rank-2 (at slice 46) CPD.

We compare LS updating methods for the weighted and non-weighted case. The updating methods start from a rank-3 CPD of the first 20 slices of $\mathcal{T}$ and update the CPD after every new sample slice is added to the tensor, using a sliding window of 6 slices, to estimate the mean DOAs during the last 6 time steps. The updating methods are limited to $\text{it}_{\text{GN}} = \text{it}_{\text{CG}} = 5$. The memory requirements of the updating methods are much smaller than those of their batch counterparts, as only one slice has to be stored alongside the old CPD. In Table 4.2, we report the median across 100 trials of the median absolute estimation errors during the updating process. It is clear that the accuracy of the DOA estimation improves by applying the low-rank weight tensor $\mathcal{W}$ to the problem, as the WLS updating method outperforms the LS updating method significantly. In Figure 4.2, one trial of the experiment is shown. One can again see that it is beneficial to apply a low-rank weight tensor to the CPD computation. Unlike the LS updating method, which fails after sensor breakdown, the estimates of the WLS updating method remain good.

For the relatively small third-order tensors from the DOA experiment, time gains are rather limited. In Table 4.3, we report timing results for large-scale tensors. The listed timings (in seconds) are for a one-slice update in the third mode of a rank-5 CPD, weighted with a rank-2 weight tensor. All methods are limited to $\text{it}_{\text{GN}} = \text{it}_{\text{CG}} = 5$. The CPDs are computed with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0. It can be seen that the updating method outperforms its batch counterpart for sufficiently large tensors, with memory requirements that are

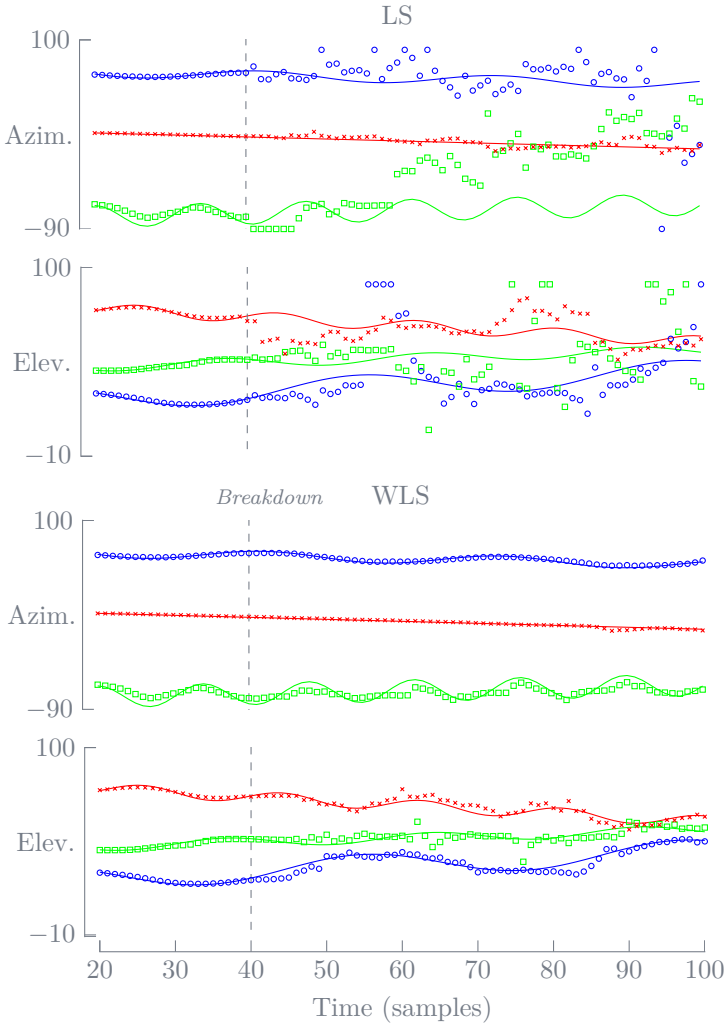

Figure 4.2: A low-rank weight tensor can improve DOA estimation by giving lower importance to values obtained by broken sensors. Three sources are tracked during 80 samples and their DOAs (full line) and estimated DOAs (marks) are displayed. Tracking clearly fails after the sensor breakdown at $t = 40$ samples for the LS updating method (top two figures), while the WLS updating method (bottom two figures) continues to obtain good estimates despite the breakdown.

Table 4.3: The updated WLS-method requires less CPU-time (s) than its batch counterpart.

Tensor dimensions	WLS Updating	Batch WLS
$400 \times 400 \times 400$	10.5	14.6
$500 \times 500 \times 500$	20.6	29.1
$600 \times 600 \times 600$	34.7	49.1
$700 \times 700 \times 700$	55.3	110
$800 \times 800 \times 800$	81.9	574

negligible compared to those of the batch method.

4.4 Conclusion and further work

A new updating method for CPD updating with low-rank weights is proposed. We exploit the CPD structure of both the weight and data tensor to efficiently compute a new CPD when new slices are added to the data tensor. Expressions are derived for the objective function, gradients and gradient-vector products of the NLS algorithm, along with an analysis of their complexity. We show that WLS CPD updating with low-rank weights outperforms the standard LS algorithm for DOA-estimation. Its accuracy is close to that of the batch WLS method, while memory requirements are much smaller.

The low-rank weight tensor can be updated while new samples are added to the tensor. Aside from using prior information, as we did in the DOA example, one could also modify the weight tensor in a data-driven way. This could, for instance, be done by either increasing or decreasing the weight of tensor entries that are badly explained by the model. Further, as the updating algorithm can handle rank changes, automatic rank-detection would be very useful to adapt the rank of the CPD of both the weight tensor and the data tensor to new data during the updating process. Finally, including subspace information by tracking an LMLRA, as in Chapter 5, alongside the CPD, would be an interesting extension. In this chapter, we focus on the case of third-order tensors, but extensions to higher-order tensors are possible. For these tensors, the efficiency gains in memory and computation time become even more significant.

Tracking the low multilinear rank approximation

5

ABSTRACT

A low multilinear rank approximation (LMLRA) of a tensor is often used to compress a large tensor into a much more compact form, while still maintaining most of the information in the tensor. This is achieved by only storing the principal subspace in every mode and truncating other singular vectors. These subspaces are then combined using a core tensor of (much) smaller dimensions than the original tensor. When the tensor is non-static, for example when new tensor slices are regularly added in one mode, one can expect that the LMLRA of the tensor changes only slightly and an LMLRA of the new tensor can be derived from the previous one. By updating the subspaces and recomputing the core tensor in an efficient way when a new tensor slice is added, this can indeed be achieved. In this chapter, a new method is proposed to track the subspaces in both the evolving and non-evolving modes, leading to efficient and accurate updates of the LMLRA, even if the subspaces change drastically from one update to the next. This is achieved by tracking a truncated multilinear UTV decomposition (TMLUTVD), which is a tensor decomposition that is multilinear rank-revealing, yet less expensive to compute than a multilinear singular value decomposition. We derive accuracy bounds for the TMLUTVD and give several strategies to speed up its computation.

5.1 Introduction

Tensors, which extend vectors and matrices to higher orders, have been steadily gaining in popularity and have been used extensively in applications in signal processing and machine learning [35, 174]. They provide a natural and convenient way to represent higher-order data without destroying its multilinear structure. As tensors can easily become impractically large, a lot of work

This chapter is based on M. Vandecappelle, and L. De Lathauwer, “Tracking the low multilinear rank approximation," to be submitted to IEEE Transactions on Signal Processing. The specific contributions of the doctoral candidate are listed in Section 11.1.

has focused on the efficient computation of tensor decompositions. Like their matrix counterparts, these decompositions provide a way to extract meaningful components from tensors or to store large tensors compactly, while retaining the most important features of the data. Some well-known decompositions are the Canonical Polyadic Decomposition (CPD) [103], the Tensor-Train decomposition (TTD) [151] and the Low Multilinear Rank Approximation (LMLRA) [208], of which the Multilinear Singular Value Decomposition (MLSVD) [47] is a special case. It decomposes an Nth-order tensor into N orthogonal factor matrices, which are linked by a core tensor, see Figure 5.1. Even though the Eckart–Young theorem cannot be extended to the MLSVD, truncating this decomposition still leads to an approximation of the tensor which is very close to optimal for the given dimensions; see [48] and references therein for a discussion of this approximation accuracy. Moreover, the numerical properties of the MLSVD computation benefit greatly from the orthogonality of the factors.

In real-life applications, tensors are often non-static. New data can become available over time and should be appended to the current tensor or old tensor values become outdated and are replaced by newer measurements. When time is limited, it can be prohibitively expensive to recompute a tensor decomposition when only a small subset of the tensor entries are modified. When the full tensor is not stored due to its size, this is impossible altogether. This shows the need for methods that can include the new information into the existing tensor decomposition while keeping execution time and memory requirements as low as possible. The truncated MLSVD (TMLSVD) is a particularly interesting decomposition to track, as it is a natural higher-order extension of the truncated SVD, is extensively used in tensor applications and can be computed efficiently. A TMLSVD of a tensor can be obtained by computing the SVD of the tensor unfoldings in every mode and trimming the multilinear singular vectors corresponding to the smallest multilinear values. The core tensor of the decomposition is then constructed afterwards. The TMLSVD orders the multilinear singular values in every mode by their magnitude. Consequently, any number of multilinear singular vectors can be trimmed in every mode while still guaranteeing a good approximation of the tensor.

When tracking a noisy low-rank tensor, one is often only interested in extracting the principal subspace of the tensor in every mode: all multilinear singular vectors in the principal subspaces are retained, while those corresponding to the noise are trimmed. The TMLSVD can be used to extract the principal subspaces by keeping only multilinear singular vectors corresponding to multilinear singular values above a predefined noise threshold. In every mode, these multilinear singular vectors are combined to form the principal subspace. Alternatively, when extraction of the principal subspaces is the only requirement, a cheaper rank-revealing decomposition can be used instead of the SVD. For example, a

rank-revealing UTV (RRUTV) decomposition, wherein the middle matrix is triangular instead of diagonal for the SVD, provides a basis for the principal subspace that is less expensive to compute than the basis obtained by the SVD. The bases for the principal subspaces in every mode can then be used as factor matrices for an LMLRA.

We focus on the tensor tracking case where new slices are added to the tensor in a certain mode, see Figure 5.2. To better handle sudden changes in the signal subspaces, a sliding window is applied to downdate older slices. LMLRA updating can be made attractive from a computational point of view, as a major part of the update can be achieved using matrix subspace tracking methods, which have been extensively studied; see for example [54] for an overview of existing methods. Unfortunately, many fast matrix subspace updating methods, such as PAST/PASTd [237] and FAPI [8] are not rank-revealing. As automatic rank determination is important in LMLRA tracking, we use a rank-revealing URV (RRURV) or ULV (RRULV) decomposition to track the principal subspaces of the different modes. As explained above, this allows the principal subspaces to be tracked more efficiently than when an SVD is updated.

Tensor subspace tracking when slices are added has been proposed before. In [107,137], all subspaces of a tensor are tracked simultaneously. These methods neglect the multilinear structure of the problem and require the storage of large intermediate matrices, making them unsuitable for large scale applications. The method from [199] was designed to estimate the subspaces of a tensor that differs only slightly from a tensor for which the subspaces are known. As such, it can be used for streaming applications. Next, in [200], a sketching approach is used to restrict the tensor to its principal subspace in each mode with the use of random projections following [93]. This allows the computation of an LMLRA with a single pass over all tensor entries, making the randomized method suitable for streaming applications. Finally. in [33], a general framework is proposed to track the different subspaces of a tensor. Only the principal subspaces in every mode are stored between updates, as well as a matricized version of the core tensor. As this framework assumes that the updates of the subspaces happen in parallel, no sequential truncating strategies are employed when moving from one subspace to the next, although these can significantly speed up the algorithm in non-parallel settings.

In this chapter, we propose a method that allows to efficiently track an LMLRA of a tensor when new slices are added in one mode along a sliding window. This is done by updating a truncated multilinear UTV decomposition (TMLUTVD) of the tensor as slices are added and removed. Bases of the principal subspaces are tracked automatically for every mode and their corresponding dimensions are determined. In every update, the approximation error of the TMLUTVD can be bounded in function of the predefined noise threshold. In Section 5.2,

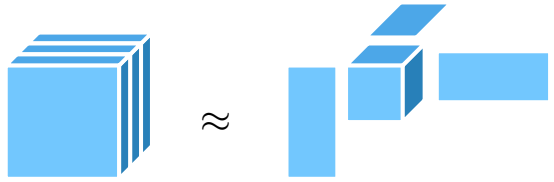

Figure 5.1: LMLRA of a third-order tensor. The tensor is decomposed into three orthogonal factor matrices, linked by a small core tensor.

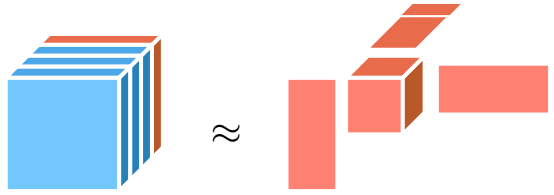

Figure 5.2: Update of the LMLRA of a third-order tensor. The three factor matrices holding the subspaces are changed slightly and the third factor matrix is extended in the evolving direction. The core tensor changes as well.

the TMLUTVD is introduced and the basic updating algorithm is discussed in Section 5.3. Next, in Section 5.4, bounds are derived and proved for the approximation error of the TMLUTVD. In Section 5.5, three strategies are discussed that can significantly decrease the storage and computational requirement of the method, at the cost of a slight decrease in accuracy. Finally, Section 5.6 shows some experiments on synthetic data. The algorithm is laid out for a real third-order tensor that is updated in the third mode. Thus, mode 3 is hereafter considered as the evolving mode and the dimension of this mode changes during the updating process. The mode-1 and mode-2 subspaces are updated in every iteration as well, but as their dimensions remain the same, these modes are called non-evolving modes. Evidently, the method can be applied to complex tensors as well and any mode can be chosen as the updating mode. Extending the method to higher orders is straightforward, as this only increases the number of non-evolving modes.

5.1.1 Notation and terminology

Scalars, vectors and matrices are denoted by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively, while tensors are written in calligraphic script ($\mathcal{A}$). The identity matrix and the zero matrix are denoted by $\mathbf{I}$ and $\mathbf{0}$, respectively. The Kronecker product of matrices is written as $\otimes$. When

using subscripts to specify certain entries of a tensor $\mathcal{T}$, a colon refers to all entries of a specific mode, e.g. $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. When formatting the entries of an Nth order tensor $\mathcal{T}$ in matrix format, the notation $\mathbf{T}_{[i_{m_1},i_{m_k};i_{m_{k+1}},i_{m_N}]}$ is used, with $[i_{m_1}, i_{m_k}; i_{m_{k+1}}, i_{m_N}]$ providing the processing order of the different tensor modes in the row and column dimensions, i.e., the entries before the semicolon (;) specify the order in which the different modes are processed in the rows of the matrix, while those after the semicolon specify the mode order for the columns. If all modes appear in the column dimension, the end result is a vector. In this case, the semicolon is dropped and a lowercase letter is used: for a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, $\mathbf{t}_{[3,2,1]} \in \mathbb{R}^{I_1 I_2 I_3 \times 1}$ refers to the standard mode order for the vectorization of a third-order tensor. A mode-n fiber of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfoldings $\mathbf{T}_{[1;3,2]}$, $\mathbf{T}_{[2;3,1]}$ and $\mathbf{T}_{[3;2,1]}$ of a third order tensor $\mathcal{T}$ are constructed by stacking all its mode-n fibers as columns of a matrix, using the mode ordering specified by the notation. For example: in the mode-2 unfolding $\mathbf{T}_{[2;3,1]}$, the column ordering of the mode-2 fibers runs along mode 3 and mode 1, in that order, thus $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{[2;3,1]}\right)_{i_2,q}$, with $q = i_1 + (i_3 - 1)I_1$. The mode-n tensor-matrix product of a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and a matrix $\mathbf{X} \in \mathbb{R}^{J \times I_n}$ is written as $\mathcal{T} \cdot_n \mathbf{X}$. For $n = 1$, 2 and 3, $\mathcal{T} \cdot_n \mathbf{X} \in \mathbb{R}^{J \times I_2 \times I_3}$, $\mathbb{R}^{I_1 \times J \times I_3}$ and $\mathbb{R}^{I_1 \times I_2 \times J}$, respectively. In unfolded form, computing the mode-n product is the same as computing $\mathbf{X}\mathbf{T}_{[n;1,2,\ldots,n-1,n+1,n+2,\ldots,N]}$. Kronecker products of the form $\mathbf{X}(\mathbf{Z} \otimes \mathbf{Y})$ are never explicitly formed, as they can be computed as matricized tensor times Kronecker products (MTKRONPRODs), for which specialized routines exist [218]. It suffices to store the individual matrices $\mathbf{Y}$ and $\mathbf{Z}$.

A low multilinear rank approximation (LMLRA) of a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is defined as $\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$, where $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times R_1}$, $\mathbf{U}_2 \in \mathbb{R}^{I_2 \times R_2}$, $\mathbf{U}_3 \in \mathbb{R}^{I_3 \times R_3}$. The vector (R_1, R_2, R_3) with $R_1 \leq I_1$, $R_2 \leq I_2$, $R_3 \leq I_3$ holds the number of (linearly independent) columns that are used in the factor matrices $\mathbf{U}_i$ of the approximation for every mode. If the approximation is exact and the $R^{(i)}$ are minimal, then this vector is called the multilinear rank of the tensor. One way to obtain the factor matrices of an LMLRA is to compute bases $\mathbf{U}_n$ for the principal column spaces of all mode-n unfoldings of the tensor. The core tensor $\mathcal{S}$ can then be found by forming $\mathcal{S} = \mathcal{T} \cdot_1 \mathbf{U}_1^{\mathrm{T}} \cdot_2 \mathbf{U}_2^{\mathrm{T}} \cdot_3 \mathbf{U}_3^{\mathrm{T}}$. While this construction method does not yield the least-squares optimal approximation, it is often only mildly suboptimal.

5.2 Multilinear UTV decomposition

The MLSVD [47] of a tensor decomposes an Nth-order tensor into a set of N orthogonal factor matrices, which are obtained through N SVDs. These factor matrices are linked by a core tensor. The MLSVD is a highly-structured decomposition that reveals a lot of information about the tensor. For each mode, the (vectorizations of the) mode-n slices of the core tensor are orthogonal to each other and ordered according to their norms, also called mode-n multilinear singular values. Due to the latter property, the MLSVD can be truncated in any mode by dropping the factor matrix columns corresponding to the smallest multilinear singular values for that mode. Regardless of the number of columns that is truncated, the resulting LMLRA is close to optimal for the given multilinear rank [47]. The truncation can either be performed in parallel, i.e., for each mode the SVD is computed and truncated independently, or sequentially, i.e., the truncations are interleaved with the computation of the SVDs for the different modes. These approaches result in the TMLSVD [47] and the sequentially truncated MLSVD (STMLSVD) [217], respectively. The STMLSVD can generally be computed more efficiently, but it cannot be parallelized and its truncation error is sometimes larger than for the TMLSVD.

When time is limited, computing an MLSVD can be prohibitively expensive. At the same time, it is often not necessary to compute all information that this decomposition offers. Instead of finding all multilinear singular values and allowing truncation to an arbitrary multilinear rank, it can be sufficient to compute an orthogonal basis for each mode, as in [82] or to split each mode-n subspace into its principal subspace, say of dimension R_n, and the orthogonal complement thereof. In this chapter, we focus on the latter case. Ideally, the mode-n principal subspace corresponds to the underlying signal, while the orthogonal complement corresponds to noise. For each mode, this split can be performed using a decomposition that is cheaper to compute than the SVD and it is not necessary to explicitly compute all multilinear singular values. In fact, one can even use a decomposition that can be computed in a finite number of steps instead of the SVD whose computation is an iterative process. Using a rank-revealing UTV decomposition, wherein the middle matrix is triangular (ULV for lower-triangular and URV for upper triangular), for instance, one obtains a multilinear UTV decompostion (MLUTVD).

Some properties of the MLSVD can be extended to the MLUTVD. For instance, one has, for mode 1 of an MLURVD $\mathcal{T} = \mathcal{S} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$:

$$\begin{aligned} \mathbf{U}_1\mathbf{R}_1\mathbf{V}_1^{\mathrm{T}} &= \mathbf{T}_{[1;3,2]} \\ \Leftrightarrow \mathbf{U}_1\mathbf{R}_1\mathbf{V}_1^{\mathrm{T}} &= \mathbf{U}_1\mathbf{S}_{[1;3,2]}\left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)^{\mathrm{T}} \\ \Leftrightarrow \mathbf{R}_1\mathbf{V}_1^{\mathrm{T}} &= \mathbf{S}_{[1;3,2]}\left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)^{\mathrm{T}} \\ \Leftrightarrow \mathbf{R}_1\mathbf{V}_1^{\mathrm{T}}\left(\mathbf{U}_3 \otimes \mathbf{U}_2\right) &= \mathbf{S}_{[1;3,2]}. \end{aligned}$$

The upper triangularity of $\mathbf{R}_1$ means that the last row of $\mathbf{S}_{[1;3,2]}$ is generated by the last row of the orthogonal matrix $\mathbf{V}_1^{\mathrm{T}}\left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)$, the second-to-last row of $\mathbf{S}_{[1;3,2]}$ is generated by the last two rows of the orthogonal matrix $\mathbf{V}_1^{\mathrm{T}}\left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)$, and so on. This is a generalization of the 'all-orthogonality' [47] property of the MLSVD, i.e., for an MLSVD $\mathcal{T} = \tilde{\mathcal{S}} \cdot_1 \tilde{\mathbf{U}}_1 \cdot_2 \tilde{\mathbf{U}}_2 \cdot_3 \tilde{\mathbf{U}}_3$, the factor matrix generating the principal mode-1 subspace is explicitly rotated such that the matrix $\tilde{\mathbf{R}}_1$ in the expression $\mathbf{T}_{[1;3,2]} = \tilde{\mathbf{U}}_1\tilde{\mathbf{R}}_1\tilde{\mathbf{V}}_1^{\mathrm{T}}$ is diagonal and, consequently, each row of the unfolded core tensor is generated by the corresponding row of the orthogonal matrix $\tilde{\mathbf{V}}_1^{\mathrm{T}}\left(\tilde{\mathbf{U}}_3 \otimes \tilde{\mathbf{U}}_2\right)$.

Like the MLSVD, the MLUTVD allows for a truncation to a rank-$(R_1, R_2, \ldots, R_N)$ LMLRA, see Figure 5.3, with a truncation error that is close to minimal. Unlike the MLSVD, however, truncation to an arbitrary multilinear rank does not come for free, but requires additional computations. For the MLUTVD, the truncation can also be performed in parallel or sequentially, as UTV counterparts of the TMLSVD and the STMLSVD, respectively. As before, the sequentially truncated MLUTVD can be computed more efficiently, but its truncation error can be larger. For the TMLSVD, bounds can be derived that express the maximal truncation error in function of the multilinear singular values that correspond to the truncated part in each mode. Something similar can be done for the truncated MLUTVD (TMLUTVD) by looking at the norms of the parts of the triangular matrices that are truncated in each mode. In Section 5.4, we derive bounds on the approximation error of the TMLUTVD in both the Frobenius and spectral norms. In the following section, we show how the TMLUTVD can be applied in an updating context.

5.3 MLUTV Updating

Low-rank matrix tracking methods can be applied when a new signal vector becomes available at certain time intervals. Assume that one has a matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$, where every column $\mathbf{x}_{:j}$ represents a sample obtained at time j.

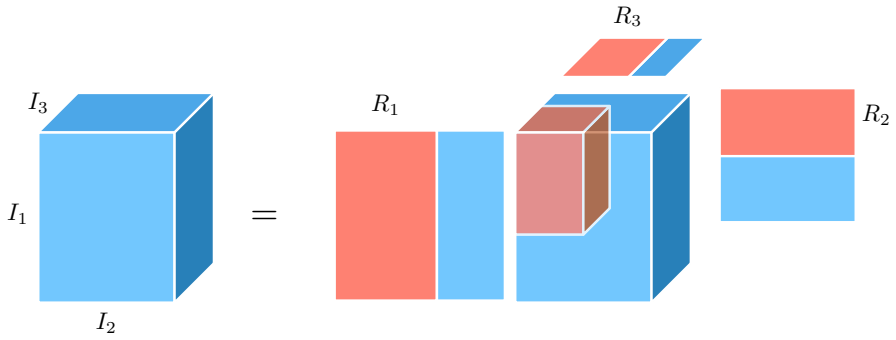

Figure 5.3: The MLUTVD can be truncated to obtain an LMLRA of a tensor by only using the columns of the factor matrices that correspond to the principal mode-n subspaces and the corresponding entries of the core tensor, marked in red.

In the SVD of $\mathbf{X}$, one has matrices $\mathbf{U} \in \mathbb{R}^{I \times R}$, $\mathbf{\Sigma} \in \mathbb{R}^{R \times R}$ and $\mathbf{V} \in \mathbb{R}^{J \times R}$, with $R \leq I$, $\mathbf{U}$ and $\mathbf{V}$ column-wise orthonormal and $\mathbf{\Sigma}$ diagonal such that $\mathbf{X} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$. Every time sample $\mathbf{x}_{:j}$ can be represented as $\mathbf{U}\mathbf{\Sigma}\mathbf{v}_{j:}^{\mathrm{T}}$, as shown in Figure 5.4 (top). The other way around, $\mathbf{U}^{\mathrm{T}}\mathbf{x}_{:j} \in \mathbb{R}^{R}$ is a low dimensional representation of the time sample $\mathbf{x}_{:j} \in \mathbb{R}^{I}$. When new time samples are added as new columns to the matrix $\mathbf{X}$, the matrix $\mathbf{U}$ has to be updated so that it represents the principal column space of the new $\mathbf{X}$. Using this updated $\mathbf{U}$, we can again find low dimensional approximations of any time sample in $\mathbf{X}$.

If the time sample is not a vector, but rather a matrix or even a tensor, this low-rank matrix tracking problem can be extended to a tensor tracking problem. By keeping the tensor structure intact, one can often obtain more meaningful approximations than when matrix methods are used. Let $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ be a tensor where the frontal slice $\mathbf{T}_{::k}$ represents the sample obtained at time k. Then, an LMLRA $\mathcal{C} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$ of $\mathcal{T}$ can be obtained by computing bases for the principal subspaces of $\mathcal{T}$ in mode-1, mode-2 and mode-3. These bases are stored in $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times R_1}$, $\mathbf{U}_2 \in \mathbb{R}^{I_2 \times R_2}$ and $\mathbf{U}_3 \in \mathbb{R}^{I_3 \times R_3}$, respectively. The core tensor $\mathcal{C} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ captures the interactions between modes. Unfolded in the third mode, the approximation has the following form: $\mathbf{T}_{[2,1;3]} = (\mathbf{U}_2 \otimes \mathbf{U}_1)\,\mathbf{C}_{[2,1;3]}\mathbf{U}_3^{\mathrm{T}}$, as shown in Figure 5.4 (bottom). A similar, but stronger, interpretation can be given compared to the matrix case: by compressing both mode 1 and mode 2 to their respective principal subspaces, a lower dimensional representation can be obtained for each frontal slice of the tensor. For the kth frontal slice, this lower dimensional representation looks as follows: $\mathbf{T}_{::k} \cdot_1 \mathbf{U}_1^{\mathrm{T}} \cdot_2 \mathbf{U}_2^{\mathrm{T}}$. When new time samples are added to the tensor,

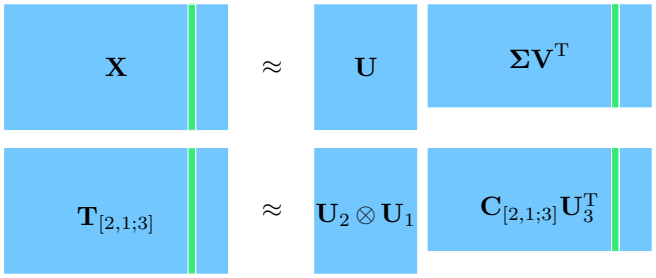

Figure 5.4: Top: low-rank approximation of a matrix $\mathbf{X}$ as the product of the matrices $\mathbf{U}$ and $\mathbf{\Sigma V}^{\mathrm{T}}$. One time-sample is highlighted in $\mathbf{X}$ and $\mathbf{\Sigma V}^{\mathrm{T}}$. Bottom: Third-order extension of the factorization above. The samples are frontal slices of a tensor $\mathcal{T}$ and the LMLRA is given by $\mathbf{T}_{[2,1;3]} = (\mathbf{U}_2 \otimes \mathbf{U}_1)\,\mathbf{C}_{[2,1;3]}\mathbf{U}_3^{\mathrm{T}}$. As for the matrix case, one time sample is highlighted in $\mathbf{T}_{[2,1;3]}$ and $\mathbf{C}_{[2,1;3]}\mathbf{U}_3^{\mathrm{T}}$.

the mode-1 and mode-2 principal subspaces should be updated in a stable and efficient way to include these new samples in the LMLRA. If old time samples are to be forgotten, the subspaces should be adjusted accordingly as well. The case where only the subspaces of the non-evolving modes 1 and 2 are tracked, which is relevant for direction-of-arrival estimation, for instance, is outlined in [211]. In the more general case proposed below, the principal subspace of the evolving mode 3 is tracked as well. If the full LMLRA of the tensor is required, the core tensor can be tracked alongside the subspaces.

When a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is tracked with a sliding window of length w, new slices are added and removed in the updating mode. Only after adding the first w slices, do we start to remove the oldest slice. From then on, the tracked tensor is always of dimensions $(I_1 \times I_2 \times w)$. When no slices are removed, the window is said to be infinite and the tracked tensor at update n has dimensions $(I_1 \times I_2 \times w)$, with $w = n$. Superscripts of objects refer to the index of the update in which they have been computed, so $\mathbf{X}^{(n)}$ refers to a matrix that was computed in the nth updating step.

Below, we supply the basic steps to perform a full LMLRA update. In later sections, we extend this basic algorithm to make it more efficient, both computationally and with respect to storage requirements, further facilitating its utilization in real-time applications. To perform an update (and/or downdate) of the LMLRA, several operations have to be performed. The first is an update of the subspaces (the principal column spaces of the unfoldings of $\mathcal{T}$) in the non-evolving modes 1 and 2, discussed in Section 5.3.1. The second operation is an update of the subspace in the evolving mode 3, which is explained in Section 5.3.2. In Section 5.3.3, the storage and computational complexities of

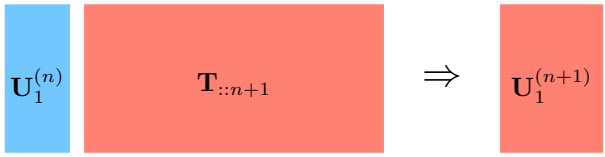

Figure 5.5: Update of the LMLRA for mode 1. The new columns of the mode-1 unfolding are included in the subspace $\mathbf{U}_1^{(n)}$.

the algorithm are compared to those of the batch method and strategies for setting truncation threshold values are discussed in Section 5.3.4.

5.3.1 Subspace tracking of the non-evolving modes

For mode 1, of which the dimension of the ambient space (I_1) does not evolve, the goal is to track the column space of the mode-1 unfolding $\mathbf{T}_{[1;3,2]}$ of $\mathcal{T}$. Let us assume that this column space has dimension $R_1^{(n)}$ after n slices have been added in the third mode, with $1 \leq R_1^{(n)} \leq I_1$. When a new slice $\mathbf{T}_{::n+1} \in \mathbb{R}^{I_1 \times I_2 \times 1}$ is added to the tensor, this corresponds to adding all mode-1 fibers of the slice as columns to $\mathbf{T}_{[1;3,2]}^{(n)}$, see Figure 5.5. Thus, the mode-1 subspace should take every mode-1 fiber of the new slice $\mathbf{T}_{::n+1}$ into account. Likewise, when an old slice is removed from the tensor, the contributions of all mode-1 fibers of that old slice should be removed.

We first set a noise threshold γ_1 which is used to decide whether a singular vector of a matrix $\mathbf{X}$ belongs to its principal subspace or to the orthogonal complement. The singular values of $\mathbf{X}$ are the square roots of the eigenvalues $\lambda_i \geq 0$ of $\mathbf{X}\mathbf{X}^{\mathrm{T}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$, with $\mathbf{U}$ orthogonal and the diagonal matrix $\mathbf{\Lambda}$ holding the sorted eigenvalues. The noise threshold γ_1 is chosen such that the large eigenvalues $\lambda_i \geq \gamma_1^2$ correspond to the underlying signal, while small eigenvalues $\lambda_i < \gamma_1^2$ correspond to the noise. Ideally, γ_1 is chosen within a clear gap between the large and small eigenvalues that indicates that the latter have been introduced by the noise. The k columns of $\mathbf{U}$ that belong to the retained eigenvalues then span the principal subspace of $\mathbf{X}$. Consequently, singular vectors corresponding to singular values above γ_1 are considered to be part of the principal subspace and the other singular values as corresponding to the noise. In Section 5.3.4, strategies are discussed for choosing γ_1 appropriately.

As explained in Section 5.1, we only wish to track the principal subspace of $\mathbf{T}_{[1;3,2]}$ and not the individual singular vectors, so any principal subspace-revealing decomposition can be updated instead of the SVD. Here, a signed RRULV decomposition is used, which is less expensive to compute than an SVD.

While its resulting subspace is not the same as the one obtained with the SVD, it typically lies quite close. When computing an RRULV decomposition, one can exploit the fact that if $\mathbf{X}$ has d singular values above γ_1, then $\mathbf{X}\mathbf{X}^{\mathrm{T}} - \mathbf{N}\mathbf{N}^{\mathrm{T}}$ has d eigenvalues above zero for any $\mathbf{N}$ such that $\mathbb{E}(\mathbf{N}\mathbf{N}^{\mathrm{T}}) = \gamma_1^2\mathbf{I}$. By separating the subspace corresponding to the positive eigenvalues from the one corresponding to the negative eigenvalues, the rank of the principal subspace with respect to the noise threshold γ_1 can be extracted exactly and a good estimate of the principal subspace can be obtained.

After processing n tensor slices, the signed RRULV decomposition of the mode-1 unfolding of the tensor has the following form:

$$\mathbf{Q}_1^{(n)^{\mathrm{T}}} \left[\mathbf{N}_1^{(n)}, \mathbf{S}_1^{(n)}\right] \mathbf{\Phi}_1^{(n)} = \left[\mathbf{L}_N^{(n)}, \mathbf{0}, \mathbf{L}_S^{(n)}, \mathbf{0}\right]. \tag{5.1}$$

Herein, $\mathbf{S}_1^{(n)} \in \mathbb{R}^{I_1 \times I_2 w}$ is a matrix holding all mode-1 fibers that are currently within the sliding window, while $\mathbf{N}_1^{(n)}\mathbb{R}^{I_1 \times I_2 (n-w)}$ is a matrix holding all mode-1 fibers that have been dropped from the sliding window and are thus no longer included in the subspace computation. Thus $\left[\mathbf{N}_1^{(n)}, \mathbf{S}_1^{(n)}\right]$ is a matrix representation of all processed mode-1 fibers that grows as more updates are performed. The matrix $\mathbf{Q}_1^{(n)} \in \mathbb{R}^{I_1 \times I_1}$ is orthogonal, $\mathbf{\Phi}_1^{(n)} \in \mathbb{R}^{I_2 n \times I_2 n}$ is J-orthonormal and $\mathbf{L}_1^{(n)} = \left[\mathbf{L}_N^{(n)}, \mathbf{L}_S^{(n)}\right] \in \mathbb{R}^{I_1 \times I_1}$ is lower triangular. The matrix $\mathbf{\Phi}_1^{(n)}$ being J-orthonormal means that $\mathbf{\Phi}_1^{(n)}\mathbf{J}\mathbf{\Phi}_1^{(n)^{\mathrm{T}}} = \mathbf{J}$ and $\mathbf{\Phi}_1^{(n)^{\mathrm{T}}}\mathbf{J}\mathbf{\Phi}_1^{(n)} = \mathbf{J}$ hold for $\mathbf{J} = \mathrm{diag}\left([1; \ldots; 1; -1; \ldots; -1]\right)$. The positive-negative structure of $\mathbf{J}$ follows the block-partitioning of $\mathbf{\Phi}_1^{(n)}$, which in turn depends on γ_1. Because of the J-orthonormal structure of $\mathbf{\Phi}_1^{(n)}$, columns of $\mathbf{L}_S^{(n)} \in \mathbb{R}^{I_1 \times R_1^{(n)}}$, which are the columns of $\mathbf{L}_1^{(n)}$ corresponding to the principal mode-n subspace, have a different signature than columns that correspond to the orthogonal complement, stored in $\mathbf{L}_N^{(n)}$. Consequently, the principal subspace can be obtained by splitting the columns of $\mathbf{Q}_1^{(n)}$ into orthonormal bases for the principal column space $\mathbf{Q}_S^{(n)} \in \mathbb{R}^{I_1 \times R_1^{(n)}}$ and for its orthogonal complement $\mathbf{Q}_N^{(n)} \in \mathbb{R}^{I_1 \times (I_1 - R_1^{(n)})}$: $\mathbf{Q}_1^{(n)} = \left[\mathbf{Q}_N^{(n)} | \mathbf{Q}_S^{(n)}\right]$. The signed RRULV decomposition can be updated in a stable and non-iterative way using the Signed URV (SURV) method of Zhou et al. [243] with a complexity comparable to that of a QR update ($\mathcal{O}(I_1^2)$ flop). Both exponential and sliding windows can be handled in the SURV algorithm. Compared to other updating methods, the latter is particularly easy, as updates and downdates of new columns are handled similarly. Below, we consider sliding window updating.

If a new slice $\mathbf{T}_{::n+1}$ is added to the tensor, this corresponds to appending its columns to the matrix $\mathbf{S}_1^{(n)}$. The decomposition (5.1) is thus

updated to include the new slice, which results in a new decomposition $\mathbf{Q}_1^{(n+1)^{\mathrm{T}}} \left[\mathbf{N}_1^{(n+1)}, \mathbf{S}_1^{(n+1)}\right] \mathbf{\Phi}_1^{(n+1)} = \left[\mathbf{L}_N^{(n+1)}, \mathbf{0}, \mathbf{L}_S^{(n+1)}, \mathbf{0}\right]$, with $\mathbf{N}_1^{(n+1)} = \mathbf{N}_1^{(n)}$ and $\mathbf{S}_1^{(n+1)} = \left[\mathbf{S}_1^{(n)}, \mathbf{T}_{::n+1}\right]$. The matrix $\mathbf{S}_1^{(n)} \in \mathbb{R}^{I_1 \times I_2 w}$ and the possibly very large matrices $\mathbf{N}_1^{(n)} \in \mathbb{R}^{I_1 \times I_2 (n-w)}$ and $\mathbf{\Phi}_1^{(n)} \in \mathbb{R}^{I_2 n \times I_2 n}$, which grow with every slice update, are never stored during the updating process, making the algorithm memory-efficient as well. The algorithm is initialized by setting $\mathbf{L}_1^{(0)} = \gamma_1 \mathbf{I}$, $\mathbf{Q}_1^{(0)} = \mathbf{I}$, $\mathbf{N}_1^{(0)}$ empty and $\mathbf{S}_1^{(0)} = \gamma_1 \mathbf{I}$. In this way, γ_1 is implicitly set as splitting threshold.

The SURV updating algorithm applies hyperbolic rotations and Givens row and column rotations to maintain the lower-triangular structure of $\mathbf{L}_1^{(n)}$, while being careful that only stable hyperbolic rotations are used. The matrix $\mathbf{L}_1^{(n)}$ holds the weights of the columns in $\left[\mathbf{Q}_N^{(n)} | \mathbf{Q}_S^{(n)}\right]$ and is tracked alongside $\mathbf{Q}_1^{(n)}$. Using $\mathbf{L}_1^{(n)}$, the dimension of the subspace is automatically adjusted if necessary: subspace vectors that contribute to the subspace by more than the predefined threshold value γ_1 are moved to $\mathbf{Q}_S^{(n+1)} \in \mathbb{R}^{I_1 \times R_1^{(n+1)}}$, while the other vectors are kept in $\mathbf{Q}_N^{(n+1)} \in \mathbb{R}^{I_1 \times (I_1 - R_1^{(n+1)})}$. The matrix representation $\mathbf{Q}_S^{(n+1)}$ of the principal mode-1 subspace then takes the role of the mode-1 factor matrix $\mathbf{U}_1^{(n+1)}$ in the new LMLRA.

For mode 2, of which the dimension of the ambient space (I_2) does not evolve either, the goal is to update the mode-2 subspace up to a threshold value of γ_2 with all columns of the transposed new slice $\mathbf{T}_{::n+1}^{\mathrm{T}} \in \mathbb{R}^{I_2 \times I_1}$.

When a sliding window is used, every slice update is accompanied by a downdate of the oldest slice. This is done in order to keep the window length constant. For the non-evolving modes 1 and 2, downdating an old slice $\mathbf{T}_{::n-w}$ is very similar to updating a new slice. The columns of the old slice are appended to $\mathbf{N}_1^{(n)}$ instead of $\mathbf{S}_1^{(n)}$ and the decomposition is updated with SURV downdates [243], whereby one SURV downdate is performed for every column of $\mathbf{T}_{::n-w}$. In Algorithm 5.1, the update of the non-evolving modes is shown in pseudocode.

5.3.2 Subspace tracking of the evolving mode

For the non-evolving modes 1 and 2, adding a new slice to the tensor corresponds to the inclusion of new columns into the mode-n unfolding. As we are only concerned about the principal subspace of the unfolding, the specific ordering of the columns does not matter. Consequently, both non-evolving modes can be handled with the SURV updating method. For the mode-3 unfolding, however,

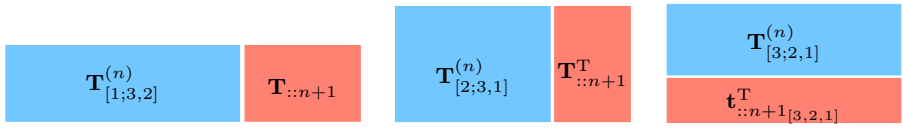

Figure 5.6: Appending a new slice to the tensor in the evolving mode 3 (right) corresponds to adding a new row to the mode-3 unfolding, while for the non-evolving modes 1 (left) and 2 (center), the new slice is added in the column dimension.

Algorithm 5.1: SURV updating of the non-evolving modes 1 and 2

function $[\mathbf{Q}_i^{(n+1)}, \mathbf{L}_i^{(n+1)}, R_i^{(n+1)}] =$ SURV_nonevol$(\mathbf{Q}_i^{(n)}, \mathbf{L}_i^{(n)}, R_i^{(n)}, \gamma_i, \mathbf{T}_{\text{new}}, \mathbf{T}_{\text{old}})$

Input : SURV matrices $\mathbf{Q}_i^{(n)}$, $\mathbf{L}_i^{(n)}$, rank values $R_i^{(n)}$ and thresholds γ_i for $i = 1, 2$, new slice $\mathbf{T}_{\text{new}}$, old slice $\mathbf{T}_{\text{old}}$

Output : SURV matrices $\mathbf{Q}_i^{(n+1)}$, $\mathbf{L}_i^{(n+1)}$ and rank values $R_i^{(n+1)}$ for $i = 1, 2$

for $i = 1, 2$ **do**

  % SURV update and downdate for mode-1 and 2

  for $j = 1, \ldots, I_i$ **do**

    $[\mathbf{Q}_{\text{up}}, \mathbf{L}_{\text{up}}, R_i^{(n+1)}] \leftarrow \texttt{SURVup}(\mathbf{Q}_i^{(n)}, \mathbf{L}_i^{(n)}, R_i^{(n)}, \gamma_i, \mathbf{T}_{\text{new}})$

    $[\mathbf{Q}_i^{(n+1)}, \mathbf{L}_i^{(n+1)}, R_i^{(n+1)}] \leftarrow \texttt{SURVdown}(\mathbf{Q}_{\text{up}}, \mathbf{L}_{\text{up}}, R_i^{(n+1)}, \gamma_i, \mathbf{T}_{\text{old}})$

  end

  % Extract factor matrices from $\mathbf{Q}_i$-matrices

  $\mathbf{U}_i^{(n+1)} \leftarrow \mathbf{Q}_i{}^{(n+1)}_{:,end-R_i^{(n+1)}+1:end}$

end

the new slice takes the form of one long vector $\mathbf{t}^{\mathrm{T}} := \mathbf{t}^{\mathrm{T}}_{::n+1[3,2,1]}$ appended to the bottom of $\mathbf{T}^{(n)}_{[3;2,1]}$, see Figure 5.6. A different approach is thus required for the evolving mode 3.

In many applications, one is mainly interested in tracking the principal row space of $\mathbf{T}^{(n)}_{[3;2,1]}$ of dimension R_3, which is assumed to be stationary or slowly-evolving [33]. For instance, consider the following 2D harmonic retrieval application. Two far-field signals are observed on a rectangular array, with one signal arriving via two different paths and the other arriving via a single path. In this case, a suitable LMLRA $\mathcal{T} \approx \mathcal{C} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$ would have $R_1 = R_2 = 3$ and $R_3 = 2$, with the first two modes corresponding to the spatial dimensions and the third mode to the time dimension. The subspaces $\mathrm{col}(\mathbf{U}_1)$ and $\mathrm{col}(\mathbf{U}_2)$ give the azimuth and elevation of the directions of arrival (DOAs)

and col($\mathbf{C}_{[2,1;3]}$) describes how the paths and transmitted signals are connected. Only these subspaces are slowly-varying and can thus be tracked. If needed, the transmitted signals at any time instance can be recovered up to rotation through the computation of the corresponding row of $\mathbf{U}_3$.

Figure 5.6 shows that the matrix $\mathbf{T}^{(n)}_{[3;2,1]}$ for which we want to track the stationary row space obtains a new row in every update. Consequently, if we transpose the problem, we track the stationary column space of a matrix that receives a new column in every update. Note that this is precisely the same setting as for modes 1 and 2. We can thus transpose the problem and apply the SURV approach as before to track the principal column space of $\mathbf{T}^{(n)}_{[2,1;3]} = \mathbf{T}^{(n)^{\mathrm{T}}}_{[3;2,1]}$, which corresponds to the principal row space of $\mathbf{T}^{(n)}_{[3;2,1]}$. The core tensor $\mathcal{C}^{(n)} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ can then easily be computed through $\mathbf{C}^{(n)}_{[2,1;3]} = \left(\mathbf{U}^{(n)^{\mathrm{T}}}_2 \otimes \mathbf{U}^{(n)^{\mathrm{T}}}_1\right) \mathbf{Q}_S \mathbf{L}_{22}$, with $\mathbf{Q}$ and $\mathbf{L}$ originating from the SURV decomposition of $\mathbf{T}^{(n)}_{[2,1;3]}$ and $\mathbf{L}_{22}$ referring to the bottom right $(R_3 \times R_3)$-block of $\mathbf{L}$. Together, $\mathbf{U}^{(n)}_1$, $\mathbf{U}^{(n)}_2$ and $\mathcal{C}^{(n)}$ describe the stationary subspace that is tracked: the matrix $\left(\mathbf{U}^{(n)}_2 \otimes \mathbf{U}^{(n)}_1\right) \mathbf{C}^{(n)}_{[2,1;3]}$ takes on a similar function as the matrix $\mathbf{U}$ in matrix subspace tracking methods, where one tracks a decomposition of the form $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$, as explained in the beginning of this section. Similarly, the factor matrix $\mathbf{U}^{(n)}_3$ of the LMLRA corresponds to the $\mathbf{V}$-matrix of the matrix methods. It is often unnecessary to explicitly form this matrix: if required, a row of $\mathbf{U}^{(n)}_3$ can be obtained by expressing the corresponding column of $\mathbf{T}^{(n)}_{[2,1;3]}$ in the basis generated by $\left(\mathbf{U}^{(n)}_2 \otimes \mathbf{U}^{(n)}_1\right) \mathbf{C}^{(n)}_{[2,1;3]}$.

A drawback of the SURV approach described above is that it is not particularly fast. The vectorized new slices have $I_1 I_2$ entries which makes updates and downdates computationally costly (one update has a cost of $\mathcal{O}(I_1^2 I_2^2)$ flop, see Section 5.3.3). A priori, there are two ways to decrease the computational cost. First, one can exploit the low multilinear rank to work with matrices with $R_1 R_2$ instead of $I_1 I_2$ rows and/or columns through the use of sequential truncation. Second, one can reduce the dimension from $I_1 I_2$ to w when $w < I_1 I_2$, as the mode-3 rank can never exceed the window length. Neither strategy is easy to combine with the SURV approach, though, because of the fact that the $\mathbf{\Phi}$-matrix is never stored. Let us make this more intuitive, without going into technical details. For the case of sequential truncation, the reduction from I_1 to R_1 brings no difficulties for mode 2: it simply reduces the number of columns to be appended. However, there is a deeper effect in mode-3, as the dimension of the ambient space changes from $I_1 I_2$ to $R_1 R_2$; this makes it more difficult to assess the accumulated error and, hence, hinders future decisions about increasing or decreasing the mode-3 rank or error bounds. In

the rest of this text, we therefore choose to track a two-sided decomposition instead of the SURV decomposition. In Section 5.5, we show that updates and downdates of this two-sided decomposition can be sped up very significantly by exploiting small window lengths ($w < I_1 I_2$), compressing the new slice and applying sequential truncation, in contrast to the SURV updates and downdates. Additionally, the two-sided decomposition gives us provable accuracy bounds for the approximation of $\mathcal{T}$ that it provides, see Section 5.4. Again, as we only need to determine the principal subspaces up to a certain noise threshold γ_3 and not the individual singular vectors, we use an RRURV decomposition to estimate the principal column and row spaces, as this is more efficient computationally than tracking an SVD. The remainder of this section discusses the basic method without compression and with $w \geq I_1 I_2$: we show how the RRURV decomposition of $\mathbf{T}^{(n)}_{[3;2,1]}$ can be updated and downdated for the addition and removal of rows, without yet making attempts to reduce the storage or computational costs, which is addressed in Section 5.5.

Denote by $\mathbf{Q}_3^{(n)}\mathbf{R}_3^{(n)}{\mathbf{V}_3^{(n)}}^{\mathrm{T}}$ the RRURV decomposition of $\mathbf{T}^{(n)}_{[3;2,1]}$ after update n, thus $\mathbf{Q}_3^{(n)} \in \mathbb{R}^{w \times I_1 I_2}$, is column-wise orthonormal, $\mathbf{R}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$ is upper triangular and $\mathbf{V}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$ is column-wise orthonormal. Using the mode-3 threshold value γ_3, the $\mathbf{R}$-matrix of the RRURV decomposition has the following form: $\mathbf{R}_3^{(n)} = \begin{bmatrix} \mathbf{E}_3^{(n)} & \mathbf{F}_3^{(n)} \\ \mathbf{0} & \mathbf{G}_3^{(n)} \end{bmatrix}$, with $\mathbf{E}_3^{(n)} \in \mathbb{R}^{R_3^{(n)} \times R_3^{(n)}}$ and $\nu = \sqrt{\|\mathbf{F}_3^{(n)}\|^2 + \|\mathbf{G}_3^{(n)}\|^2} \leq \gamma_3$. We can obtain the principal column space of $\mathbf{T}^{(n)}_{[3;2,1]}$ by taking the span of the R_3 leftmost columns of $\mathbf{Q}_3^{(n)}$, as the other columns will not contribute more than γ_3 to the norm of $\mathbf{T}^{(n)}_{[3;2,1]}$. Analogously, we can obtain the principal row space as the subspace spanned by the R_3 left-most columns of $\mathbf{V}_3^{(n)}$. Additionally, the mode-3 unfolding of the core tensor of the TMLUTVD can be obtained by computing $\mathbf{C}^{(n)}_{[3;2,1]} = \mathbf{R}^{(n)}_{3_{1:R_3,1:R_3}} {\mathbf{V}^{(n)}_{3_{:,1:R_3}}}^{\mathrm{T}} \left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right) \in \mathbb{R}^{R_3 \times R_1 R_2}$. An alternative strategy to obtain the core tensor is supplied in Appendix 5.B. Below, we give an approach to find the updated decomposition $\mathbf{Q}_3^{(n+1)}\mathbf{R}_3^{(n+1)}{\mathbf{V}_3^{(n+1)}}^{\mathrm{T}}$ after adding a new slice.

To include the information of the new row $\mathbf{t}^{\mathrm{T}}$ in the old decomposition $\mathbf{Q}_3^{(n)}\mathbf{R}_3^{(n)}{\mathbf{V}_3^{(n)}}^{\mathrm{T}}$, we can write

$$\begin{bmatrix} \mathbf{T}^{(n)}_{[3;2,1]} \\ \mathbf{t}^{\mathrm{T}} \end{bmatrix} \approx \begin{bmatrix} \mathbf{Q}_3^{(n)} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_3^{(n)} \\ \mathbf{x}^{\mathrm{T}} \end{bmatrix} {\mathbf{V}_3^{(n)}}^{\mathrm{T}} = \begin{bmatrix} \mathbf{Q}_3^{(n)}\mathbf{R}_3^{(n)}{\mathbf{V}_3^{(n)}}^{\mathrm{T}} \\ \mathbf{x}^{\mathrm{T}}{\mathbf{V}_3^{(n)}}^{\mathrm{T}} \end{bmatrix}. \tag{5.2}$$

In (5.2), the vector $\mathbf{x}^{\mathrm{T}}{\mathbf{V}_3^{(n)}}^{\mathrm{T}} \in \mathbb{R}^{I_1 I_2}$ is exactly equal to the new row $\mathbf{t}^{\mathrm{T}}$. The new $\mathbf{R}$-matrix can be brought back to upper triangular form by zeroing the

entries of $\mathbf{x}^{\mathrm{T}}$ one by one with the use of Givens row and column rotations without destroying the rank-revealing structure, following the scheme from [197]. For the rightmost $(I_1 I_2 - R_3 - 1)$ entries of the bottom row of the $\mathbf{R}$-matrix, Givens column rotations are used to zero the entries from right to left, with each Givens column rotation followed by a Givens row rotation to erase the resulting fill-in. Afterwards, the leftmost $(R_3 + 1)$ entries of the bottom row are zeroed from left to right using Givens row rotations. All row and column rotations should be absorbed in the new $\mathbf{Q}$- and $\mathbf{V}$-matrices, respectively, to obtain a new URV decomposition $\mathbf{Q}_{\mathrm{up}}\mathbf{R}_{\mathrm{up}}\mathbf{V}_{\mathrm{up}}^{\mathrm{T}}$ of $\mathbf{T}_{[3;2,1]}^{(n+1)}$, wherein $\mathbf{Q}_{\mathrm{up}} \in \mathbb{R}^{(w+1)\times(I_1 I_2+1)}$, $\mathbf{R}_{\mathrm{up}} \in \mathbb{R}^{(I_1 I_2+1)\times I_1 I_2}$ and $\mathbf{V}_{\mathrm{up}} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$.

After the update, a downdate of the mode-3 subspace can be performed with QR-downdates [73,87]. Starting from the updated URV decomposition $\mathbf{Q}_{\mathrm{up}}\mathbf{R}_{\mathrm{up}}\mathbf{V}_{\mathrm{up}}^{\mathrm{T}}$, we apply elementary rotations to zero the first row of $\mathbf{Q}_{\mathrm{up}}$. Applying the inverse rotations to $\mathbf{R}_{\mathrm{up}}$ then gives us the decomposition

$$\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{\mathrm{dn}} \end{bmatrix} \begin{bmatrix} \mathbf{y}^{\mathrm{T}} \\ \mathbf{R}_{\mathrm{dn}} \end{bmatrix} \mathbf{V}_{\mathrm{up}}^{\mathrm{T}}.$$

The matrix $\mathbf{V}_{\mathrm{up}}$ is unchanged, so by setting $\mathbf{V}_{\mathrm{dn}} = \mathbf{V}_{\mathrm{up}}$, $\mathbf{Q}_{\mathrm{dn}}\mathbf{R}_{\mathrm{dn}}\mathbf{V}_{\mathrm{dn}}^{\mathrm{T}}$ is a URV decomposition of $\mathbf{T}_{[3;2,1]}^{(n+1)}$ after downdating its first row.

Both the update and the downdate preserve the URV-form of the decomposition and the small values in the right-most columns of $\mathbf{R}_3^{(n)}$. Still, an increase or decrease by one of the rank of the $\mathbf{R}$-matrix might be in order (adding or removing a row can only increase or decrease the rank by one, respectively). The non-rank-revealing URV decomposition $\mathbf{Q}_{\mathrm{dn}}\mathbf{R}_{\mathrm{dn}}\mathbf{V}_{\mathrm{dn}}^{\mathrm{T}}$ with $\mathbf{R}_{\mathrm{dn}} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$ can be brought back into RRURV form with the method from [197]. This method uses an iterative approach, wherein the matrix $\mathbf{R}_{\mathrm{dn}}$ is deflated until the requested rank-revealing upper triangular form is obtained. The required Givens rotations for the deflation are absorbed into the matrices $\mathbf{U}_{\mathrm{dn}}$ and $\mathbf{V}_{\mathrm{dn}}$ to obtain the downdated URV decomposition $\mathbf{Q}_3^{(n+1)}\mathbf{R}_3^{(n+1)}{\mathbf{V}_3^{(n+1)}}^{\mathrm{T}}$. In our case, at most one deflation step is required to restore the rank-revealing property. A high-level overview of the procedure for the evolving mode 3 is given as Algorithm 5.2.

5.3.3 Algorithm complexity

Ideally, an updating method has both a lower computational and storage cost than its corresponding batch algorithm, as computation time and storage space are typically limited in tracking applications. Below, we show that while the basic algorithm from Sections 5.3.1 and 5.3.2 has a provably upper bounded

Algorithm 5.2: RRURV tracking for the evolving mode

1: function $[\mathbf{Q}_3^{(n+1)}, \mathbf{R}_3^{(n+1)}, \mathbf{V}_3^{(n+1)}, \mathbf{U}_3^{(n+1)}, R_3^{(n+1)}] = \text{RRURV_evol}(\mathbf{Q}_3^{(n)}, \mathbf{R}^{(n)}, \mathbf{V}_3^{(n)}, R_3^{(n)}\ \gamma_3, \mathbf{t})$
2: **Input** : RRURV matrices $\mathbf{Q}_3^{(n)}$, $\mathbf{R}_3^{(n)}$, $\mathbf{V}_3^{(n}$, rank $R_3^{(n)}$, threshold γ_3, vectorized new slice $\mathbf{t}$
3: **Output** : RRURV matrices $\mathbf{Q}_3^{(n+1)}$, $\mathbf{R}_3^{(n+1)}$, $\mathbf{V}_3^{(n+1)}$, rank $R_3^{(n)}$, mode-3 factor matrix $\mathbf{U}_3^{(n+1)}$
4: % RRURV update and QR downdate
5: $[\mathbf{Q}_{\text{up}}, \mathbf{R}_{\text{up}}, \mathbf{V}_{\text{up}}] \leftarrow \texttt{URV_up}(\mathbf{Q}_3^{(n)}, \mathbf{R}_3^{(n)}, \mathbf{V}_3^{(n)}, \mathbf{t})$
6: $[\mathbf{Q}_{\text{dn}}, \mathbf{R}_{\text{dn}}] \leftarrow \texttt{QR_down}(\mathbf{Q}_{\text{up}}, \mathbf{R}_{\text{up}})$
7: % Restore rank-revealing structure and extract $\mathbf{U}_3^{(n+1)}$
8: $[\mathbf{Q}_3^{(n+1)}, \mathbf{R}_3^{(n+1)}, \mathbf{V}_3^{(n+1)}, R_3^{(n+1)}] \leftarrow \texttt{deflate}(\mathbf{Q}_{\text{dn}}, \mathbf{R}_{\text{dn}}, \mathbf{V}_{\text{up}}, R_3^{(n)}, \gamma_3)$
9: $\mathbf{U}_3^{(n+1)} \leftarrow \mathbf{Q}_{3:,1:R_3}^{(n+1)}$

approximation error, its memory and computational complexity do not yet offer significant reductions compared to the batch algorithm. Therefore, in Section 5.5, further optimizations are discussed to dramatically reduce both these complexities.

Storage cost

The tensor that is to be decomposed has $I_1 I_2 w$ entries. For large values of I_1, I_2 and w, storing the windowed tensor can become costly. For finite window lengths, the proposed updating method essentially still requires all slices within the sliding window to be stored, as these slices are used to downdate the subspaces in later iterations. As a slice is only downdated after w updates, though, it can be stored in non-local memory. Naturally, all $I_1 I_2$ entries of the new slice have to be stored as well.

For the SURV updating of a non-evolving mode, say mode 1, only the matrices $\mathbf{Q}_1^{(n)} \in \mathbb{R}^{I_1 \times I_1}$ and $\mathbf{R}_1^{(n)} \in \mathbb{R}^{I_1 \times I_1}$ have to be stored, totaling $\mathcal{O}(I_1^2)$ entries (their triangular or orthonormal structures can be exploited to save storage space). Updating of the evolving mode 3 requires the storage of the matrices $\mathbf{Q}_3^{(n)} \in \mathbb{R}^{w \times I_1 I_2}$, $\mathbf{R}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$ and $\mathbf{V}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$, with $\mathcal{O}(w I_1 I_2 + I_1^2 I_2^2)$ entries in total. For the three modes together, $\mathcal{O}(w I_1 I_2 + I_1^2 I_2^2)$ entries have to be stored, not taking the slices in non-local memory into account, which are stored until they are eventually downdated. Since no compression has been considered, this is not smaller than the $\mathcal{O}(I_1 I_2 w)$ entries of the full tensor. In Section 5.5, we will consider strategies to reduce the storage cost.

Table 5.1: Memory and computational complexity for the batch TMLSVD method and the proposed updating method. Superscripts are dropped and $R = R_1^{(n)} = R_2^{(n)} = R_3^{(n)}$ is assumed.

Memory		
Batch		$\mathcal{O}(I_1 I_2 w)$
Updating		
	New slice	$\mathcal{O}(I_1 I_2)$
	Slices in window	$\mathcal{O}(w I_1 I_2)$
	Modes 1 and 2	$\mathcal{O}(I_1^2 + I_2^2)$
	mode 3	$\mathcal{O}(I_1^2 I_2^2 + w I_1 I_2)$
Computation		
Batch		$\mathcal{O}(I_1 I_2 w \min(I_1, I_2) + I_1 I_2 w R)$
Updating		
	Modes 1 and 2	$\mathcal{O}(I_1^2 I_2 + I_2^2 I_1)$
	mode 3	$\mathcal{O}(I_1 I_2 (w + I_1 I_2))$
	Core	$\mathcal{O}(R^2 I_1 I_2)$

Computational complexity

For mode 1, the complexity of one subspace update is similar to that of a single QR-update ($\mathcal{O}(I_1^2)$ flop). For comparision, an update of the SVD would require $\mathcal{O}(I_1^3)$ flop [197]. Using potentially cheaper $\mathcal{O}(I_1 R_1^k)$ flop algorithms would require the rank to be fixed and known in advance. Of course, every new slice adds I_2 columns to $\mathbf{T}^{(n)}_{[1;3,2]}$, while removing one slice corresponds to removing I_2 columns from $\mathbf{T}^{(n)}_{[1;3,2]}$, so the total complexity of one slice update or downdate for mode 1 is $\mathcal{O}(I_1^2 I_2)$.

For the evolving mode 3, the update consists of computing the vector $\mathbf{x}^{\mathrm{T}} = \mathbf{t}^{\mathrm{T}}\mathbf{V}$ and triangularizing the $\mathbf{R}$-matrix, which require $\mathcal{O}(I_1^2 I_2^2)$ and $\mathcal{O}(w I_1 I_2 + I_1^2 I_2^2)$ flop, respectively. The complexity of the QR downdate is $\mathcal{O}(w I_1 I_2)$ flop (the matrix $\mathbf{V}_{\mathrm{up}}$ is not involved in the downdate). Finally, making the URV decomposition rank-revealing requires one deflation step, at a cost of $\mathcal{O}(I_1^2 I_2^2)$ flop [73, 87]. If the core tensor is required, it can be computed in $\mathcal{O}(I_1 I_2 R^2)$ flop, assuming $R = R_1^{(n)} = R_2^{(n)} = R_3^{(n)}$. The total complexity is thus $\mathcal{O}(I_1 I_2 (w + I_1 I_2))$ flop. The results from this section are summarized in Table 5.1.

5.3.4 Choosing the threshold values γ_i

The SURV decomposition adjusts the dimension of the subspace in the ith mode automatically through the value of the threshold γ_i. Whenever a singular value drops below this threshold, its corresponding vector is removed from the subspace and the dimension of the subspace is decreased. Depending on the expected noise level of the tensor, the value of γ_i has to be chosen accordingly: the higher the noise, the larger the singular value threshold should be set. For each mode, we consider the mode-i unfolding of a tensor, so we can use results from matrix subspace tracking to set γ_i. Zhou et al. [243] propose to set the threshold value equal to the estimated size of the largest singular value of a matrix with i.i.d. zero-mean Gaussian distributed entries, multiplied by a scaling factor $\alpha > 1$ to avoid false positives. When the contribution of a subspace vector exceeds this threshold, the vector is considered to be part of the principal subspace. This leads to a threshold value of $\gamma_i = \alpha\sigma(\sqrt{m} + \sqrt{n})$ [167], wherein σ is an estimate of the standard deviation of the entry-wise noise and m and n are the dimensions of the mode-i unfolding of $\mathcal{T}$ after applying the sliding window. For example, for mode 1, $m = I_1$ and $n = wI_2$. The same type of threshold can be set for the evolving mode 3. In this case $m = w$ and $n = I_1 I_2$.

5.4 Error bounds for the TMLUTVD

The truncation of the MLUTVD by discarding the information in the non-principal subspaces generally introduces an error. As for the MLSVD, however, we can bound this truncation error by looking at the norms of the parts of the triangular matrices that are discarded. Below, we derive such bounds for both the Frobenius and spectral norms.

5.4.1 Frobenius norm

In Figure 5.7, it can be seen that the squared Frobenius norm of the TMLUTVD error can be bounded by the sum of the squared Frobenius norms of the errors that are introduced by the truncations in the different tensor modes [47]. For the TMLUTVD, these mode-wise errors are introduced by truncating the SURV (for modes 1 and 2) and the RRURV (for mode 3) decompositions. Writing ϵ_{F}

for the approximation error $\left\|\mathcal{T}^{(n)} - \hat{\mathcal{T}}^{(n)}\right\|_{\mathrm{F}}^2$, we have:

$$\begin{aligned}
\epsilon_{\mathrm{F}} &= \left\|\mathcal{T}^{(n)} - \mathcal{S}^{(n)} \cdot_1 \mathbf{U}_1^{(n)} \cdot_2 \mathbf{U}_2^{(n)} \cdot_3 \mathbf{U}_3^{(n)}\right\|_{\mathrm{F}}^2 \\
&= \left\|\mathcal{T}^{(n)} - \left(\mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n)^{\mathrm{T}}} \cdot_3 \mathbf{U}_3^{(n)^{\mathrm{T}}}\right) \cdot_1 \mathbf{U}_1^{(n)} \cdot_2 \mathbf{U}_2^{(n)} \cdot_3 \mathbf{U}_3^{(n)}\right\|_{\mathrm{F}}^2 \\
&= \left\|\mathcal{T}^{(n)} - \mathcal{T}^{(n)} \cdot_1 \left(\mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\right) \cdot_2 \left(\mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\right) \cdot_3 \left(\mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\right)\right\|_{\mathrm{F}}^2 \\
&\leq \left\|\mathbf{T}_{[1;3,2]}^{(n)} - \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\mathbf{T}_{[1;3,2]}^{(n)}\right\|_{\mathrm{F}}^2 + \left\|\mathbf{T}_{[2;3,1]}^{(n)} - \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\mathbf{T}_{[2;3,1]}^{(n)}\right\|_{\mathrm{F}}^2 \\
&\quad + \left\|\mathbf{T}_{[3;2,1]}^{(n)} - \mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\mathbf{T}_{[3;2,1]}^{(n)}\right\|_{\mathrm{F}}^2. \qquad (5.3)
\end{aligned}$$

When considering only the estimated principal subspace, both the SURV and the RRURV decompositions guarantee that the largest singular value of the truncated part in mode i does not exceed the specified threshold γ_i. This immediately leads to a bound for the TMLUTVD error in the Frobenius norm:

$$\epsilon_{\mathrm{F}} \leq (I_1 - R_1)\,\gamma_1^2 + (I_2 - R_2)\,\gamma_2^2 + (w - R_3)\,\gamma_3^2. \qquad (5.4)$$

The above bound applies to the worst-case scenario, where all singular values of the truncated part are equal to the threshold. In practice, the exact error of the SURV and RRURV decompositions can be computed from the $\mathbf{L}_i$ and $\mathbf{R}$-matrices, respectively. The following theorem gives an expression for $\left\|\mathbf{T}_{[1;3,2]}^{(n)} - \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\mathbf{T}_{[1;3,2]}^{(n)}\right\|_{\mathrm{F}}^2$, the error that is introduced by discarding the orthogonal complement of the principal subspace in the SURV decomposition for mode 1.

Theorem 5.4.1: Approximation bound for non-evolving modes
Let $\mathbf{Q}_1\mathbf{L}_1\mathbf{J}\mathbf{\Phi}_1^{\mathrm{T}}$ be an SURV decomposition of $\begin{bmatrix}\mathbf{N}_1 & \mathbf{T}_{[1;3,2]}\end{bmatrix}\mathbf{J}$. Split $\mathbf{Q}_1 \in \mathbb{R}^{I_1 \times I_1}$ into $\mathbf{Q}_1 = [\mathbf{Q}_N \;\; \mathbf{Q}_S]$, wherein $\mathbf{Q}_S \in \mathbb{R}^{I_1 \times R_1}$ is a basis for the principal column space and $\mathbf{Q}_N \in \mathbb{R}^{I_1 \times (I_1 - R_1)}$ is a basis for the orthogonal complement. Partition $\mathbf{\Phi}_1$ into $\mathbf{\Phi}_1 = \begin{bmatrix}\mathbf{\Phi}_{11} & \mathbf{\Phi}_{12} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_{22}\end{bmatrix}$ such that $\mathbf{\Phi}_{11}$ and $\mathbf{\Phi}_{22}$ are square and their dimensions match the column dimensions of $\mathbf{N}_1$ and $\mathbf{T}_{[1;3,2]}$, respectively. Finally, partition $\mathbf{L}_1$ into $\mathbf{L}_1 = \begin{bmatrix}\mathbf{L}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{0} & \mathbf{L}_{22} & \mathbf{0}\end{bmatrix}$, with $\mathbf{L}_{11}$ and $\mathbf{L}_{22}$ square, such that it matches the structure of $\mathbf{Q}_1$ and $\mathbf{\Phi}_1$. Then

$$\left\|\mathbf{T}_{[1;3,2]} - \mathbf{Q}_S\mathbf{Q}_S^{\mathrm{T}}\mathbf{T}_{[1;3,2]}\right\|_{\mathrm{F}}^2 = \gamma_1^2(I_1 - R_1) - \|\mathbf{L}_{11}\|_{\mathrm{F}}^2\,.$$

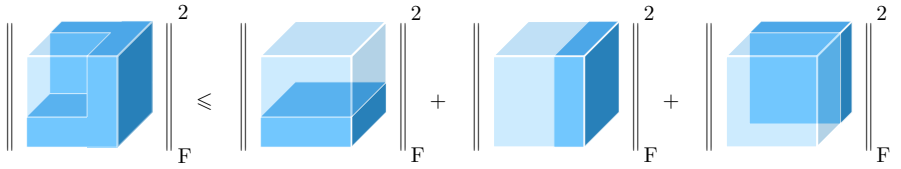

Figure 5.7: The squared Frobenius norm of the TMLUTVD error is upper-bounded by the sum of the truncation errors in the different modes. As discussed in Section 5.4.2, a similar inequality holds for the spectral norm.

Proof. Can be found in Appendix 5.A. □

In the above expression, $\|\mathbf{L}_{11}\|_{\mathrm{F}}^2$ is the squared Frobenius norm of the part that is truncated from the SURV decomposition. This quantity is equal to the sum of the squares of the singular values σ_l, for $l = 1, 2 \ldots, (I_1 - R_1)$, of $\mathbf{L}_{11}$. However, the SURV considers a shifted decomposition, i.e., it looks at the eigenvalues of $\mathbf{T}_{[1;3,2]}\mathbf{T}_{[1;3,2]}^{\mathrm{T}} - \mathbf{N}_1\mathbf{N}_1^{\mathrm{T}}$, instead of those of $\mathbf{T}_{[1;3,2]}\mathbf{T}_{[1;3,2]}^{\mathrm{T}}$. Consequently, the true squared singular values of the truncated part are $\gamma_1^2 - \sigma_l^2$, for $l = 1, 2 \ldots, (I_1 - R_1)$, leading to the result from Theorem 5.4.1.

For mode 2, exactly the same approach can be used as for mode 1. Consequently, a similar expression can be derived as in Theorem 5.4.1 for the error that is introduced by discarding the orthogonal complement of the principal subspace in mode 2. For mode 3, we track an RRURV decomposition $[\mathbf{Q}_S \;\; \mathbf{Q}_N] \begin{bmatrix} \mathbf{E}_3^{(n)} & \mathbf{F}_3^{(n)} \\ \mathbf{0} & \mathbf{G}_3^{(n)} \end{bmatrix} \mathbf{V}^{(n)^{\mathrm{T}}}$. As the RRURV decomposition does not apply the SURV shift, the error $\left\| \mathbf{T}_{[3;2,1]}^{(n)} - \mathbf{Q}_S^{(n)} \mathbf{Q}_S^{(n)^{\mathrm{T}}} \mathbf{T}_{[3;2,1]}^{(n)} \right\|_{\mathrm{F}}^2$ is simply equal to $\left\| \mathbf{Q}_N^{(n)} \mathbf{G}_3^{(n)} \mathbf{V}^{(n)^{\mathrm{T}}} \right\|_{\mathrm{F}}^2 = \left\| \mathbf{G}_3^{(n)} \right\|_{\mathrm{F}}^2$, the part of the R-matrix at the bottom right that is associated with the orthogonal complement of the principal subspace. Combining the bounds for the different modes gives a tighter upper bound than (5.4) for ϵ_{F}:

$$\epsilon_{\mathrm{F}} \leq \gamma_1^2 (I_1 - R_1) - \left\| \mathbf{L}_{1_{11}}^{(n)} \right\|_{\mathrm{F}}^2 + \gamma_2^2 (I_2 - R_2) - \left\| \mathbf{L}_{2_{11}}^{(n)} \right\|_{\mathrm{F}}^2 + \left\| \mathbf{G}_3^{(n)} \right\|_{\mathrm{F}}^2,$$

wherein $\mathbf{L}_1$ and $\mathbf{L}_2$ refer to the $\mathbf{L}$-matrices in mode 1 and mode 2, respectively.

5.4.2 Spectral norm

Aside from the bound in the Frobenius norm, we can also derive a spectral norm bound for the TMLUTVD error. The spectral norm of a tensor $\mathcal{A}$ is defined as

$$\|\mathcal{A}\|_2 = \max_{\|\mathbf{x}\|=\|\mathbf{y}\|=\|\mathbf{z}\|=1} \mathcal{A} \cdot_1 \mathbf{x} \cdot_2 \mathbf{y} \cdot_3 \mathbf{z}.$$

Using the triangle inequality, we can write the norm of the approximation error $\epsilon_2 = \left\|\mathcal{T}^{(n)} - \hat{\mathcal{T}}^{(n)}\right\|_2$ of the TMLUTVD in function of the error in the different modes:

$$\begin{aligned}
\epsilon_2 &= \left\|\mathcal{T}^{(n)} - \mathcal{S}^{(n)} \cdot_1 \mathbf{U}_1^{(n)} \cdot_2 \mathbf{U}_2^{(n)} \cdot_3 \mathbf{U}_3^{(n)}\right\|_2 \\
&= \left\|\mathcal{T}^{(n)} - \mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}} \cdot_3 \mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\right\|_2 \\
&\leq \left\|\mathcal{T}^{(n)} - \mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\right\|_2 + \left\|\mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}} - \mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\right\|_2 \\
&\quad + \left\|\mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}} - \mathcal{T}^{(n)} \cdot_1 \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}} \cdot_3 \mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\right\|_2.
\end{aligned}$$

As the spectral norm of a tensor is upper-bounded by the spectral norm of its unfoldings [97], we can write:

$$\begin{aligned}
\epsilon_2 &\leq \left\|\mathbf{T}_{[1;3,2]}^{(n)} - \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\mathbf{T}_{[1;3,2]}^{(n)}\right\|_2 \\
&\quad + \left\|\left(\mathbf{T}_{[2;3,1]}^{(n)} - \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\mathbf{T}_{[2;3,1]}^{(n)}\right)\left(\mathbf{I}_{I_3} \otimes \left[\mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\right]\right)\right\|_2 \\
&\quad + \left\|\left(\mathbf{T}_{[3;2,1]}^{(n)} - \mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\mathbf{T}_{[3;2,1]}^{(n)}\right)\left(\left[\mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\right] \otimes \left[\mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\right]\right)\right\|_2 \\
&\leq \left\|\mathbf{T}_{[1;3,2]}^{(n)} - \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}\mathbf{T}_{[1;3,2]}^{(n)}\right\|_2 + \left\|\mathbf{T}_{[2;3,1]}^{(n)} - \mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}\mathbf{T}_{[2;3,1]}^{(n)}\right\|_2 \\
&\quad + \left\|\mathbf{T}_{[3;2,1]}^{(n)} - \mathbf{U}_3^{(n)}\mathbf{U}_3^{(n)^{\mathrm{T}}}\mathbf{T}_{[3;2,1]}^{(n)}\right\|_2 \\
&\leq \gamma_1 + \gamma_2 + \gamma_3.
\end{aligned}$$

The second inequality above holds because of the submultiplicativity of the spectral matrix norm, i.e., $\mathbf{I}_{I_1}\ \mathbf{U}_1^{(n)}\mathbf{U}_1^{(n)^{\mathrm{T}}}$ and $\mathbf{U}_2^{(n)}\mathbf{U}_2^{(n)^{\mathrm{T}}}$ are orthogonal

projectors, so they have spectral norm 1 and products with these matrices cannot increase the norm. For the spectral norm, the shift of the SURV decomposition does not affect the bound, as it only shifts each squared singular value σ^2 to $\gamma_i^2 - \sigma^2$. The result is that for SURV the maximal error γ_i is achieved when a particular singular value σ_l of $\mathbf{L}_{i11}$ is equal to zero, while for the RRURV decomposition, it occurs when $\sigma_r = \gamma_i$, with σ_r a singular value of $\mathbf{G}_3$. Like the (squared) Frobenius norm in (5.3), the spectral norm of the TMLUTVD error can thus also be bounded by the sum of the spectral norms of the truncated parts in the three tensor modes.

5.5 Strategies to reduce the complexity

In the previous sections, we outlined the basic TMLUTVD updating algorithm. While it allows us to obtain a new TMLUTVD from the previous decomposition with provably bounded approximation error, it is not very efficient with respect to memory and computational cost. Below, we provide three strategies to improve the efficiency of the algorithm. The first strategy can be used when the window length is small compared to the product of the dimensions of the non-evolving modes 1 and 2 and comes at no cost to the accuracy of the method. The second strategy is to first compute a low-rank approximation of the new slice (details are omitted). Using this decomposition instead of the full slice can greatly reduce the number of SURV updates that have to be performed; the bounds from Section 5.4 can be adapted to include the approximation error of the new slice. The final strategy is to truncate sequentially, i.e., immediately after processing each mode, like for the STMLSVD. With this strategy, modes that are processed later, such as the evolving mode 3, require less work than when truncation is performed after all mode-n subspaces have been computed. If the mode-1 and mode-2 subspaces are stationary, the bounds of Section 5.4 can be modified similarly to how the MLSVD bounds [47] are adapted to the STMLSVD [217]. If they are non-stationary, however, the situation is more involved, as previously truncated information cannot be recovered if it is needed later (within the next $(w-1)$ updates). Thus, potentially, the bounds form Section 5.4 need to be increased to account for this information loss. If the ambient space is slowly-varying, though, the information loss is minor and the accuracy of the method is generally not significantly impacted, see Section 5.6. In Table 5.2, the storage cost and computational complexity are shown when the strategies from this section are used.

5.5.1 Exploiting small window lengths

If $w < I_1 I_2$, tracking an ambient space of dimensions $I_1 I_2$ for the evolving mode 3 is unnecessary, as the dimension of the mode-3 subspace can never exceed w. Instead, one can consider an ambient space of dimension w and extract the principal mode-3 subspace of dimension R_3 from it without loss of accuracy. More concretely, one can track an RRURV decomposition $\mathbf{Q}_3^{(n)} \mathbf{R}_3^{(n)} {\mathbf{V}_3^{(n)}}^{\mathrm{T}}$ of the mode-3 unfolding of $\mathcal{T}^{(n)}$, with $\mathbf{Q}_3^{(n)} \in \mathbb{R}^{w \times w}$, $\mathbf{R}_3^{(n)} \in \mathbb{R}^{w \times w}$ and $\mathbf{V}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times w}$. As the window length w is smaller than $I_1 I_2$ in this case, the computational cost of updating and downdating this decomposition is reduced by the use of the smaller RRURV matrices. The only drawback is that the rank-revealing structure of the $\mathbf{R}$-matrix can be lost and up to $R_3^{(n)}$ deflation steps may be required to restore this structure. This approach leads to a computational complexity for the evolving mode of $\mathcal{O}(w(wR_3 + I_1 I_2))$ flop: the update and downdate require $\mathcal{O}(wI_1 I_2)$ and $\mathcal{O}(w^2)$ flop, respectively, when using the smaller RRURV matrices and restoring the rank-revealing structure with at most R_3 deflation steps on a $(w \times w)$-matrix costs $\mathcal{O}(w^2 R_3)$ flop. A more detailed exposition is given below.

Update After adding a new row $\mathbf{t}^{\mathrm{T}}$ to the unfolded tensor $\mathbf{T}_{[3;2,1]}^{(n)}$, one can explicitly construct the following decomposition of the augmented mode-3 unfolding of $\mathcal{T}$:

$$\mathbf{T}_{[3;2,1]}^{(n+1)} = \begin{bmatrix} \mathbf{T}_{[3;2,1]}^{(n)} \\ \mathbf{t}^{\mathrm{T}} \end{bmatrix} \approx \begin{bmatrix} \mathbf{Q}_3^{(n)} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_3^{(n)} & \mathbf{0} \\ \mathbf{x}^{\mathrm{T}} & y \end{bmatrix} \begin{bmatrix} {\mathbf{V}_3^{(n)}}^{\mathrm{T}} \\ \mathbf{v}^{\mathrm{T}} \end{bmatrix}. \tag{5.5}$$

The vector $\mathbf{x}^{\mathrm{T}}$ from (5.5) is obtained as the coefficients of the projection of the new row $\mathbf{t}^{\mathrm{T}}$ on the space spanned by the rows of ${\mathbf{V}_3^{(n)}}^{\mathrm{T}}$. The vector $y\mathbf{v}^{\mathrm{T}}$, with $\|\mathbf{v}\| = 1$ holds the part of $\mathbf{t}^{\mathrm{T}}$ that lies in the orthogonal complement. Writing $\mathbf{V}_{\mathrm{up}} = \begin{bmatrix} \mathbf{V}_3^{(n)} & \mathbf{v} \end{bmatrix} \in \mathbb{R}^{I_1 I_2 \times (w+1)}$, one obtains:

$$\mathbf{T}_{[3;2,1]}^{(n+1)} \approx \begin{bmatrix} \mathbf{Q}_3^{(n)} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_3^{(n)} & \mathbf{0} \\ \mathbf{x}^{\mathrm{T}} & y \end{bmatrix} \mathbf{V}_{\mathrm{up}}^{\mathrm{T}}.$$

The new middle matrix can overall be brought back to upper-triangular form by zeroing the entries in its last row $\begin{bmatrix} \mathbf{x}^{\mathrm{T}} & y \end{bmatrix}$ from left to right with the use of Givens row rotations [197]. The inverse rotations are applied as column rotations to the left-most matrix to obtain the new URV decomposition $\mathbf{Q}_{\mathrm{up}} \mathbf{R}_{\mathrm{up}} \mathbf{V}_{\mathrm{up}}^{\mathrm{T}}$.

Downdate The downdate of the oldest row of $\mathbf{T}^{(n)}_{[3;2,1]}$ does not depend on the structure of the $\mathbf{V}$-matrix, so we can proceed as in Section 5.3.2 with a QR downdate [87]. Afterwards, the rank-revealing structure of the $\mathbf{R}$-matrix is restored with deflation steps. The new $\mathbf{R}$- and $\mathbf{V}$-matrices now have $(w+1)$ columns instead of w, so after eliminating one column, we obtain the new URV decomposition $\mathbf{Q}_3^{(n+1)}\mathbf{R}_3^{(n+1)}{\mathbf{V}_3^{(n+1)}}^{\mathrm{T}}$. As before, the first $R_3^{(n+1)}$ columns of $\mathbf{Q}_3^{(n+1)}$ form a basis for the new mode-3 subspace and are thus used to construct $\mathbf{U}_3^{(n+1)} \in \mathbb{R}^{w \times R_3^{(n+1)}}$.

5.5.2 Compressing the new slice

To update the mode-1 subspace of the tensor with an entire new slice $\mathbf{T}_{::n+1}$, I_2 SURV updates are required: one for each column of $\mathbf{T}_{::n+1}$. If I_2 is large, this can make a slice update computationally costly, even though the SURV algorithm itself is quite efficient, see Section 5.3.3. However, if the mode-2 rank $R_2^{(n)}$ of the LMLRA from the previous update is small compared to I_2, it is reasonable to expect the new slice to have low column rank as well. In this case, it can be computationally efficient to first compute a low-rank approximation of rank $p^{(n+1)}$ of the new slice $\mathbf{T}_{::n+1}$ before applying the SURV updates. Instead of applying an SURV update for all I_2 columns of $\mathbf{T}_{::n+1}$, one can thus update the mode-1 subspace with the $p^{(n+1)}$ columns of the low-rank approximation. The lower number of SURV updates more than compensates for the one-time $\mathcal{O}(I_1 I_2 p^{(n+1)})$ flop cost of computing the approximation. Using this approach, the complexity for the mode-1 update drops from $\mathcal{O}(I_1^2 I_2)$ to $\mathcal{O}(I_1^2 p^{(n+1)})$ flop. When $p^{(n+1)} = \mathcal{O}(R_1^{(n+1)})$ is small compared to I_2, this offers a considerable efficiency gain. The noise threshold for the low-rank approximation can be set as $\epsilon = \delta\gamma_1$, with $\delta < 1$, in accordance with the SURV threshold value γ_1. By choosing $\delta = 0.1$, for instance, subspace vectors that do not lie in the principal mode-1 subspace of the new slice, but that may be significant enough after accumulation over multiple slice updates, are not immediately discarded. The same low-rank approximation can be used for the update of the SURV in mode 2.

The low-rank approximation of the slice can also be used to make the downdating step less computationally costly. For downdating, old slices have to be stored until they are dropped from the sliding window. This is problematic, as this means that at any point, w $(I_1 \times I_2)$-matrices have to be stored, which can be expensive. Fortunately, we can reuse the low-rank approximation of the old slice that was computed w time steps before during the updating process. This approach can significantly lower the memory requirements of the method, as only a rank-$p^{(n+1)}$ approximation of each slice is stored for w time steps. This

rank-$p^{(n+1)}$ approximation $\hat{\mathbf{T}}_{::n-w}$ is then used to downdate the mode-1 and mode-2 subspaces when the slice is eventually dropped from the sliding window. Using only the $p^{(n+1)}$ columns of the low-rank approximation directly lowers the computational cost of the slice downdates. There is no overhead of the computation of the approximation, as it is already available.

5.5.3 Sequential truncation

Instead of independently computing the principal subspaces for all modes and then truncating the decomposition, we can also truncate immediately after updating each mode, as for the STMLSVD [217]. This has the advantage that the new slice becomes smaller as more modes are processed and the computational cost for the later processed modes is decreased. Below, we explain how sequential truncation can be applied to the TMLUTVD updating process. Using this approach and the compression from Section 5.5.2, the total complexity of the TMLUTVD update can be reduced to $\mathcal{O}(R^2(\min(R^4, w^2) + I^2R))$ flop, where $R = R_1 = R_2 = R_3$ and $i = I_1 = I_2$ are assumed, see Table 5.2.

Non-evolving modes 1 and 2

After updating the mode-1 subspace, the newly computed principal subspace $\operatorname{col}\left(\mathbf{U}_1^{(n+1)}\right)$ can be applied to reduce the computational cost of the subspace updates in the following modes. With a matrix projection of the new slice onto the mode-1 principal subspace, the number of columns of $\mathbf{T}_{::n+1}^{\mathrm{T}}$ can be reduced from I_1 to $R_1^{(n+1)}$. We compute $\mathbf{T}_{::n+1}^{\mathrm{T}}\mathbf{U}_1^{(n+1)} \in \mathbb{R}^{I_2 \times R_1^{(n+1)}}$ and the $R_1^{(n+1)}$ columns of the resulting matrix are then used to perform $R_1^{(n+1)}$, instead of I_1, SURV updates for mode 2. After obtaining the new mode-2 factor matrix $\mathbf{U}_2^{(n+1)}$ from the SURV updates, this matrix can be used to further project $\mathbf{T}_{::n+1}^{\mathrm{T}}\mathbf{U}_1^{(n+1)}$, now to the mode-2 principal subspace: $\tilde{\mathbf{T}}_{::n+1} = \mathbf{U}_1^{(n+1)^{\mathrm{T}}}\mathbf{T}_{::n+1}\mathbf{U}_2^{(n+1)} = \mathbf{T}_{::n+1} \cdot_1 \mathbf{U}_1^{(n+1)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n+1)^{\mathrm{T}}} \in \mathbb{R}^{R_1^{(n+1)} \times R_2^{(n+1)}}$. The matrix $\tilde{\mathbf{T}}_{::n+1}$ can then be used to make the mode-3 update, i.e., the update in the evolving mode, computationally more efficient, see Section 5.5.3 below. For the downdate of the oldest slice in the window, the same sequential truncation approach can be used as for the update to decrease the computational cost.

In a sequentially truncated approach, it is beneficial to order the updates of the non-evolving modes strategically to minimize the approximation error and computation cost. To determine the ordering of the modes that offers the largest speedup, one can compare the operation counts of both possible orderings.

Updating mode 2 after mode 1 is faster if $I_1^2 I_2 + I_2^2 R_1^{(n+1)} < I_2^2 I_1 + I_1^2 R_2^{(n+1)}$. Defining the dimension reductions S_i for both modes as $S_1 = I_1 - R_1^{(n+1)}$ and $S_2 = I_2 - R_2^{(n+1)}$, the equation can be rewritten as $I_1^2 I_2 + I_2^2 (I_1 - S_1) < I_2^2 I_1 + I_1^2 (I_2 - S_2)$, or $I_2^2 S_1 > I_1^2 S_2$. When the values S_i lie close to each other, one should thus update the mode with the smallest dimension first. As discussed in [217], this ordering is twice beneficial, as it generally leads to a relatively low approximation error as well.

Evolving mode 3

In Section 5.3.2 for mode 3, we track an RRURV decomposition $\mathbf{Q}_3^{(n)} \mathbf{R}_3^{(n)} \mathbf{V}_3^{(n)^{\mathrm{T}}}$ of the mode-3 unfolding of $\mathcal{T}^{(n)}$. The matrix $\mathbf{V}_3^{(n)}$ is an $(I_1 I_2 \times I_1 I_2)$-matrix, so storage of this matrix, and computations with it, are problematic for large tensors. Fortunately, we can work with a $\mathbf{V}$-matrix that has $R_1^{(n+1)} R_2^{(n+1)}$, instead of $I_1 I_2$, columns by using the sequentially truncated new slice $\tilde{\mathbf{T}}_{::n+1} \in \mathbb{R}^{R_1^{(n+1)} \times R_2^{(n+1)}}$ instead of the full new slice. This significantly decreases the storage and computational cost of the mode-3 update.

Due to the sequential truncation of $\mathcal{T}^{(n)}$ in mode 1 and mode 2, we can consider the row space of $\mathbf{T}_{[3;2,1]}^{(n)}$ to be spanned by the rows of $\left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right)^{\mathrm{T}}$ with dimension $R_1^{(n)} R_2^{(n)}$. Consequently, an RRURV decomposition of $\mathbf{T}_{[3;2,1]}^{(n)}$ can be written as follows:

$$\mathbf{T}_{[3;2,1]}^{(n)} \approx \mathbf{Q}_3^{(n)} \mathbf{R}_3^{(n)} \tilde{\mathbf{V}}_3^{(n)^{\mathrm{T}}} \left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right)^{\mathrm{T}},$$

wherein the matrix $\left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right) \tilde{\mathbf{V}}_3^{(n)}$ takes the role of the matrix $\mathbf{V}_3^{(n)}$. Herein $\tilde{\mathbf{V}}_3^{(n)} \in \mathbb{R}^{R_1^{(n)} R_2^{(n)} \times R_1^{(n)} R_2^{(n)}}$ requires much less memory to store than $\mathbf{V}_3^{(n)} \in \mathbb{R}^{I_1 I_2 \times I_1 I_2}$. Another advantage of this approach is that we can obtain the updated core tensor without much computational effort: the mode-3 unfolding of the core tensor is equal to $\mathbf{R}_{3_{1:R_3}, 1:R_3}^{(n)} \tilde{\mathbf{V}}_{3_{:,1:R_3}}^{(n)^{\mathrm{T}}} \in \mathbb{R}^{R_3 \times R_1 R_2}$.

Below, we describe the update and downdate for the evolving mode in detail. We assume that $w \geq R_1^{(n)} R_2^{(n)}$, but sequential truncation can be combined with the strategies from Section 5.5.1 to speed up the updates even more if $w < R_1^{(n)} R_2^{(n)}$.

Update After computing the new mode-1 and mode-2 subspaces, the RRURV decomposition for mode-3 is adapted to take these subspace changes into

account before including the contributions of the new slice $\tilde{\mathbf{T}}_{::n+1}$. We construct a decomposition of the form

$$\mathbf{T}^{(n+1)}_{[3;2,1]} \approx \mathbf{Q}^{(n)}_3 \mathbf{R}^{(n)}_3 \mathbf{P}^{\mathrm{T}} \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}},$$

through the introduction of the matrix $\mathbf{P} \in \mathbb{R}^{R^{(n+1)}_1 R^{(n+1)}_2 \times R^{(n)}_1 R^{(n)}_2}$, which is an updated version of the matrix $\tilde{\mathbf{V}}^{(n)}_3$ that takes the new mode-1 and mode-2 subspaces into account:

$$\mathbf{P} = \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}} \left(\mathbf{U}^{(n)}_2 \otimes \mathbf{U}^{(n)}_1 \right) \tilde{\mathbf{V}}^{(n)}_3. \tag{5.6}$$

It is possible to avoid the explicit computation of the Kronecker product in (5.6) by applying the elementary rotations that transform $\mathbf{U}^{(n)}_1$ and $\mathbf{U}^{(n)}_2$ into $\mathbf{U}^{(n+1)}_1$ and $\mathbf{U}^{(n+1)}_2$, respectively, directly on the tensorization of $\tilde{\mathbf{V}}^{(n)}_3$. Afterwards, we orthogonalize $\mathbf{P}$ through the computation of a QL-decomposition $\mathbf{P} = \mathbf{Q}'\mathbf{L}'$ to obtain:

$$\mathbf{T}^{(n+1)}_{[3;2,1]} \approx \mathbf{Q}^{(n)}_3 \mathbf{R}^{(n)}_3 \mathbf{L}'^{\mathrm{T}} \mathbf{Q}'^{\mathrm{T}} \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}}.$$

Taking $\mathbf{R}^{(n)}_3 \mathbf{L}'^{\mathrm{T}} \in \mathbb{R}^{R^{(n)}_1 R^{(n)}_2 \times R^{(n+1)}_1 R^{(n+1)}_2}$ as the new $\mathbf{R}$-matrix[1] and $\mathbf{Q}' \in \mathbb{R}^{R^{(n+1)}_1 R^{(n+1)}_2 \times R^{(n+1)}_1 R^{(n+1)}_2}$ as the new $\tilde{\mathbf{V}}$-matrix, we now proceed as in Section 5.3.2 to include the contribution of the new slice $\tilde{\mathbf{T}}_{::n+1}$ into the decomposition $\mathbf{Q}_{\mathrm{up}} \mathbf{R}_{\mathrm{up}} \tilde{\mathbf{V}}^{\mathrm{T}}_{\mathrm{up}} \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}}$. As $\tilde{\mathbf{T}}_{::n+1}$ has already been compressed to dimensions $R^{(n+1)}_1 \times R^{(n+1)}_2$, we can ignore the matrix $\left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}}$ for this step and work only with the small matrix $\tilde{\mathbf{V}}_{\mathrm{up}} \in \mathbb{R}^{R^{(n+1)}_1 \times R^{(n+1)}_2 \times R^{(n+1)}_1 \times R^{(n+1)}_2}$ which is very convenient computationally.

Downdate The downdate in Section 5.3.2 does not depend on the $\mathbf{V}_{\mathrm{up}}$-matrix, so we can use exactly the same strategy as for the case without sequential truncation to downdate $\mathbf{Q}_{\mathrm{up}} \mathbf{R}_{\mathrm{up}} \tilde{\mathbf{V}}^{\mathrm{T}}_{\mathrm{up}} \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}}$ into $\mathbf{Q}_{\mathrm{dn}} \mathbf{R}_{\mathrm{dn}} \tilde{\mathbf{V}}^{\mathrm{T}}_{\mathrm{dn}} \left(\mathbf{U}^{(n+1)}_2 \otimes \mathbf{U}^{(n+1)}_1 \right)^{\mathrm{T}}$. If the mode-1 and mode-2 subspaces are mildly non-stationary, the rank-revealing property of $\mathbf{R}_{\mathrm{dn}}$ can be restored by a small number of deflation steps [73] to obtain the new

[1] Note that some columns right of the diagonal are nonzero in the QL-decomposition of the wide matrix $\mathbf{P}$ in case that $R^{(n+1)}_1 R^{(n+1)}_2 < R^{(n)}_1 R^{(n)}_2$. Additionally, the new $\mathbf{R}$-matrix has more rows than columns, so one or more of its least significant rows, and the corresponding columns of $\mathbf{Q}^{(n)}_3$, should be eliminated. Consequently, slightly more work is required in this case before the update can be performed.

Table 5.2: With the strategies from Section 5.5, TMLUTVD updating requires less storage and is faster than the batch algorithm. Storage and computational complexity for the batch TMLSVD method and the proposed updating method. Superscripts are dropped and $R = R_1^{(n)} = R_2^{(n)} = R_3^{(n)}$ is assumed.

Memory		
Batch		$\mathcal{O}(I_1 I_2 w)$
Updating		
	New slice	$\mathcal{O}(I_1 I_2)$
	Slices in window	$\mathcal{O}(w(I_1 + I_2)p)$
	Modes 1 and 2	$\mathcal{O}(I_1^2 + I_2^2)$
	mode 3	$\mathcal{O}(\min(w, R^2)(w + \min(w, R^2) + R^2))$
Computation		
Batch		$\mathcal{O}(I_1 I_2 w \min(I_1, I_2) + I_1 I_2 w R)$
Updating		
	Modes 1 and 2	$\mathcal{O}((I_1 + I_2) I_1 p + I_2^2 R)$
	mode 3	$\mathcal{O}(R^2(\min(w^2, R^4) + I_1 + I_2))$
	Core	$\mathcal{O}(R^4)$

RRURV $\mathbf{Q}_3^{(n+1)} \mathbf{R}_3^{(n+1)} \tilde{\mathbf{V}}_3^{(n+1)^{\mathrm{T}}} \left(\mathbf{U}_2^{(n+1)} \otimes \mathbf{U}_1^{(n+1)} \right)^{\mathrm{T}}$. In highly non-stationary environments, up to $R_3^{(n+1)}$ deflation steps may be required.

5.6 Experiments

In a first experiment, the LMLRA of a tensor is tracked throughout a series of abrupt subspace changes. In a second experiment, we perform tracking in a DOA estimation context. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0 [228].

5.6.1 Updating with abrupt subspace changes

In a first experiment, four low multilinear rank tensors are concatenated in the third mode, with each tensor of dimensions $100 \times 100 \times 400$, simulating a situation where abrupt subspace changes occur during the TMLUTVD tracking application. The multilinear rank of the subtensors is $(3, 4, 5)$, $(3, 6, 5)$, $(4, 5, 3)$

and $(5, 4, 4)$, respectively. The entries of the factor matrices and core tensor are drawn from the uniform distribution on $[0, 1]$ and Gaussian distributed noise with an SNR of 30dB is added to the tensor, where the SNR is defined as $\mathbb{E}\left(20 \log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)\right)$ for a tensor $\mathcal{T}$ and a noise tensor $\mathcal{N}$ of the same dimensions. First, the subspaces are computed for the first 50 slices and then, the decomposition is updated with a window length of 50. In Figure 5.8, the estimated dimensions of the principal subspaces are shown, as are the relative Frobenius errors with respect to the noiseless windowed tensor and the computation times for both the basic TMLUTVD tracking algorithm and the batch TMLSVD method. The former method is applied with the thresholds from Section 5.3.4 and $\alpha = 1.24$, as in [243] and without the strategies from Section 5.5. The latter method computes the MLSVD of the windowed tensor from scratch after every update, without sequential truncation (to be consistent with the TMLUTVD method) and uses the same thresholds as the TMLUTVD method to decide on the dimension of the principal subspace. As can be seen from Figure 5.8, the correct dimensions of the principal subspaces are found throughout the whole updating process and the relative approximation error of the windowed tensor is lower for the TMLUTVD tracking method than for the batch TMLSVD method. The lower error is a result of the better subspace estimate of the mode-1 and mode-2 subspaces by the incremental SURV computation compared to the batch MLSVD of the windowed tensor, which is consistent with observations in [243]. When the sliding window moves into a new low multilinear rank tensor and thus covers two low multilinear rank tensors partially, the estimated mode-n rank values are equal to the sums of the mode-n rank values of the two low multilinear rank tensors in the window, which is as expected. When the last slice of the old low multilinear rank tensor is downdated, the estimated mode-n rank values become equal to those of the most recent low multilinear rank tensor. The TMLUTVD tracking method is slower than the batch TMLSVD method, as the computational complexity depends on $I_1^2 I_2^2$ instead of $I_1 I_2 w$ and SVDs can be computed very efficiently in MATLAB. As discussed before, however, it has the advantage that the full tensor does not need to be processed in every time instance: only the newest and oldest slices are used to perform an update and downdate.

In Figure 5.9, the experiment is repeated using compression and sequential truncation, combining the strategies from Section 5.5.2 and 5.5.3. It can be seen that the computation time of the updating method is significantly decreased and the TMLUTVD tracking method is now faster than the batch TMLSVD method while the accuracy of the method is unaffected.

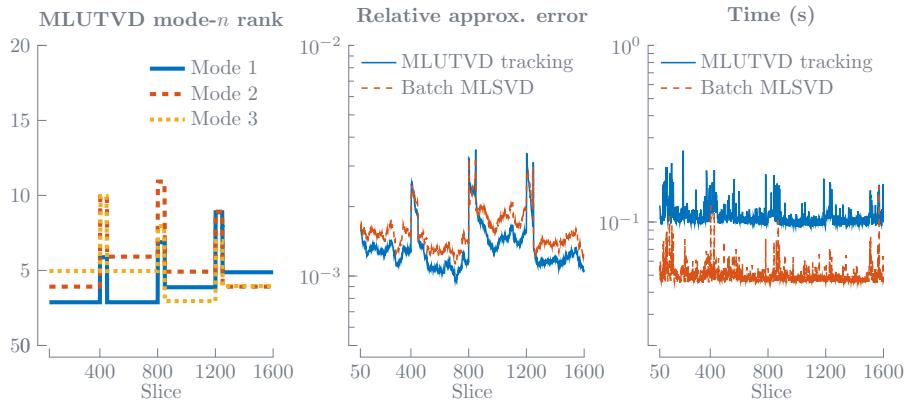

Figure 5.8: The updating method tracks an LMLRA of a tensor with high accuracy, even when abrupt changes in the subspaces occur. Updating of the LMLRA of four low rank $(100 \times 100 \times 400)$-tensors, concatenated in the third mode, with a window length of 50. Gaussian distributed noise is added with an SNR of 30dB. Left: Estimated rank values by the TMLUTVD tracking method for all three modes. Middle: Relative Frobenius error of the noiseless windowed tensor for TMLUTVD tracking and batch TMLSVD. Right: Computation time for the initialization and each update for TMLUTVD tracking and batch TMLSVD.

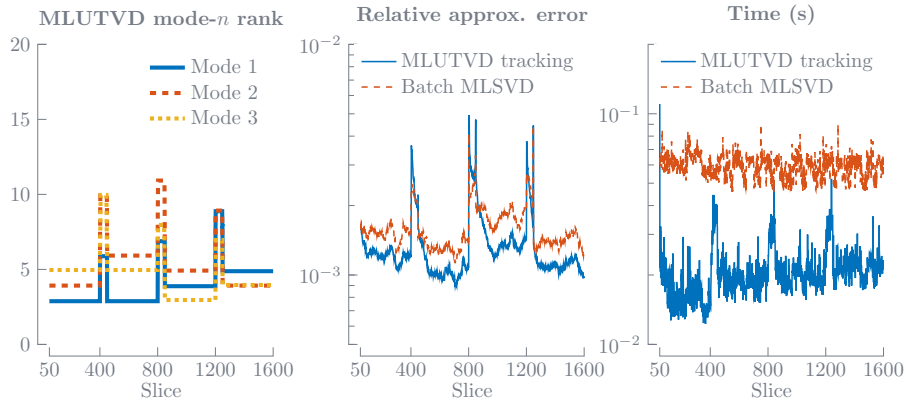

Figure 5.9: By compressing the new slices (Section 5.5.2) and using sequential truncation (Section 5.5.3), the method can be sped up significantly, without negatively impacting the accuracy. Same experiment setting as in Figure 5.8.

5.6.2 Direction-of-arrival estimation

In a second experiment, we estimate the DOAs of a number of sources impinging from the far-field on a uniform rectangular array (URA), using a data model similar to the one in [91]. We consider a set of R undamped exponentials impinging on a URA of dimensions $K \times L$ for N time instances. Because of the structure of the URA, the entries of the observation tensor $\mathcal{Y} \in \mathbb{C}^{K \times L \times N}$ satisfy

$$y_{kln} = \sum_{r=1}^{R} s_r(n) \mathrm{e}^{j(k-1)\mu_{r1}} \mathrm{e}^{j(l-1)\mu_{r2}} + n_{kln},$$

with j the imaginary unit, n_{kln} noise, $s_r(n)$ the complex amplitude of the rth exponential at the nth time instance and μ_r the coordinates of the rth spatial frequency vector. The latter coordinates depend on the DOAs of the R sources. This means that we can model $\mathcal{Y}$ as follows:

$$\mathcal{Y} = \mathcal{A} \cdot_3 \mathbf{S}^{\mathrm{T}} + \mathcal{N},$$

wherein $\mathcal{S} \in \mathbb{C}^{R \times N}$ holds the source signals, $\mathcal{N}$ is a noise tensor and $\mathcal{A} \in \mathbb{C}^{I \times J \times R}$ is the array steering tensor. The frontal slices of $\mathcal{A}$ satisfy:

$$\mathbf{A}_{::r} = \mathbf{a}^{(1)}(\mu_{r1}) \left(\mathbf{a}^{(2)}(\mu_{r2}) \right)^{\mathrm{T}},$$

wherein

$$\mathbf{a}^{(1)}(\mu_{r1}) = \left[1 \ \mathrm{e}^{j\mu_{r1}} \ \mathrm{e}^{2j\mu_{r1}} \ \ldots \ \mathrm{e}^{(K-1)j\mu_{r1}} \right]^{\mathrm{T}} \in \mathbb{C}^{K},$$

and $\mathbf{a}^{(2)}(\mu_{r2}) \in \mathbb{C}^{L}$ is defined analogously. From this model, we can derive that, in the absence of noise, $\mathcal{Y}$ has rank R and that the different mode-n rank values of $\mathcal{Y}$ do not exceed R.

From the data model above, a TMLSVD can be used to recover the signal subspace with a higher accuracy than with matrix-based methods [91]. One computes a TMLSVD $\mathcal{Y} \approx \mathcal{C} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \cdot_3 \mathbf{U}_3$ and compares the column space of the estimated unfolded steering tensor $\hat{\mathbf{A}}_{[2,1;3]}$, with $\hat{\mathcal{A}} = \mathcal{C} \cdot_1 \mathbf{U}_1 \cdot_2 \mathbf{U}_2 \in \mathbb{C}^{K \times L \times R_3^{(n)}}$, with the column space of the true unfolded steering tensor $\mathbf{A}_{[2,1;3]}$. The comparison is performed through the computation of the subspace angle between the latter $R_3^{(n)}$-dimensional subspaces. Naturally, the TMLUTVD can be applied in the same way to obtain an estimate of the steering tensor. Using the proposed TMLUTVD tracking method, we can track the steering tensor in real time and account for changes in the number of sources or their location.

We consider an experiment on a URA of dimensions $K \times L$, with $K = L = 80$. For the first 400 time instances, we consider $R = 4$ sources with azimuth and elevation (in degrees) equal to $\left[45 \ -45 \ 35 \ 75 \right]$ and $\left[45 \ -45 \ 15 \ 85 \right]$, respectively. For

the next 400 time instances, sources three and four no longer appear and four new sources are added in their place. We thus have $R = 6$ sources with azimuth and elevation equal to $\begin{bmatrix} 45 & -45 & 65 & 15 & 35 & -65 \end{bmatrix}$ and $\begin{bmatrix} 45 & -45 & -5 & 25 & 85 & -50 \end{bmatrix}$, respectively. Both the real and imaginary parts of the source signals are sampled from the standard normal distribution and we add additive Gaussian noise with an SNR of 0 dB to both the real and imaginary parts of the observed samples. We then track the signal subspaces using the TMLUTVD tracking method with a window length of 50 time instances and compare the approach to batch TMLSVD, again with the thresholds as in Section 5.3.4 and $\alpha = 1.24$ for both methods. The results can be seen in Figures 5.10 and 5.11 for the basic TMLSVD approach and for the approach with compression and sequential truncation (see Section 5.5), respectively. Aside from the error on the estimation of the column space of the steering matrix (middle plot), we also show the estimated mode-3 rank values, which correspond to the estimated number of sources, and the time per update for both methods.

From both figures, it can be seen that the TMLUTVD tracking method estimates the correct number of sources and achieves at least the same accuracy as the batch TMLSVD method when the sources are stationary. As for the first experiment, the former method is actually slightly more accurate than the latter. When the sources change halfway through the experiment, both methods needs some time to correctly estimate the new steering matrix. This is as expected for a sliding window approach: the original sources only disappear from the window after 50 new updates have been processed. From Figures 5.10 and 5.11, we can clearly see that compression and sequential truncation can speed up the tracking method dramatically with minimal effect on the accuracy of the subspace estimation.

5.7 Conclusion

An updating method was proposed to track an LMLRA of a tensor as new slices are added and removed in a certain mode. The algorithm uses an TMLUTVD to track the subspaces of both the evolving and non-evolving modes with automatic rank detection in a numerically stable way. We showed that the approximation error of the tracked LMLRA can be bounded and derived the complexities with respect to storage and computation. Several strategies were proposed to significantly speed up the algorithm. We demonstrated that the algorithm is faster than the batch method, while maintaining good accuracy of the decomposition, even when abrupt changes in the subspaces are encountered.

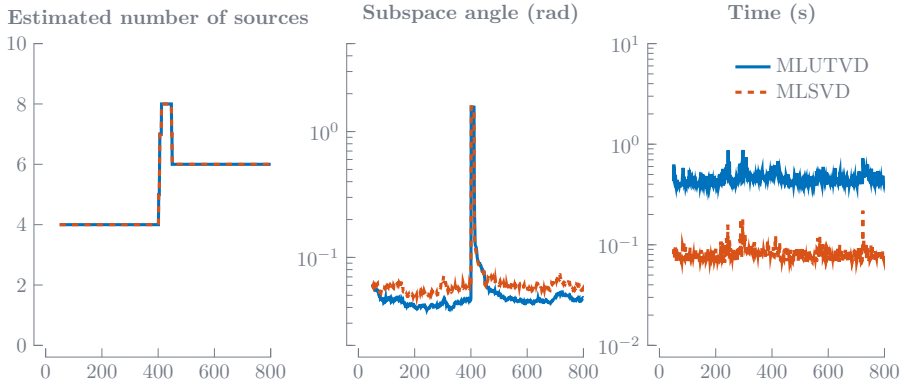

Figure 5.10: The TMLUTVD updating method tracks the DOAs of several sources with high accuracy and correctly estimates the number of sources. For the first and last 400 time instances, four and six sources are tracked, respectively. Gaussian distributed noise is added with an SNR of 0dB and a window length of 50 is used. Left: Estimated mode-3 rank values. Middle: Angle in radians between the column spaces of the true and estimated unfolded steering tensors. Right: Computation time for each update for TMLUTVD updating and each batch TMLSVD computation.

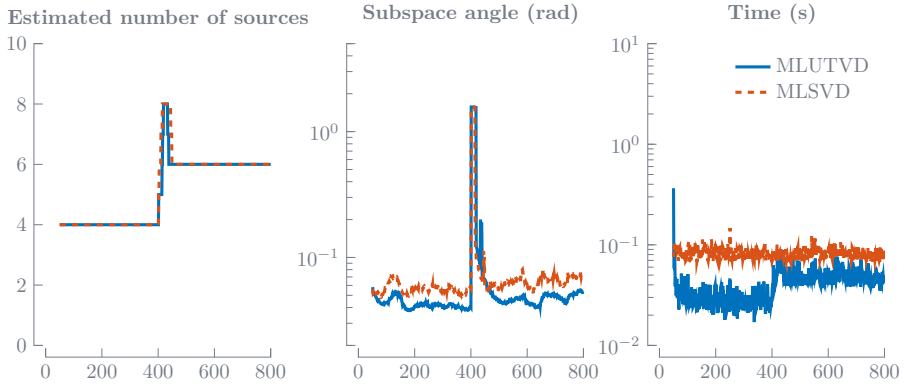

Figure 5.11: With compression (Section 5.5.2) and sequential truncation (Section 5.5.3), the TMLUTVD tracking method becomes faster than the batch TMLSVD algorithm, with minimal loss in accuracy for the estimation of the subspaces. Same experiment setting as in Figure 5.11.

5.A Proof of approximation bound

Theorem 5.A.1: Approximation bound for non-evolving modes Let $\mathbf{Q}_1\mathbf{L}_1\mathbf{J}\mathbf{\Phi}_1^{\mathrm{T}}$ be an SURV decomposition of $\begin{bmatrix}\mathbf{N}_1 & \mathbf{T}_{[1;3,2]}\end{bmatrix}\mathbf{J}$. Split $\mathbf{Q}_1 \in \mathbb{R}^{I_1 \times I_1}$ into $\mathbf{Q}_1 = [\mathbf{Q}_N \;\; \mathbf{Q}_S]$, wherein $\mathbf{Q}_S \in \mathbb{R}^{I_1 \times R_1}$ is a basis for the principal column space and $\mathbf{Q}_N \in \mathbb{R}^{I_1 \times (I_1 - R_1)}$ is a basis for the orthogonal complement. Partition $\mathbf{\Phi}_1$ into $\mathbf{\Phi}_1 = \begin{bmatrix}\mathbf{\Phi}_{11} & \mathbf{\Phi}_{12} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_{22}\end{bmatrix}$ such that $\mathbf{\Phi}_{11}$ and $\mathbf{\Phi}_{22}$ are square and their dimensions match the column dimensions of $\mathbf{N}_1$ and $\mathbf{T}_{[1;3,2]}$, respectively. Finally, partition $\mathbf{L}_1$ into $\mathbf{L}_1 = \begin{bmatrix}\mathbf{L}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{0} & \mathbf{L}_{22} & \mathbf{0}\end{bmatrix}$, with $\mathbf{L}_{11}$ and $\mathbf{L}_{22}$ square, such that it matches the structure of $\mathbf{Q}_1$ and $\mathbf{\Phi}_1$. Then

$$\left\|\mathbf{T}_{[1;3,2]} - \mathbf{Q}_S\mathbf{Q}_S^{\mathrm{T}}\mathbf{T}_{[1;3,2]}\right\|_{\mathrm{F}}^2 = \gamma_1^2(I_1 - R_1) - \|\mathbf{L}_{11}\|_{\mathrm{F}}^2 .$$

Proof. For simplicity, we will give the proof for an infinite length sliding window. First, we write $\begin{bmatrix}\overline{\mathbf{L}}_{11} & \mathbf{0} \\ \overline{\mathbf{L}}_{21} & \overline{\mathbf{L}}_{22}\end{bmatrix} := \begin{bmatrix}\mathbf{L}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{L}_{21} & \mathbf{0} & \mathbf{L}_{22} & \mathbf{0}\end{bmatrix}$ by extending $\mathbf{L}_{11}$, $\mathbf{L}_{21}$ and $\mathbf{L}_{22}$ with zero columns. Then, we replace $\mathbf{T}_{[1;3,2]}$ in the expression for the Frobenius norm of the mode-1 approximation error by its decomposition $-\mathbf{Q}_1\mathbf{L}_1\begin{bmatrix}\mathbf{\Phi}_{21} & -\mathbf{\Phi}_{22}\end{bmatrix}^{\mathrm{T}} = \mathbf{Q}_1\mathbf{L}_1\begin{bmatrix}-\mathbf{\Phi}_{21} & \mathbf{\Phi}_{22}\end{bmatrix}^{\mathrm{T}}$, wherein the minuses come from the multiplications with $\mathbf{J}$:

$$\begin{aligned}
\left\|\mathbf{T}_{[1;3,2]} - \mathbf{Q}_S\mathbf{Q}_S^{\mathrm{T}}\mathbf{T}_{[1;3,2]}\right\|_{\mathrm{F}}^2 &= \left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\mathbf{T}_{[1;3,2]}\right\|_{\mathrm{F}}^2 \\
&= \left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\mathbf{Q}_1\mathbf{L}_1\begin{bmatrix}-\mathbf{\Phi}_{21} & \mathbf{\Phi}_{22}\end{bmatrix}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 \\
&= \left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\left(\left(\mathbf{Q}_N\overline{\mathbf{L}}_{11} + \mathbf{Q}_S\overline{\mathbf{L}}_{21}\right)\left(-\mathbf{\Phi}_{21}^{\mathrm{T}}\right) + \mathbf{Q}_S\overline{\mathbf{L}}_{22}\mathbf{\Phi}_{22}^{\mathrm{T}}\right)\right\|_{\mathrm{F}}^2 \\
&= \left\|-\mathbf{Q}_N\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{21}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 \\
&= \left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{21}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 ,
\end{aligned}$$

wherein we used that $\mathbf{Q}_N^{\mathrm{T}}\mathbf{Q}_S = \mathbf{0}$ and $\mathbf{Q}_N^{\mathrm{T}}\mathbf{Q}_N = \mathbf{I}$ due to the orthogonality of $\mathbf{Q}_1$. Analogously, we find

$$\left\|\mathbf{N}_1 - \mathbf{Q}_S\mathbf{Q}_S^{\mathrm{T}}\mathbf{N}_1\right\|_{\mathrm{F}}^2 = \left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\mathbf{N}_1\right\|_{\mathrm{F}}^2 = \left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 . \tag{5.7}$$

As $\|\mathbf{A}\|_{\mathrm{F}}^2 = \mathrm{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right) = \mathrm{tr}\left(\mathbf{A}\mathbf{A}^{\mathrm{T}}\right)$ for any matrix $\mathbf{A}$, we can write

$$\begin{aligned}
\left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{21}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 &= \mathrm{tr}\left(\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{21}^{\mathrm{T}}\mathbf{\Phi}_{21}\overline{\mathbf{L}}_{11}^{\mathrm{T}}\right) \\
&= \mathrm{tr}\left(\overline{\mathbf{L}}_{11}\left(\mathbf{\Phi}_{11}^{\mathrm{T}}\mathbf{\Phi}_{11} - \mathbf{I}\right)\overline{\mathbf{L}}_{11}^{\mathrm{T}}\right),
\end{aligned}$$

where $\mathbf{\Phi}_{21}^{\mathrm{T}}\mathbf{\Phi}_{21} = \mathbf{\Phi}_{11}^{\mathrm{T}}\mathbf{\Phi}_{11} - \mathbf{I}$ due to the J-orthogonality of $\mathbf{\Phi}$. The trace operator is linear, so we find

$$\begin{aligned}\left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{21}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 &= \operatorname{tr}\left(\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\mathbf{\Phi}_{11}\overline{\mathbf{L}}_{11}^{\mathrm{T}}\right) - \operatorname{tr}\left(\overline{\mathbf{L}}_{11}\overline{\mathbf{L}}_{11}^{\mathrm{T}}\right)\\ &= \left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 - \left\|\overline{\mathbf{L}}_{11}\right\|_{\mathrm{F}}^2\\ &= \left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2,\end{aligned}$$

We can now replace $\left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\right\|_{\mathrm{F}}^2$ by $\left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\mathbf{N}_1\right\|_{\mathrm{F}}^2$ following (5.7) and use the fact that $\mathbf{N}_1\mathbf{N}_1^{\mathrm{T}} = \gamma_1^2\mathbf{I}$:

$$\begin{aligned}\left\|\mathbf{T}_{[1;3,2]} - \mathbf{Q}_S\mathbf{Q}_S^{\mathrm{T}}\mathbf{T}_{[1;3,2]}\right\|_{\mathrm{F}}^2 &= \left\|\overline{\mathbf{L}}_{11}\mathbf{\Phi}_{11}^{\mathrm{T}}\right\|_{\mathrm{F}}^2 - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2\\ &= \left\|\mathbf{Q}_N\mathbf{Q}_N^{\mathrm{T}}\mathbf{N}_1\right\|_{\mathrm{F}}^2 - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2\\ &= \operatorname{tr}\left(\mathbf{Q}_N^{\mathrm{T}}\mathbf{N}_1\mathbf{N}_1^{\mathrm{T}}\mathbf{Q}_N\right) - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2\\ &= \gamma_1^2\left\|\mathbf{Q}_N\right\|_{\mathrm{F}}^2 - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2\\ &= \gamma_1^2\left(I_1 - R_1\right) - \left\|\mathbf{L}_{11}\right\|_{\mathrm{F}}^2\end{aligned}$$

□

5.B Update of the core tensor

The mode-3 unfolding of the core tensor of the TMLUTVD can be obtained from the RRURV for the evolving mode 3 by computing $\mathbf{C}_{[3;2,1]}^{(n)} = \mathbf{R}_{3_{1:R_3},1:R_3}^{(n)}\mathbf{V}_{3_{:,1:R_3}}^{(n)^{\mathrm{T}}}\left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right) \in \mathbb{R}^{R_3 \times R_1 R_2}$. When sequential truncation is used, the core can even be found with very limited computational effort, see Section 5.5.3. Below, we describe another way to obtain the updated core tensor. For large values of I_1 and I_2, it is more computationally efficient than the straightforward computation above when no sequential truncation is applied.

Let $\mathcal{C}^{(n)}$ and $\mathcal{C}^{(n+1)}$ be the old and new core tensor, respectively. In batch methods, $\mathcal{C}^{(n+1)}$ is computed as

$$\mathcal{C}^{(n+1)} = \mathcal{T}^{(n+1)} \cdot_1 \mathbf{U}_1^{(n+1)^{\mathrm{T}}} \cdot_2 \mathbf{U}_2^{(n+1)^{\mathrm{T}}} \cdot_3 \mathbf{U}_3^{(n+1)^{\mathrm{T}}}. \tag{5.8}$$

However, in an updating context, using the full tensor $\mathcal{T}^{(n+1)}$ is generally computationally expensive. We thus use only the compact TMLUTVD

representation $\mathcal{C}^{(n)} \cdot_1 \mathbf{U}_1^{(n)} \cdot_2 \mathbf{U}_2^{(n)} \cdot_3 \overline{\mathbf{U}}_3^{(n)}$ of $\mathcal{T}^{(n)}$, wherein $\overline{\mathbf{U}}_3^{(n)}$ is equal to the matrix $\mathbf{U}_3^{(n)}$ without its first row, as this row corresponds to the downdated slice. As shown below, the update of the core of the TMLUTVD can be split into two terms, one depending on $\mathcal{T}^{(n)}$, for which we use only the compact TMLUTVD representation, and one depending on the new slice $\mathbf{T}_{::n+1}$. Unfolding (5.8) in the third mode gives:

$$\begin{aligned}\mathbf{C}_{[3;2,1]}^{(n+1)} &= {\mathbf{U}_3^{(n+1)}}^{\mathrm{T}} \mathbf{T}_{[3;2,1]}^{(n+1)} \left(\mathbf{U}_2^{(n+1)} \otimes \mathbf{U}_1^{(n+1)}\right) \\ &\approx {\mathbf{U}_3^{(n+1)}}^{\mathrm{T}} \begin{bmatrix} \overline{\mathbf{U}}_3^{(n)} \mathbf{C}_{[3;2,1]}^{(n)} \left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right)^{\mathrm{T}} \\ \mathbf{t}^{\mathrm{T}} \end{bmatrix} \left(\mathbf{U}_2^{(n+1)} \otimes \mathbf{U}_1^{(n+1)}\right).\end{aligned}$$

Let $\mathbf{u}_3^{(n+1)}$ be the last row of $\mathbf{U}_3^{(n+1)}$ and let $\underline{\mathbf{U}}_3^{(n+1)}$ hold all other rows. Then $\mathbf{C}_{[3;2,1]}^{(n+1)}$ can be written as follows:

$$\begin{aligned}\mathbf{C}_{[3;2,1]}^{(n+1)} &\approx \left[{\underline{\mathbf{U}}_3^{(n+1)}}^{\mathrm{T}} \overline{\mathbf{U}}_3^{(n)} \mathbf{C}_{[3;2,1]}^{(n)} \left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right)^{\mathrm{T}} + {\mathbf{u}_3^{(n+1)}}^{\mathrm{T}} \mathbf{t}^{\mathrm{T}} \right] \left(\mathbf{U}_2^{(n+1)} \otimes \mathbf{U}_1^{(n+1)}\right) \\ &= \mathbf{W}_3 \mathbf{C}_{[3;2,1]}^{(n)} \left(\mathbf{U}_2^{(n)} \otimes \mathbf{U}_1^{(n)}\right)^{\mathrm{T}} \left(\mathbf{U}_2^{(n+1)} \otimes \mathbf{U}_1^{(n+1)}\right) + {\mathbf{u}_3^{(n+1)}}^{\mathrm{T}} \tilde{\mathbf{t}}^{\mathrm{T}} \\ &= \mathbf{W}_3 \mathbf{C}_{[3;2,1]}^{(n)} (\mathbf{W}_2 \otimes \mathbf{W}_1)^{\mathrm{T}} + {\mathbf{u}_3^{(n+1)}}^{\mathrm{T}} \tilde{\mathbf{t}}^{\mathrm{T}}, \end{aligned} \tag{5.9}$$

where $\mathbf{W}_i = {\mathbf{U}_i^{(n+1)}}^{\mathrm{T}} \mathbf{U}_i^{(n)}$ for $i = 1, 2$ and $\mathbf{W}_3 = {\underline{\mathbf{U}}_3^{(n+1)}}^{\mathrm{T}} \overline{\mathbf{U}}_3^{(n)}$. In the first term, the $\mathbf{W}_i^{(n+1,n)}$-matrices are generally small, i.e., they are of dimensions $(R_i^{(n)} \times R_i^{(n+1)})$. Their explicit construction is avoided by applying the elementary rotations that transform $\mathbf{U}_1^{(n)}$ and $\mathbf{U}_2^{(n)}$ into $\mathbf{U}_1^{(n+1)}$ and $\mathbf{U}_2^{(n+1)}$, respectively, directly on the old core tensor $\mathcal{C}^{(n)}$.

We can analyze the cost of the computation of the core tensor via (5.9). Assume that the mode-i rank values stay the same during the iteration. The first term has a complexity of $\mathcal{O}(R^2 w)$ flop for the computation of $\mathbf{W}_3^{(n+1,n)} = {\overline{\mathbf{U}}_3^{(n+1)}}^{\mathrm{T}} \underline{\mathbf{U}}_3^{(n)}$. Explicitly forming the matrices ${\mathbf{U}_i^{(n)}}^{\mathrm{T}} \mathbf{U}_i^{(n+1)}$, for $i = 1, 2$, can be avoided by applying the transformations to change $\mathbf{U}_i^{(n}$ into $\mathbf{U}_i^{(n+1)}$ to the core tensor as well. Further, the first term requires $\mathcal{O}(R^4)$ flop for the MTKRONPROD. The second term has a complexity of $\mathcal{O}(R^3)$ flop. This gives a total complexity of $\mathcal{O}(R^2 w + R^4)$ flop. Computing the new core tensor naively, without exploiting the LMLRA structure, has a complexity of $\mathcal{O}(I_1 I_2 w R)$ flop, so a significant efficiency gain is achieved with the proposed updating strategy. For tensors of orders larger than three, the gain increases even further.

Part III: Variety

A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost

6

ABSTRACT

The canonical polyadic decomposition (CPD) can be used to extract meaningful components from a tensor. Most existing optimization methods for fitting the CPD use as cost function the least-squares distance between the tensor and its CPD. While the minimum of this cost function coincides with the maximum likelihood estimator for data with additive i.i.d. Gaussian distributed noise, for other noise distributions, better-suited cost functions exist. For such cost functions, first-order, gradient-based optimization methods have been proposed. However, (approximate) second-order methods, which additionally use information from the Hessian of the cost function to achieve faster convergence, are still largely unexplored. In this chapter, we generalize the Gauss–Newton nonlinear least-squares algorithm to twice differentiable entry-wise cost functions. The low-rank structure of the problem is exploited to keep the computational cost low. As a special case, β-divergence cost functions are examined. We show that quadratic convergence can be obtained close to the global optimum with a reasonable extra cost in memory and computation time, making the proposed method particularly useful when high accuracy of the decomposition is desired.

This chapter is based on M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost," IEEE Transactions on Signal Processing, vol. 68, pp. 4454–4465, 8 2020. The specific contributions of the doctoral candidate are listed in Section 11.1.

6.1 Introduction

Tensors—higher-order generalizations of matrices—are used extensively in machine learning and signal processing [35, 174]. In contrast to matrices, tensors preserve higher-order structure that is present in the data. Building on the well-established field of matrix decompositions, different types of tensor decompositions have been proposed to reveal the underlying information in the tensor in a compact way. Several algebraic and optimization-based algorithms have been developed for tensor decompositions such as the canonical polyadic decomposition (CPD) [2, 62, 157, 183]. The CPD is a popular tensor decomposition due to its simplicity and mild uniqueness conditions [58, 123] and therefore, its computation has been intensively studied. Still, optimization-based algorithms for computing the CPD of a tensor have mainly considered least-squares (LS) cost functions. They minimize the Frobenius norm of the difference between the tensor and its CPD model. This is indeed optimal if the noise on the tensor data is additive, independent and identically distributed (i.i.d.) and follows a Gaussian distribution, as the LS solution is the maximum likelihood estimator (MLE) under these assumptions for the noise [146]. However, using a different criterion could be better if the noise is differently distributed [38, 80, 105]. While methods exist that allow the use of β-divergence [38, 80] or general [105] cost functions, these are all first-order methods that use only gradient information to determine the next optimization step. Second-order methods additionally use curvature information from the Hessian of the cost function. Consequently, every iteration is typically more costly than for first-order methods, but the use of second-order information can lead to fewer iterations and better convergence in general. In practice, the robustness with respect to initialization is often increased as well. If used naively, however, these methods quickly become prohibitively expensive in computation time and memory. We therefore propose a method for computing the CPD for arbitrary twice differentiable entry-wise cost functions that uses Hessian information and exploits the multilinear structure of the problem. The method obtains fast local convergence, while at the same time keeping storage and time requirements reasonably low. In Chapter 7, an inexact version of the method is outlined, along with a stochastic variant.

We fix notation and give definitions in the remaining of this section. In Sec. 6.2, we propose a second-order optimization algorithm to compute the CPD of a tensor for general cost functions. By exploiting the low-rank structure of the CPD, the curvature information in the Hessian can be used efficiently. We then focus on β-divergences as an example of an alternative cost function in Sec. 6.3. Lastly, numerical experiments are shown in Sec. 6.4. We consider third-order tensors, but with minor modifications, the results also apply to higher orders.

6.1.1 Notation

We denote scalars, vectors and matrices by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively. Tensors are written in calligraphic script ($\mathcal{A}$). Powers, divisions and logarithms of matrices are entry-wise throughout the text. For matrices $\mathbf{X}$ and $\mathbf{Y}$ the former is written as $\mathbf{X}^{\bullet a}$, for the other two their scalar notation is used: $\frac{\mathbf{X}}{\mathbf{Y}}$ and $\log(\mathbf{X})$, respectively. The transpose of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\mathrm{T}}$ and $\mathrm{Diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its diagonal. The $M \times M$-identity matrix is written as $\mathbf{I}_M$. When using subscripts to specify certain entries of a tensor $\mathcal{T}$, a colon refers to all entries of a specific mode, e.g. $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. The vectorization of the tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ maps t_{ijk} to $\mathbf{t} = \mathrm{vec}(\mathcal{T})_q$, with $q = i + (j-1)I + (k-1)IJ$. Its inverse (with context-implied dimensions) is denoted by $\mathrm{unvec}(\mathbf{t})$. The vectorization $\mathrm{vec}(\mathbf{X}) \in \mathbb{R}^{IJ \times 1}$ of a matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$ can be seen as a special case. A mode-n vector of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfolding $\mathbf{T}_{(n)}$ of a tensor $\mathcal{T}$ is constructed by stacking all its mode-n vectors as columns of a matrix. For example, $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{i_2, q}$, with $q = i_1 + (i_3 - 1)I$. We also introduce the permutation matrices $\mathbf{\Pi}^{(n)} \in \{0,1\}^{IJK \times IJK}$ for vectorizations of $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$. The matrix $\mathbf{\Pi}^{(n)}$ permutes the modes of $\mathcal{T}$ directly in its vectorization by bringing the nth mode to the first position. Thus $\mathbf{\Pi}^{(1)} = \mathbf{I}_{IJK}$, and $\mathbf{\Pi}^{(2)}$ and $\mathbf{\Pi}^{(3)}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(3,1,2)$, respectively. The inverse permutation is denoted by $\mathbf{\Pi}^{(n)^{\mathrm{T}}}$, thus bringing the first mode to the nth position. Again, $\mathbf{\Pi}^{(1)^{\mathrm{T}}} = \mathbf{I}_{IJK}$, while $\mathbf{\Pi}^{(2)^{\mathrm{T}}}$ and $\mathbf{\Pi}^{(3)^{\mathrm{T}}}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(2,3,1)$, respectively.

The Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK \times JL}.$$

For $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$, the column-wise Khatri–Rao product is defined as $\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_{:1} \otimes \mathbf{b}_{:1} & \cdots & \mathbf{a}_{:K} \otimes \mathbf{b}_{:K} \end{bmatrix}$, while the row-wise Khatri–Rao product of $\mathbf{A} \in \mathbb{R}^{K \times I}$ and $\mathbf{B} \in \mathbb{R}^{K \times J}$ is $\mathbf{A} \odot^{\mathrm{T}} \mathbf{B} = \left(\mathbf{A}^{\mathrm{T}} \odot \mathbf{B}^{\mathrm{T}}\right)^{\mathrm{T}}$. The Hadamard, or entry-wise, product of $\mathbf{A}$ and $\mathbf{B} \in \mathbb{R}^{I \times J}$ is written as $\mathbf{A} * \mathbf{B}$. Finally, the outer product of vectors is denoted by $\otimes$. A third-order tensor is a rank-1 tensor if it is the outer product of three nonzero vectors. The minimal number of terms needed to write the tensor as a sum of rank-1 tensors, is the rank R of the tensor. Such a decomposition is called a CPD:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r =: [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

The column vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ can be collected as columns of factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively.

6.2 General cost functions for the CPD

In this section, we show how one can use curvature information from the Hessian of the cost function without requiring an inordinate amount of computational resources. We first introduce a general cost function for fitting the CPD and derive efficient expressions for its gradient. Next, we explain that the Hessian approximation of the generalized Gauss–Newton method can be expected to perform well for the CPD and exploit the multilinear structure of the problem to compute this Hessian approximation cheaply.

Computing the CPD of a tensor is a difficult problem in general. The cost function is nonconvex and may have spurious local minima. For the LS cost function, optimization-based algorithms find a rank-R CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ by fitting matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$ that solve $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f$, with

$$f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{1}{2} \left(m_{ijk} - t_{ijk}\right)^2. \tag{6.1}$$

As the triple sum is simply a sum over all tensor entries, it can be replaced by a single sum $\sum_{n=1}^{N}$, where $N = IJK$. We also write $\mathbf{z} = [\mathrm{vec}(\mathbf{A}); \mathrm{vec}(\mathbf{B}); \mathrm{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$, for the vector holding all optimization variables. While the LS distance is by far the most popular cost function when fitting a CPD, its solution does not necessarily correspond to the MLE when the noise on the tensor entries is not additive i.i.d. Gaussian noise [105]. For more general settings, we may consider the following extension of (6.1):

$$\min_{\mathbf{z}} f = \min_{\mathbf{z}} \sum_{n=1}^{N} f_n(m_n(\mathbf{z})).$$

Note that, for the LS cost function in (6.1), $f_n = \frac{1}{2}\left(m_n(\mathbf{z}) - t_n\right)^2$. Here $m_n(\mathbf{z})$ is the (unconstrained) CPD model mapping the variables in $\mathbf{z}$ to the tensor elements in $\mathcal{M}$. In our discussion, the f_n may be different for different tensor entries. They can be any kind of twice differentiable fit function for the corresponding model entry m_n, although the problem may become significantly harder if the function is nonconvex. One class of alternative cost functions for nonnegative t_n and m_n is given by β-divergence cost functions, with as special cases the Kullback–Leibler (KL) divergence $t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n$ and the

Itakura–Saito (IS) divergence $\frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1$, where the first one is convex and the second one is not. A second alternative cost function is the maximum likelihood estimator for data that follows a Rician distribution [126]. The entry-wise modulus of a complex-valued tensor follows such a Rician distribution when the real and imaginary parts are independently perturbed by additive Gaussian noise and they have equal noise variance. As a third alternative cost function, correntropy [133] can be considered. This function behaves as the LS distance if points are close, as the 1-norm distance if points are moderately far apart and as the 0-norm distance if they are far apart, which makes it robust against outliers. Naturally, many other entry-wise differentiable cost functions exist.

6.2.1 Gradient

The gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g_A}; \mathbf{g_B}; \mathbf{g_C}]$. The subgradients $\mathbf{g_A}$, $\mathbf{g_B}$ and $\mathbf{g_C}$ can be found by applying the chain rule. We first derive $\mathbf{g_A}$:

$$\mathbf{g_A} = \sum_{n=1}^{N} \frac{\mathrm{d}f_n}{\mathrm{d}m_n} \frac{\partial m_n}{\partial \mathrm{vec}(\mathbf{A})}. \tag{6.2}$$

Note that all expressions of the form $\frac{\partial m_n}{\partial \mathrm{vec}(\mathbf{A})}$ are transposed rows of the Jacobian matrix $\mathbf{J_A} \in \mathbb{R}^{IJK \times IR}$ for the CPD:

$$\mathbf{J_A} = \frac{\partial \mathrm{vec}\left(\mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}\right)}{\partial \mathrm{vec}(\mathbf{A})^{\mathrm{T}}} = \begin{bmatrix} \frac{\partial m_1}{\partial \mathrm{vec}(\mathbf{A})^{\mathrm{T}}} \\ \vdots \\ \frac{\partial m_N}{\partial \mathrm{vec}(\mathbf{A})^{\mathrm{T}}} \end{bmatrix} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I. \tag{6.3}$$

By substituting $\mathbf{J_A} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I$ into (6.2) and exploiting the multilinear structure, $\mathbf{g_A}$ can be computed as

$$\mathbf{g_A} = \mathrm{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right),$$

where the tensor $\mathcal{R}$ is of the same dimensions as $\mathcal{T}$ and $\mathcal{M}$ and holds the N derivatives $\frac{\partial f_n}{\partial m_n}$. For the LS cost function, $\mathcal{R}$ is simply the residual tensor $\mathcal{M} - \mathcal{T}$, which is particularly convenient computationally. For the aforementioned KL and IS divergences, the tensor $\mathcal{R}$ is equal to $\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}}$ and $\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet 2}}$, respectively. Using permutation matrices, the Jacobians with respect to $\mathbf{B}$ and $\mathbf{C}$ can be written as

$$\mathbf{J_B} = \mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J\right),$$

and

$$\mathbf{J}_{\mathbf{C}} = \mathbf{\Pi}^{(3)^{\mathrm{T}}} \left((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K \right). \tag{6.4}$$

Using these expressions, the vectors $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be computed similarly to $\mathbf{g}_{\mathbf{A}}$:

$$\mathbf{g}_{\mathbf{B}} = \operatorname{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A}) \right),$$

and

$$\mathbf{g}_{\mathbf{C}} = \operatorname{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A}) \right).$$

The matricized tensor times Khatri–Rao products (`MTKRPROD`) in $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be computed efficiently using specialized routines [218, 223, 226].

6.2.2 Hessian approximation

While gradient descent methods use the direction of the negative gradient to obtain a new intermediate solution $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha \mathbf{p}$, second-order optimization methods find the next iterate by solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ for the step direction $\mathbf{p}$. They thus require, besides the gradient, also (a good approximation of) the Hessian $\mathbf{H} = \nabla^2 f \in \mathbb{R}^{(I+J+K)R \times (I+J+K)R}$. The Hessian can be computed as the Jacobian of the gradient $\mathbf{g}$, so we take the derivative of $\mathbf{g}$ with respect to the variables $\mathbf{z}$ again. The chain rule for derivatives gives:

$$\begin{aligned}
\mathbf{H} &= \sum_{n=1}^{N} \frac{\partial}{\partial \mathbf{z}} \left(\frac{\mathrm{d}f_n}{\mathrm{d}m_n} \frac{\partial m_n}{\partial \mathbf{z}} \right) \\
&= \sum_{n=1}^{N} \frac{\partial}{\partial \mathbf{z}} \left(\frac{\mathrm{d}f_n}{\mathrm{d}m_n} \right) \frac{\partial m_n}{\partial \mathbf{z}} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n} \frac{\partial}{\partial \mathbf{z}} \left(\frac{\partial m_n}{\partial \mathbf{z}} \right) \\
&= \sum_{n=1}^{N} \left(\frac{\partial m_n}{\partial \mathbf{z}} \right) \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \left(\frac{\partial m_n}{\partial \mathbf{z}} \right)^{\mathrm{T}} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n} \left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2} \right) \\
&= \sum_{n=1}^{N} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \mathbf{J}_{n:} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n} \left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2} \right),
\end{aligned}$$

where $\mathbf{J} = [\mathbf{J}_{\mathbf{A}}, \mathbf{J}_{\mathbf{B}}, \mathbf{J}_{\mathbf{C}}]$. Note that the second term of the Hessian consists of two factors. The factor $\frac{\mathrm{d}f_n}{\mathrm{d}m_n}$, i.e., the change in cost function, is expected to become small as we move closer to the (global) optimum. The factor $\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2} \right)$ is sparse for the CPD model, as every tensor entry depends only on a limited number of model entries [183]. One can therefore argue that the generalized Gauss–Newton (GGN) method, which approximates the Hessian by only its

first term $\sum_{n=1}^{N} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d} m_n^2} \mathbf{J}_{n:}$, will give a good approximation [169,221,223]. The values $\frac{\mathrm{d}^2 f_n}{\mathrm{d} m_n^2}$ can be collected in a tensor $\mathcal{D}$ of the same dimensions as $\mathcal{T}, \mathcal{M}$ and $\mathcal{R}$. The Hessian approximation can then be written as $\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, where $\mathbf{Z} = \mathrm{Diag}\left(\mathrm{vec}(\mathcal{D})\right) \in \mathbb{R}^{IJK \times IJK}$. Note that, for the LS cost function, $\mathcal{D}$ has all entries equal to 1, which means that $\mathbf{Z} = \mathbf{I}_{IJK}$. For this cost function, the Hessian approximation is thus simply the Gramian of the Jacobian, and we recover the standard Gauss–Newton method for the CPD, as described in [183]. Although we would like the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ to be positive semidefinite (PSD), i.e., having only nonnegative eigenvalues, this does not hold in general. For nonconvex cost functions f_n, negative values can occur in $\mathbf{Z}$ when moving away from the optimum, possibly introducing negative eigenvalues in $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ as well. Special care should be taken when this occurs to ensure that a descent direction is chosen and continued convergence of the optimization algorithm is guaranteed. This can be done either by modifying f, by using an algorithm that can handle indefinite Hessian approximations or by modifying $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ to make it PSD; see Sec. 6.3.1 for an extended discussion. The step obtained from solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ can be combined with the (first-order) Cauchy point within a (dogleg) trust-region approach [150] to guarantee global convergence of the method. In this case, the step is restricted to a small region around the previous iterate, wherein the local second-order approximation of the cost function can be trusted.

6.2.3 Computing $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$

Naively computing the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ of a rank-R $(I \times I \times I)$-tensor requires the multiplication of the matrix $\mathbf{J}^{\mathrm{T}}$, the diagonal matrix $\mathbf{Z}$ and the matrix $\mathbf{J}$ and requires $\mathcal{O}(I^5R^2)$ flop, which already becomes costly for moderate values of I and R. However, by exploiting the sparsity and multilinear structure in $\mathbf{J}$, this computation can be made more efficient; see Sec. 6.2.1. The full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ consists of nine blocks in a 3-by-3 grid. (For a tensor of order M, the Hessian approximation has M^2 blocks in a M-by-M grid.) The diagonal and off-diagonal blocks are of the form $\mathbf{J}_m^{\mathrm{T}}\mathbf{Z}\mathbf{J}_m$ and $\mathbf{J}_l^{\mathrm{T}}\mathbf{Z}\mathbf{J}_m$ with $m \neq l$, respectively, where $\mathbf{J}_1 = \mathbf{J}_{\mathbf{A}}$, $\mathbf{J}_2 = \mathbf{J}_{\mathbf{B}}$ and $\mathbf{J}_3 = \mathbf{J}_{\mathbf{C}}$. Due to symmetry, only the blocks with $m \geq l$ need to be computed. Efficient approaches to obtain the diagonal and off-diagonal blocks are given below.

Diagonal blocks

We show how the diagonal blocks of the GGN Hessian approximation can be computed while exploiting their specific structure and sparsity. For $m = 1$, the

diagonal block $\mathbf{G}^{(1,1)}$ of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J} \in \mathbb{R}^{IR \times IR}$ is given by:

$$\mathbf{G}^{(1,1)} = \mathbf{J}_{\mathbf{A}}^{\mathrm{T}}\mathbf{Z}\mathbf{J}_{\mathbf{A}} = \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right)^{\mathrm{T}} \mathbf{Z} \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right).$$

The Kronecker product with the identity matrix $\mathbf{I}_I$ introduces many zeros into the Jacobian which in turn lead to zeros in $\mathbf{G}^{(1,1)}$. We can exploit this sparsity by only computing the nonzero values of $\mathbf{G}^{(1,1)}$. Denote by $\mathbf{G}_{\mathrm{nz}}^{(1,1)}$ the matrix of dimensions $(I \times R^2)$ holding these nonzero values. One can obtain $\mathbf{G}_{\mathrm{nz}}^{(1,1)}$ from the `MTKRPROD`

$$\mathbf{G}_{\mathrm{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B})\right),$$

where the matrices $\mathbf{C} \odot^{\mathrm{T}} \mathbf{C} \in \mathbb{R}^{K \times R^2}$ and $\mathbf{B} \odot^{\mathrm{T}} \mathbf{B} \in \mathbb{R}^{J \times R^2}$ hold all pair-wise Hadamard products of columns of $\mathbf{C}$ and $\mathbf{B}$, respectively. The proof can be found in Appendix 6.A. As $\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}$ and $\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}$ only have $R(R+1)/2$ unique columns and $\mathbf{G}_{\mathrm{nz}}^{(1,1)}$ has the same repetition pattern, the latter also has only $R(R+1)/2$ unique columns, which can be exploited. The matrix $\mathbf{G}^{(1,1)} \in \mathbb{R}^{IR \times IR}$ can then be obtained from $\mathbf{G}_{\mathrm{nz}}^{(1,1)} \in \mathbb{R}^{I \times R^2}$ by reshaping and reintroducing the zeros. Similarly $\mathbf{G}_{\mathrm{nz}}^{(2,2)}$ and $\mathbf{G}_{\mathrm{nz}}^{(3,3)}$ are computed as

$$\mathbf{G}_{\mathrm{nz}}^{(2,2)} = \mathbf{D}_{(2)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A})\right),$$

and

$$\mathbf{G}_{\mathrm{nz}}^{(3,3)} = \mathbf{D}_{(3)} \left((\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A})\right).$$

Intermediate results of the form $\mathbf{D}_{(3)}^{\mathrm{T}} \left(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}\right)$ can be reused instead of recomputing all `MTKRPROD`s from scratch using dimension trees [158], as $\mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B})\right)$ and $\mathbf{D}_{(2)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A})\right)$ can be derived from it. These matrices are also needed for the off-diagonal blocks.

Off-diagonal blocks

The off-diagonal blocks of the Hessian approximation can be computed efficiently as well using a similar yet slightly more involved approach as for the diagonal blocks. The $(1,2)$ off-diagonal block of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ has dimensions $(IR \times JR)$ and is of the form

$$\mathbf{G}^{(1,2)} = \mathbf{J}_{\mathbf{A}}^{\mathrm{T}}\mathbf{Z}\mathbf{J}_{\mathbf{B}} = \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right)^{\mathrm{T}} \mathbf{Z}\boldsymbol{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J\right),$$

where $\mathbf{J}_{\mathbf{A}}$ and $\mathbf{J}_{\mathbf{B}}$ are substituted using (6.3) and (6.4). In Appendix 6.A, it is shown that $\mathbf{G}^{(1,2)}$ can be computed as

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{b}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{b}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,2)},$$

where the matrix $\mathbf{W}^{(1,2)}$ is obtained by rearranging the entries of the matrix $\mathbf{D}^{\mathrm{T}}_{(3)}(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \in \mathbb{R}^{IJ \times R^2}$ into an $(IR \times JR)$-matrix; see Appendix 6.A.

For the $(1,3)$- and $(2,3)$-blocks of the Gramian, $\mathbf{G}^{(1,3)}$ and $\mathbf{G}^{(2,3)}$ are analogously computed as:

$$\mathbf{G}^{(1,3)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{c}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{c}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{c}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{c}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,3)},$$

and

$$\mathbf{G}^{(2,3)} = \begin{bmatrix} \mathbf{b}_{:1}\mathbf{c}_{:1}^{\mathrm{T}} & \cdots & \mathbf{b}_{:R}\mathbf{c}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{b}_{:1}\mathbf{c}_{:R}^{\mathrm{T}} & \cdots & \mathbf{b}_{:R}\mathbf{c}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(2,3)}.$$

Analogously to the $(1,2)$-block, the matrices $\mathbf{W}^{(1,3)}$ and $\mathbf{W}^{(2,3)}$ are now obtained by rearranging the entries of the matrices $\mathbf{D}^{\mathrm{T}}_{(2)}(\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \in \mathbb{R}^{IK \times R^2}$ and $\mathbf{D}^{\mathrm{T}}_{(1)}(\mathbf{A} \odot^{\mathrm{T}} \mathbf{A}) \in \mathbb{R}^{JK \times R^2}$ into $(IR \times KR)$- and $(JR \times KR)$-matrices, respectively. As for the diagonal blocks, the computation of the `MTKRPROD` in $\mathbf{W}^{(1,2)}$ can be sped up by using dimension trees [158].

Complexity

The computational complexity of computing one diagonal block for an $(I \times I \times I)$-tensor is dominated by the complexity of the `MTKRPROD` that is used to compute $\mathbf{G}^{(m,m)}_{\mathrm{nz}}$, which is $\mathcal{O}(I^3R^2)$ flop when computed straightforwardly. Likewise, the computation of one off-diagonal block is dominated by the `MTKRPROD` that is used to compute $\mathbf{W}^{(l,m)}$ and requires $\mathcal{O}(I^3R^2)$ flop. Adding the costs of the three diagonal and the three off-diagonal blocks together, the efficient computation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ requires a total of $\mathcal{O}(I^3R^2)$ flop. Thus, compared to the $\mathcal{O}(I^5R^2)$ flop for the naive computation, the exploitation of the multilinear structure offers an efficiency gain of two orders of magnitude.

6.3 β-divergences

The LS cost function does not perform optimally when the tensor is perturbed by noise that is not i.i.d. additive Gaussian. For this purpose, we consider in this section β-divergences d_β. These divergences, a special class of Bregman divergences [100], are defined for $\beta \in \mathbb{R}$ and interpolate between the Itakura–Saito (IS) divergence ($\beta = 0$), Kullback–Leibler (KL) divergence ($\beta = 1$) and

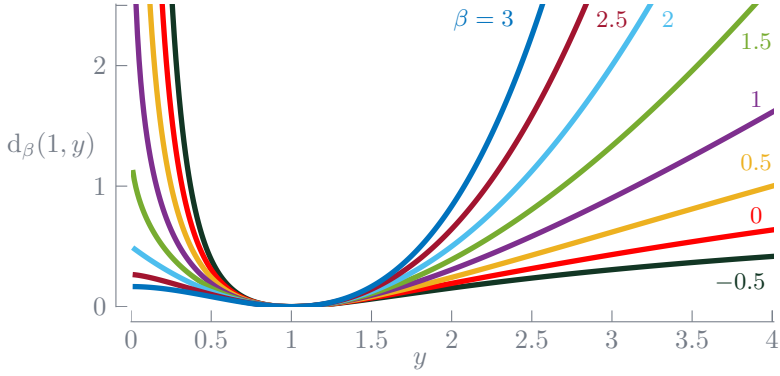

Figure 6.1: Plot of $d_\beta(1, y)$ as a function of y for different values of β. These functions are only convex for $\beta \in [1, 2]$. For other values of β, they are only locally convex in a neighborhood of their minimum.

the LS distance ($\beta = 2$):

$$d_\beta(x, y) = \begin{cases} \frac{x^\beta + (\beta-1)y^\beta - \beta\left(xy^{(\beta-1)}\right)}{\beta(\beta-1)} & \beta \in \mathbb{R} \setminus \{0, 1\}, \\ x \log\left(\frac{x}{y}\right) - x + y & \beta = 1, \\ \frac{x}{y} - \log\left(\frac{x}{y}\right) - 1 & \beta = 0. \end{cases}$$

Note that they are not distances, except for $\beta = 2$, as they are not symmetric and do not satisfy the triangle inequality.

Where for the LS distance, only the absolute difference between two values x and y is considered, other β-divergences also take the size of x and y themselves into account, meaning that $d_\beta(x + c, y + c) \neq d_\beta(x, y)$ for $\beta \neq 2$. More specifically, we can see from the property $d_\beta(kx, ky) = k^\beta d_\beta(x, y)$, with $k > 0$, that the parameter β controls the relative importance of the magnitude of x and y. If $\beta = 0$, the divergence is scale invariant, meaning that $d_0(x, 2x)$ is the same for any value of x. If $\beta > 0$, $d_\beta(x, 2x)$ will be larger for large values of x, while for $\beta < 0$, $d_\beta(x, 2x)$ will be larger for small values of x; see Fig. 6.1 for a visual representation of the β-divergence cost function for some specific values of β. Here x is the target value and y is its approximation. As can be seen from the figure, β-divergences are convex for $\beta \in [1, 2]$. Outside this interval, $d_\beta(x, y)$ is only locally convex, i.e., when $x \approx y$.

If $\beta < 2$, β-divergence cost functions penalize errors on small entries more heavily compared to the LS distance, while for values of $\beta > 2$, the converse is true. Because of this, β-divergences perform well on data with entries of different

magnitudes, such as audio data. For such data, β-divergence cost functions (with $\beta < 2$) manage to capture the low intensity components of the signals better than the LS cost function. β-divergence cost functions have been used extensively in nonnegative matrix factorization (NMF) of audio spectra [72]. Where the LS cost function inherently assumes additive i.i.d. Gaussian noise, NMFs with the KL and IS-divergence have been shown to give the maximum likelihood estimation under the assumption of Poisson-distributed entries or multiplicative Gamma noise, respectively [79]. As a generalization of NMF, β-divergence cost functions have been used to compute nonnegative CPDs of tensors [34, 38, 74, 94, 105, 240]. Only the method from [240], however, updates all factor matrices at the same time. Additionally, none of these methods uses Hessian information and several are only applicable for specific values of β. In contrast, all three properties are satisfied for the method from Sec. 6.2, which we particularize for β-divergence cost functions in this section. If $R = 1$, the computations can be made particularly efficient by exploiting the multilinear structure and the specific form of the β-divergence cost function; see [213].

The β-divergence cost function for a tensor $\mathcal{T}$ and its CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is defined as $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f$ with

$$f = \frac{1}{\beta(\beta-1)} \sum_{n=1}^{N} t_n^{\beta} + (\beta-1) m_n^{\beta} - \beta \left(t_n m_n^{(\beta-1)} \right).$$

For the special cases ($\beta = 1$) and ($\beta = 0$), one has

$$f = \sum_{n=1}^{N} t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \beta = 1,$$

$$f = \sum_{n=1}^{N} \left[\frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1 \right], \quad \beta = 0.$$

When $\beta = 2$, the standard LS cost function is recovered. In the remainder of the chapter, we constrain $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative matrices as β-divergences are only defined for nonnegative values, except when $\beta \in \mathbb{N}\backslash\{1\}$. For the gradient $\mathbf{g} = [\mathrm{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right); \mathrm{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right); \mathrm{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right)]$, one finds, for any value of β,

$$\mathcal{R} = \frac{\mathcal{M} - \mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}.$$

As explained above, the Hessian can be approximated by $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ wherein $\mathbf{Z} = \mathrm{Diag}\left(\mathrm{vec}\left(\mathcal{D}\right)\right)$. The tensor $\mathcal{D}$ is, for any value of β, the entry-wise second-order derivative of the β-divergence cost function:

$$\mathcal{D} = \frac{(\beta-1)\mathcal{M} - (\beta-2)\mathcal{T}}{\mathcal{M}^{\bullet(3-\beta)}} = \frac{(\beta-2)\left(\mathcal{M} - \mathcal{T}\right) + \mathcal{M}}{\mathcal{M}^{\bullet(3-\beta)}}. \tag{6.5}$$

6.3.1 Indefiniteness of the Hessian approximation

The Gramian of the Jacobian $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ for the LS distance is always PSD. Unfortunately, this does not hold for its generalization $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$. One can use a trust-region approach that can handle indefinite Hessian approximation or, alternatively, one can try to force the Hessian approximation to remain PSD during the computation of the CPD. A sufficient condition for $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ being PSD is that all entries of $\mathbf{Z}$ are nonnegative. This corresponds to the entry-wise condition

$$(\beta - 1)\mathcal{M} \geq (\beta - 2)\mathcal{T}. \tag{6.6}$$

For $\beta \in [1, 2]$, the left hand side of the equation is positive and the right hand side is negative, so (6.6) is always satisfied. For $\beta > 2$, the condition becomes $\mathcal{M} \geq [(\beta - 2)/(\beta - 1)]\mathcal{T}$, which bounds the entries of $\mathcal{M}$ from below, with the bound converging to $\mathcal{T}$ as the value of β increases. For $\beta < 1$, the condition is $\mathcal{M} \leq [(2 - \beta)/(1 - \beta)]\mathcal{T}$, now with the bound converging to $\mathcal{T}$ as the value of β decreases. Thus for values of β outside $[1, 2]$, the model has to be close enough to the tensor for $\mathcal{D}$ to be nonnegative, which guarantees that the Hessian approximation will be PSD. Having a good initialization is thus doubly important, as this not only decreases the number of iterations of the β-divergence algorithm, but can also avoid iterations with an indefinite Hessian approximation. We discuss four strategies to avoid indefinite Hessian approximations.

Modifying $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$

If all values of $\mathcal{D}$ are nonnegative, then $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is PSD. The easiest way to ensure this is thus by approximating $\mathcal{D}$ with a nonnegative tensor, for example by setting its negative values to zero or to a small positive value. A Levenberg–Marquardt modification of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is also possible: adding a scaled version of the identity matrix ($\lambda\mathbf{I}$) to $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can make the Hessian approximation PSD. Of course, if λ is large, the second-order information is essentially ignored.

Tempering β

As noted above, the Hessian approximation is always PSD for $\beta \in [1, 2]$. One can leverage this property by first computing a solution for $\beta = 2$, using standard approaches, to get the model close to the tensor followed by a few (more expensive) iterations of the algorithm with the requested value of β. A caveat is that the LS cost function minimizes the squared error on the tensor entries, while $\mathcal{D}$ also depends on entry size (cf. (6.6)). This means that $\mathcal{D}$ can

still be negative for the extreme (smallest or largest, depending on β) entries of the tensor when using this approach. In such cases, gradually increasing or decreasing β until the requested value is reached can be beneficial. The tempering approach is also discussed in [15].

Weighted least-squares

Depending on the value of β, smaller ($\beta < 2$) or larger ($\beta > 2$) tensor entries are fitted more tightly compared to the LS cost function. We can mimic such properties of cost functions by initializing with a low-rank weighted LS approach [16]. A modified problem is solved:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \left\| \mathcal{W} * (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T}) \right\|_{\mathrm{F}}^2 ,$$

where $\mathcal{W}$ is a low-rank weight tensor of the same dimensions as $\mathcal{M}$ and $\mathcal{T}$. For β-divergences with $\beta < 2$, smaller values are given a larger weight in $\mathcal{W}$ and larger values are given a smaller weight. If $\beta > 2$, larger values are given a larger weight and smaller values are given a smaller weight. An example of a weight tensor $\mathcal{W}$ with these properties can be obtained by computing for every tensor entry t_n a second-order Taylor approximation to the cost function f around $m_n = t_n$: $f \approx f(t_n) + f'(t_n)(m_n - t_n) + \frac{f''(t_n)}{2}(m_n - t_n)^2$. Equating terms with the WLS function $w_n^2(m_n - t_n)^2$ gives:

$$w_n^2 \approx \frac{f(t_n)}{(m_n - t_n)^2} + \frac{f'(t_n)}{m_n - t_n} + \frac{f''(t_n)}{2}.$$

For the β-divergence cost function ($\beta \notin \{0, 1\}$), given as

$$f_\beta = \frac{1}{\beta(\beta - 1)} \left(t_n^\beta + (\beta - 1) m_n^\beta - \beta t_n m_n^{(\beta - 1)} \right),$$

one finds

$$f_\beta(t_n) = f'_\beta(t_n) = 0,$$

and

$$f''_\beta(t_n) = t_n^{(\beta - 2)},$$

leading to a weighting factor

$$w_n = \frac{1}{\sqrt{2}} t_n^{\frac{(\beta - 2)}{2}}.$$

For values of $\beta > 2$, larger entries are indeed given larger weights, while the converse is true for $\beta < 2$. For $\beta = 2$, the LS cost function is recovered. In

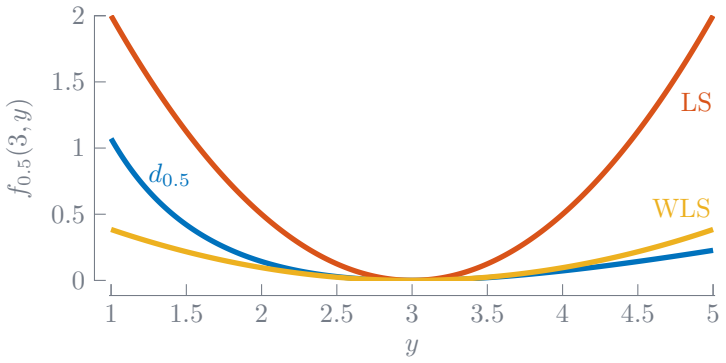

Figure 6.2: The WLS cost function gives a better local approximation to $d_{0.5}(3, y)$ than the LS cost function.

Fig. 6.2, it can be seen that a WLS cost function with the proposed weighting factor gives a better local quadratic approximation to the β-divergence cost function than the LS cost function. A good choice of weight tensor is, for instance, the best rank-1 approximation to $\mathcal{W} = \frac{1}{\sqrt{2}}\mathcal{T}^{\bullet\frac{(\beta-2)}{2}}$, as using a low-rank weight tensor makes the WLS CPD much less costly. As an alternative that is cheaper to compute, one can also take $\mathcal{W} = [\![\mathbf{a}, \mathbf{b}, \mathbf{c}]\!]$, with $\mathbf{a} = \frac{1}{\sqrt[3]{2}}\left(\frac{1}{JK}\sum_{q=1}^{JK}\mathbf{T}_{(1),q}^{\bullet\frac{(\beta-2)}{2}}\right)$, and with $\mathbf{b}$ and $\mathbf{c}$ defined analogously.

Shifting

A fourth way to force $\mathcal{D}$ to be nonnegative, and thus to guarantee that the Hessian approximation is PSD, is to add a constant nonnegative tensor $\mathcal{P}$ to both $\mathcal{T}$ and $\mathcal{M}$. Instead of fitting the model $\mathcal{M}$ to the tensor $\mathcal{T}$, we thus fit the model $\mathcal{M} + \mathcal{P}$ to the tensor $\mathcal{T} + \mathcal{P}$. In this way, we are essentially shifting our problem to a region where the cost function is nicer: the shifted problem ideally lies in the locally convex region of the β-divergence cost function (cf. Fig. 6.1). Of course, shifting introduces a bias, as the β-divergence is not shift-invariant for $\beta \neq 2$. We thus want to gradually decrease the shift back to zero during the optimization process. Revisiting (6.6), requiring $\mathcal{D}$ to be nonnegative introduces the following requirement for the shifting tensor $\mathcal{P}$:

$$\mathcal{P} \geq (\beta - 2)\mathcal{T} - (\beta - 1)\mathcal{M}.$$

In principle, every entry t_n could have its own shift p_n, but taking one common value for certain parts of the tensor or a common p for all tensor entries can

reduce storage requirements. A possible shift that satisfies the bound for all entries is $p = k \max(\mathcal{P})$, with $k \geq 1$ and $\mathcal{P} = (\beta-2)\mathcal{T} - (\beta-1)\mathcal{M}$. If $p < 0$, the Hessian is already PSD, so we can set $p = 0$. In the latter case, setting $p = \delta > 0$ when p is slightly nonnegative may improve the convergence of the algorithm by increasing the size of the locally convex domain around the minimum.

The size of the shift, which depends on the residual $\mathcal{T} - \mathcal{M}$, should decrease as the model gets closer to the tensor and should eventually become zero. The objective function value, gradient and $\mathcal{D}$ are computed with the shifted tensor and model instead of the original tensor and model, but the algorithm itself remains exactly the same. The tensor $\mathcal{P}$ is computed as an intermediate result in the computation of $\mathcal{D}$ (see (6.5)), so the only additional computations that the shifting introduces are the sums $\mathcal{T} + \mathcal{P}$ and $\mathcal{M} + \mathcal{P}$. As $\mathcal{P}$ is a constant, the shift does not increase the dimensions of the gradient and Hessian. A nice byproduct of shifting is that it also avoids possible numerical problems due to tensor entries that are zero. Shifting works better when β is close to the interval $[1, 2]$. When $\beta \ll 1$ or $\beta \gg 2$, the bound in (6.6) becomes tighter and thus a larger shift is needed. If the required shift is too large, it could dominate the original tensor entries. If this happens, it might not be possible to decrease the shift all the way to zero, making the algorithm converge to a suboptimal solution.

6.4 Experiments

A number of experiments are performed on synthetic and real-life data. First, we motivate the use of β-divergence cost functions by comparing the CPDs for different values of β. We then analyze the performance of different strategies to handle the indefiniteness of the Hessian approximation. We show how the proposed GGN algorithm performs compared to standard methods from the literature and compare the computational cost to that of the GN method. Finally, we use our method to approximate a hyperspectral image. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0 [228]. For experiments on synthetic data, entries of factor matrices are sampled independently from the uniform distribution on the interval (a, b), denoted by $U(a, b)$. Writing $\bar{\mathcal{T}}$ for the perturbed tensor $\mathcal{T}$, the signal to noise ratio (SNR) is defined for all noise types as $\mathbb{E}\left(20 \log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)\right)$, where $\mathcal{N} = \bar{\mathcal{T}} - \mathcal{T}$. When the entries of a tensor $\mathcal{T}$ are perturbed by multiplicative Gamma noise, we have $\bar{\mathcal{T}} = \mathcal{T} * \mathcal{N}_\gamma$. The entries of the noise tensor $\mathcal{N}_\gamma$ are sampled from a Gamma distribution with shape α and scale $\frac{1}{\alpha}$, hence, the SNR (in dB) is expected to be $10 \log_{10}(\alpha)$.

6.4.1 Optimal value of β

Three random factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ in $\mathbb{R}_+^{20\times 3}$ are generated with entries sampled from $U(1, 10)$. Then, a rank-3 tensor $\mathcal{T}$ is constructed with these factor matrices, either additive Gaussian noise or multiplicative Gamma noise is added and factor matrices $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ are computed with a β-divergence cost function for different values of β. In Fig. 6.3, the mean absolute factor matrix error $\max(\|\mathbf{A} - \tilde{\mathbf{A}}\|_{\mathrm{F}}, \|\mathbf{B} - \tilde{\mathbf{B}}\|_{\mathrm{F}}, \|\mathbf{C} - \tilde{\mathbf{C}}\|_{\mathrm{F}})$ is shown for an SNR of 40 dB after matching columns and normalizing all factor matrices such that $\|\mathbf{a}_{:r}\| = \|\mathbf{b}_{:r}\| = \|\mathbf{c}_{:r}\|$ and $\|\tilde{\mathbf{a}}_{:r}\| = \|\tilde{\mathbf{b}}_{:r}\| = \|\tilde{\mathbf{c}}_{:r}\|$ for $1 \leq r \leq R$. For tensors that are perturbed by additive Gaussian noise, the smallest errors are obtained for $\beta = 2$, while for multiplicative Gamma noise, the lowest errors are obtained for $\beta = 0$. This is in line with the discussion at the beginning of Sec. 6.3 and motivate the use of different cost functions depending on the distribution of the noise.

β-divergence cost functions with small values of β fit small tensor entries tighter than the LS cost function. To illustrate this, a tensor $\mathcal{T} = \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket$ is generated, where $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}_+^{10\times 3}$ with entries sampled as 10^x with x sampled from $U(-2, 2)$. This tensor is then perturbed with a small amount of noise to show the influence of the parameter β. For Fig. 6.4, the tensor is perturbed with additive Gaussian noise with an SNR of 100 dB and its CPD $\hat{\mathcal{T}}$ is computed for β-divergence cost functions with $\beta = 0$ and $\beta = 4$. A high SNR value is necessary, as the SNR value is dominated by the large tensor entries. With the large dynamic range of the entries in this experiment, the smallest entries would be completely dominated by the additive Gaussian noise for low SNRs. In Fig. 6.5, the same experiment is performed for multiplicative Gamma noise. In both figures, it can be seen that for the cost function with large β, the absolute error $|t_{ijk} - \hat{t}_{ijk}|$ of all tensor entries is approximately the same, independently of their size. As a result, the relative error $|t_{ijk} - \hat{t}_{ijk}|/t_{ijk}$ is smaller for larger tensor entries. For the cost function with small β, the relative error remains about the same along the range of tensor entries. Of course, this means that the absolute error is larger for larger tensor entries.

6.4.2 Handling indefiniteness

In the following experiment, four different strategies for handling indefiniteness of the Hessian approximation are compared: LS and WLS initialization (with a relative step size of 10^{-2} and a relative cost function change of 10^{-4} as stopping criteria), random initialization using shifting with one common shift for the whole tensor ($p = 2\max(\mathcal{P})$), and random initialization without shifting. These strategies are discussed in Subsections 6.3.1 and 6.3.1. Three random

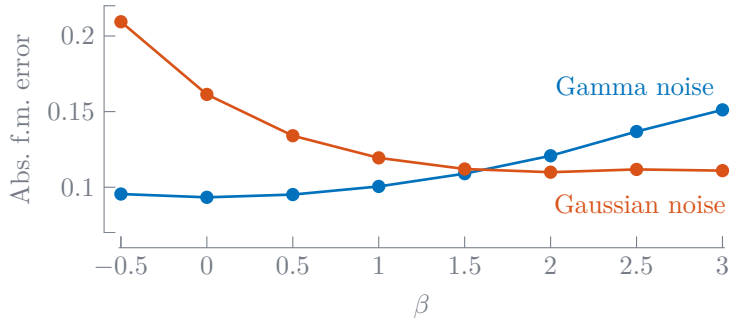

Figure 6.3: The errors for tensors perturbed by Gamma noise are about 29.4% larger when using $\beta = 2$ compared to $\beta = 0$, while for Gaussian noise, using $\beta = 2$ is optimal. The mean (50 trials) absolute factor matrix error is shown for different values of β for tensors perturbed by multiplicative Gamma noise and tensors perturbed by additive Gaussian noise, both with an SNR of 40 dB.

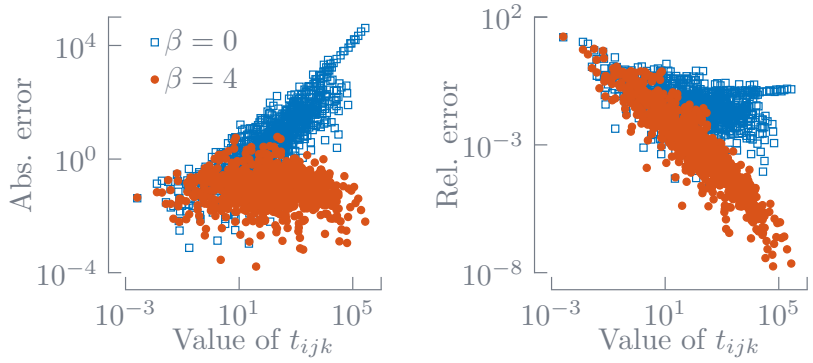

Figure 6.4: The cost function with $\beta = 4$ performs better for *additive Gaussian noise* than the cost function with $\beta = 0$. Absolute (left) and relative (right) error on the tensor entries t_{ijk} when fitting a CPD of a low-rank tensor that is perturbed with *additive Gaussian noise*. Two different β-divergence cost functions are used: $\beta = 0$ and $\beta = 4$. For the former, the relative error is approximately the same for small and large tensor entries. For the latter, larger entries tend to have a smaller relative error than smaller entries.

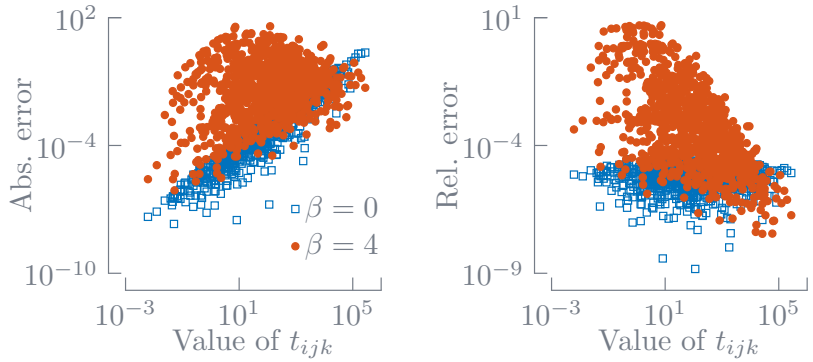

Figure 6.5: For *multiplicative Gamma noise*, the cost function with $\beta = 0$ performs better. Same setting as in Fig. 6.4, but with *multiplicative Gamma noise* instead. For $\beta = 0$, the relative error is approximately the same for small and large tensor entries. For $\beta = 4$, larger entries tend to have a smaller relative error than smaller entries.

factor matrices are generated in $\mathbb{R}_{+}^{50\times 5}$ with entries sampled from $U(0,1)$ and then the associated rank-5 tensor is constructed. We perturb the tensor with multiplicative Gamma noise and an SNR of 20 dB. A relative step size of 10^{-4} and a relative cost function change of 10^{-8} are used as stopping criteria. In Tables 6.1 and 6.2, the median runtime and median number of iterations are shown for 20 tensors and for different values of β. For random initialization and without shifting, the method failed to converge to the correct solution in 8 of the 20 trials. The reported numbers do not take these failed trials into account. From the tables, it can be seen that convergence is much faster, both in time and in number of iterations after computing a LS or WLS initialization. The WLS initialization, with as weighting tensor $\mathcal{W}$ a rank-1 approximation to $\frac{1}{\sqrt{2}}\mathcal{T}^{\bullet\left(\frac{\beta-2}{2}\right)}$, performs slightly better than the LS initialization. When starting from a random initialization, shifting generally does not increase the computation time or number of iterations, while it does fix the convergence problems for random initialization.

6.4.3 Other β-divergence CPD algorithms

The proposed GGN method is compared to the generalized CPD (GCPD) method [105] for a β-divergence cost function and to the classical multiplicative update (MU) algorithm for β-divergences [38]. Random factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C} \in \mathbb{R}^{I\times R}$ are generated with entries sampled from $U(0,1)$, and $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

Table 6.1: The LS and WLS initializations enable faster CPD computations, with WLS performing better for small β. For random initializations, shifting does not influence the computation time much, while it does assure convergence. The median time (s) (over 20 trials) is shown to compute a CPD of a rank-5 tensor $\mathcal{T} \in \mathbb{R}_{+}^{50\times50\times50}$ perturbed by multiplicative Gamma noise with an SNR of 20 dB when using four strategies for handling indefinite Hessians. For LS and WLS initialization, the initialization time is also shown. Only successful trials were used for random initialization without shifting (12 out of 20).

	Time (s) for different β					
	β					
Method	-0.5	0.0	0.5	1.0	1.5	2.5
Init. with LS	0.273	0.288	0.268	0.283	0.273	0.281
Main GGN	2.704	1.590	1.296	1.043	1.072	1.081
Init. with WLS	0.229	0.234	0.271	0.284	0.315	0.388
Main GGN	1.669	1.070	1.088	0.895	0.906	1.285
Random init. + shifting	3.391	2.143	1.885	1.821	1.605	1.725
Random init.	2.984	2.276	2.058	1.735	1.545	2.020

Table 6.2: When starting from an LS or WLS initialization, the number of remaining (costly) GGN iterations until convergence is significantly lower than for random initialization. Shifting does not change the number of iterations when starting from a random initialization, while it does assure that the method converges. The median number of iterations (over 20 trials) is displayed for the same experiment as in Table 6.1.

	Number of iterations for different β					
	β					
Method	-0.5	0.0	0.5	1.0	1.5	2.5
Init. with LS	8.0	8.0	8.0	8.0	8.0	8.0
Main GGN	16.5	11.5	9.0	8.0	7.0	7.5
Init. with WLS	7.5	7.5	8.0	8.0	8.5	7.5
Main GGN	11.0	7.5	7.5	7.0	6.0	8.0
Random init. + shifting	22.0	15.5	13.0	14.0	11.0	12.0
Random init.	19.5	16.0	14.0	13.5	11.0	13.5

In practice, one will often use a cheaper method to get close to the solution, after which the more expensive GGN method can leverage its fast local convergence to get to the true solution. Thus, to focus on the convergence behavior close to the solution, the methods are initialized by generating random factor matrices with entries sampled from $U(0, 1)$ and then applying five iterations of the GN algorithm for $\beta = 2$. The CPD is then computed for different values of β. In Fig. 6.6, the results of the different methods are shown for $I = 20$ and $R = 5$. The median computation time, relative factor matrix error and number of iterations are displayed. All methods use a relative step size of 10^{-10} as a stopping criterion. The GGN and GCPD methods both find the correct solution in all cases. The GGN method requires fewer iterations than the GCPD method, while the MU method does not converge within 1000 iterations. The median computation time of GGN is lower than GCPD, especially when $\beta \notin [1, 2]$.

Fig. 6.7 shows the typical evolution of the cost function value f_β close to the solution, here for $\beta = 0.5$. On the left, it can be seen that the proposed GGN method converges to the true solution after very few iterations. In contrast, the MU-algorithm requires thousands of iterations with slow improvement before converging to the true solution. GCPD requires hundreds of iterations to reach the true solution. While it converges with about the same rate as the GGN method far from the solution, its convergence is clearly slower close to the solution, where the GGN method can fully leverage its second-order information and achieves quadratic convergence.

6.4.4 GGN compared to GN

Using a non-LS cost function naturally adds complexity to the algorithm, as the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ is replaced by the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, where $\mathbf{Z}$ is a diagonal matrix with $\mathrm{vec}(\mathcal{D})$ on its diagonal. As discussed in Sec. 6.2.3, the structure of the factors of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can be exploited in the computation of this matrix, which drastically lowers its computation time. In Fig. 6.8, the computation time of both Hessian approximations is shown for an $(I \times I \times I)$-tensor and a rank-10 CPD. The computation time for the Hessian approximation of the general cost function is larger, but only by a factor 2. Considering that the full tensor $\mathcal{D}$ has to be computed and included in the approximation, this is an acceptable price to pay. Actually, the increase in computation time is much more significant for the cost function and gradient evaluations when replacing the LS cost function by a β-divergence cost function, as can also be seen from Fig. 6.8. For the LS cost function, the computation time for these evaluations is almost negligible compared to the computation time for the Hessian approximation, giving first-order methods an edge compared to second-order methods. This is not the case for the β-divergence cost function, particularly when the tensor

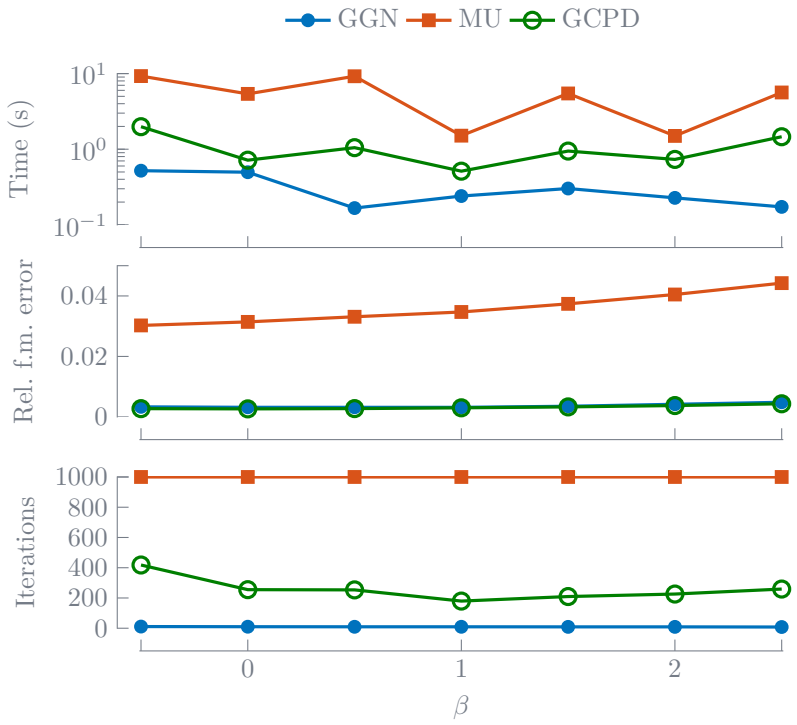

Figure 6.6: The proposed GGN method requires considerably fewer iterations than the other two methods. The performance with respect to time, relative factor matrix error and number of iterations is shown for three methods to compute a rank-5 CPD of a rank-5 $(20 \times 20 \times 20)$-tensor perturbed by multiplicative Gamma noise with an SNR of 40 dB for different values of β. All methods are capped at 1000 iterations and medians over 20 trials are shown.

is large. As a result, even first-order β-divergence CPD methods that only evaluate the cost function and gradient will be significantly more expensive than LS CPD methods. This makes the proposed second-order method appealing for general cost functions, as computing the Hessian approximation is a relatively smaller extra cost.

Both GN and GGN require the solution of the system $\mathbf{Hp} = -\mathbf{g}$ to obtain the step direction. The cost of solving this system is relatively lower for GGN, as the construction of its Hessian approximation is more costly. Still, for larger tensor dimensions, it can be beneficial to solve $\mathbf{Hp} = -\mathbf{g}$ approximately using inexact methods [215].

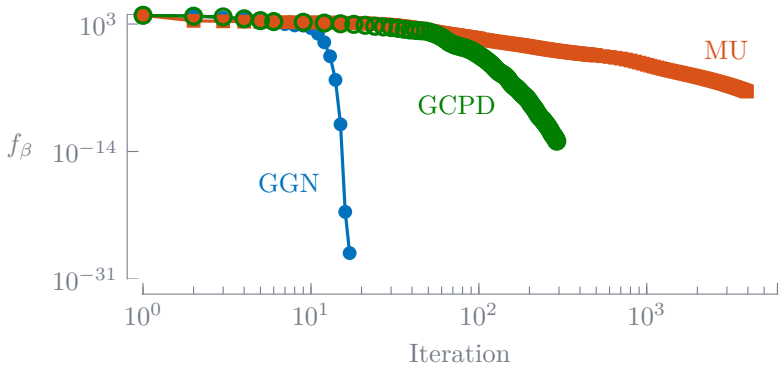

Figure 6.7: The GGN method converges quadratically (until machine precision is reached) close to the solution, while the other methods converge linearly. The typical convergence behavior of a β-divergence CPD with $\beta = 0.5$ is shown for the GGN, MU and GCPD methods on a rank-5 $(40 \times 40 \times 40)$-tensor. The cost function value f_β is plotted in function of the iteration number.

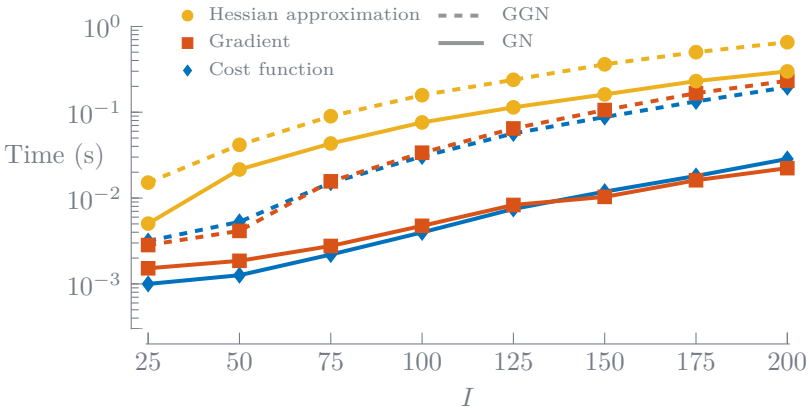

Figure 6.8: The GGN method (dashed) requires about twice as much time to compute the Hessian approximation as the GN method (solid). Second, compared to LS, the β-divergence cost function and gradient evaluations are much more costly. The mean time (over 20 trials) is shown for the computation of one cost function evaluation, one gradient evaluation, and one Hessian approximation in function of the tensor size for the LS (solid) and β-divergence with $\beta = 1.5$ (dashed) cost functions. The GN approximation is used for the Hessian of the LS cost function, while the GGN approximation is used for the β-divergence cost function. The tensors have dimensions $(I \times I \times I)$ and rank 10 and are approximated by a CPD with rank 10.

6.4.5 Hyperspectral image compression

In this example, a part of the DC Mall hyperspectral image [10] of dimensions 80×80 is considered, which contains 160 different wavelengths. The $(I \times J \times K)$-tensor $\mathcal{T}$ representing the hyperspectral image is thus of dimensions $80 \times 80 \times 160$; see Fig. 6.9a for an image at one wavelength. Hyperspectral images can easily become very large, thus to limit the storage requirements, a compressed image can be stored instead. Here, we use the CPD to approximate the hyperspectral image, which can also have a denoising effect on the image [134]. As Poisson noise can often be expected to be introduced during the image generation [161], this type of noise was included (see Fig. 6.9b) to obtain the tensor $\tilde{\mathcal{T}}$. The noise was introduced using the MATLAB command $\tilde{\mathcal{T}} = \texttt{poissrnd}(\mathcal{T}p)/p$, with p the requested noise level (higher p means less noise). This tensor was then approximated by a rank-R CPD, both with an LS and with a KL-divergence cost function, to obtain a compressed tensor $\hat{\mathcal{T}}$. The LS solution was used as an initialization for the KL-divergence CPD. In Figures 6.9c and 6.9d, the results can be seen. Although the difference is not large, the compressed image obtained with the KL-divergence cost function manages to capture the details of the image slightly better. This can also be seen when comparing quality measures of both compressed images for different noise levels and approximation ranks. The peak SNR (PSNR) and the universal image quality index (UIQI) [230] are computed for both CPDs. The former is defined as

$$\text{PSNR}(\mathcal{T}, \hat{\mathcal{T}}) = 10 \log_{10} \left(\frac{\rho^2 IJK}{\|\mathcal{T} - \hat{\mathcal{T}}\|_{\text{F}}^2} \right),$$

with ρ the maximal value of the image, which is simply equal to one in this case. The latter is defined as

$$\text{UIQI}(\mathcal{T}, \hat{\mathcal{T}}) = \frac{1}{K} \sum_{k=1}^{K} \frac{4\sigma^2_{\mathbf{t}_k \hat{\mathbf{t}}_k} \mu_{\mathbf{t}_k} \mu_{\hat{\mathbf{t}}_k}}{\left(\sigma^2_{\mathbf{t}_k} + \sigma^2_{\hat{\mathbf{t}}_k}\right)\left(\mu^2_{\mathbf{t}_k} + \mu^2_{\hat{\mathbf{t}}_k}\right)},$$

with $\mathbf{t}_k = \text{vec}\left(\mathbf{T}_{::k}\right)$ and $\hat{\mathbf{t}}_k = \text{vec}\left(\hat{\mathbf{T}}_{::k}\right)$. The PSNR is based on the LS fit of the CPD to the original tensor and is thus expected to be higher for the solution obtained with the LS cost function. The UIQI was designed specifically to judge the quality of compressed images and reflects the luminance and contrast distortion caused by the compression. The closer the UIQI is to 1, the better the approximation is.

The results are shown in Table 6.3. First, it can be seen that for CPDs with relatively low rank and less noise, the LS solution indeed gives the solution with the highest PSNR. If the rank of the CPD is high, however, or the amount of



Figure 6.9a: Original image



Figure 6.9b: With Poisson noise

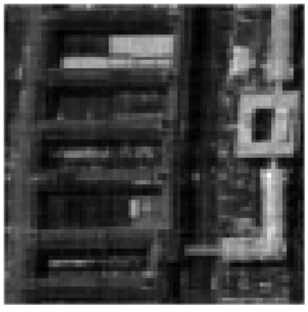

Figure 6.9c: LS cost function

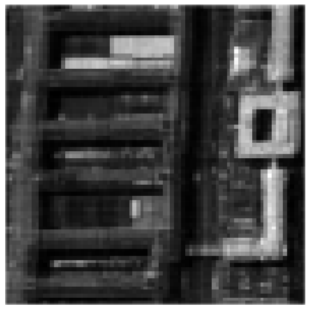

Figure 6.9d: KL cost function

Figure 6.9: Spectral band 120 of a (80×80) aerial image perturbed by Poisson noise with an SNR of 10 dB. The reconstructed images are shown when the tensor is approximated with a rank-40 CPD computed with a LS cost function and with a KL divergence cost function.

noise is increased, the KL-divergence solution actually has the largest PSNR. In these cases, the change of cost function allows the optimization algorithm to escape from the suboptimal solution that was found with the LS cost function and find a better local minimum. Second, one can see that the KL-divergence cost function consistently scores slightly better than the LS cost function with respect to the UIQI. Because the tensor has been perturbed with Poisson noise, the KL-divergence cost function manages to preserve the luminance and contrast of the original image better than the LS cost function, even when the latter offers a solution with a higher PSNR.

Table 6.3: Comparison of the PSNR (left) and the UIQI (right) for a hyperspectral image approximated by a CPD with either a LS or KL-divergence cost function. Different SNRs and approximation ranks R are considered. Best results of five initializations are shown. The KL-divergence cost function consistently gives the solution with the highest UIQI, even if the PSNR of the LS solution is higher.

		PSNR		UIQI	
SNR	R	LS	KL	LS	KL
10 dB	10	25.2953	**25.3192**	0.8777	**0.8847**
	20	25.8925	**27.7919**	0.9244	**0.9402**
	40	27.1162	**30.5213**	0.9504	**0.9671**
20 dB	10	**25.4364**	25.2722	0.8833	**0.8844**
	20	**27.7218**	27.6653	0.9337	**0.9347**
	40	27.4485	**30.6091**	0.9550	**0.9672**

6.5 Conclusion

A second-order all-at-once CPD algorithm was proposed that can handle general entry-wise cost functions. By exploiting the low-rank structure of the problem, the computational cost of the algorithm was significantly limited. The method was applied to β-divergence cost functions, which are tailored to data with non-Gaussian noise. The method was shown to outperform the widely used first-order approaches and to show fast convergence close to the solution in low-noise cases.

6.A Proofs of Sec. 6.2.3

Theorem 6.A.1:
Let $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ be a CPD of $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ and $\mathbf{Z} = \mathrm{Diag}(\mathrm{vec}(\mathcal{D}))$. Then the nonzero entries $\mathbf{G}_{\mathrm{nz}}^{(1,1)}$ of the matrix

$$\mathbf{G}^{(1,1)} = ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\mathrm{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I) \in \mathbb{R}^{I \times I}$$

can be computed as follows:

$$\mathbf{G}_{\mathrm{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \right).$$

Proof. Consider the case $R = 1$. Then $\mathbf{c} \odot \mathbf{b}$ is a vector holding all products of an entry of $\mathbf{c}$ and an entry of $\mathbf{b}$. The Kronecker product with the identity then replaces every element $c_x b_y$ of this vector with $c_x b_y$ times the $(I \times I)$-identity matrix. Thus $[(\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I]$ looks like this:

$$\begin{bmatrix} c_1 b_1 \mathbf{I}_I & \cdots & c_1 b_J \mathbf{I}_I & c_2 b_1 \mathbf{I}_I & \cdots & c_K b_J \mathbf{I}_I \end{bmatrix}^{\mathrm{T}}.$$

The product $((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\mathrm{T}} \mathbf{Z} ((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I) \in \mathbb{R}^{I \times I}$ can then be computed block-wise:

$$\begin{aligned} &((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\mathrm{T}} \operatorname{Diag}\left(\begin{bmatrix} \mathbf{d}_{:11}^{\mathrm{T}}, \cdots, \mathbf{d}_{:JK}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \right) ((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I) \\ &= c_1^2 b_1^2 \operatorname{Diag}(\mathbf{d}_{:11}) + \cdots + c_1^2 b_J^2 \operatorname{Diag}(\mathbf{d}_{:J1}) \\ &\quad + c_2^2 b_1^2 \operatorname{Diag}(\mathbf{d}_{:12}) + \cdots + c_K^2 b_J^2 \operatorname{Diag}(\mathbf{d}_{:JK}). \end{aligned}$$

Note that this is a diagonal matrix. We can compute only the nonzero values by removing the diagonalization, which gives us the vector $\mathbf{G}_{\mathrm{nz}}^{(1,1)} \in \mathbb{R}^{I \times 1}$:

$$\begin{aligned} \mathbf{G}_{\mathrm{nz}}^{(1,1)} &= c_1^2 b_1^2 \mathbf{d}_{:11} + \cdots + c_1^2 b_J^2 \mathbf{d}_{:J1} + c_2^2 b_1^2 \mathbf{d}_{:12} + \cdots + c_K^2 b_J^2 \mathbf{d}_{:JK} \\ &= \mathbf{D}_{(1)} \left(\mathbf{c}^{\bullet 2} \odot \mathbf{b}^{\bullet 2} \right). \end{aligned} \tag{6.7}$$

For the general case where $R > 1$, the matrix $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\mathrm{T}}$ is of the form

$$\begin{bmatrix} c_{11} b_{11} \mathbf{I}_I & \cdots & c_{11} b_{J1} \mathbf{I}_I & c_{21} b_{11} \mathbf{I}_I & \cdots & c_{K1} b_{J1} \mathbf{I}_I \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{1R} b_{1R} \mathbf{I}_I & \cdots & c_{1R} b_{JR} \mathbf{I}_I & c_{2R} b_{1R} \mathbf{I}_I & \cdots & c_{KR} b_{JR} \mathbf{I}_I \end{bmatrix}.$$

After computing $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\mathrm{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)$, one then obtains R^2 sums of the form:

$$\begin{aligned} &c_{1x} c_{1y} b_{1x} b_{1y} \mathbf{d}_{:11} + \cdots + c_{1x} c_{1y} b_{Jx} b_{Jy} \mathbf{d}_{:J1} \\ &\quad + c_{2x} c_{2y} b_{1x} b_{1y} \mathbf{d}_{:12} + \cdots + c_{Kx} c_{Ky} b_{Jx} b_{Jy} \mathbf{d}_{:JK} \end{aligned}$$

for $x, y \in \{1, \ldots, R\}$. This means that we now need all entry-wise products of two columns of $\mathbf{C}$ and of two columns of $\mathbf{B}$, as a generalization of the factors $\mathbf{c}^{\bullet 2}$ and $\mathbf{b}^{\bullet 2}$ in (6.7). These are obtained by computing $(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C})$ and $(\mathbf{B} \odot^{\mathrm{T}} \mathbf{B})$, respectively. This leads us to the expression

$$\mathbf{G}_{\mathrm{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \right),$$

which holds all nonzero values of $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\mathrm{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)$. □

Theorem 6.A.2:
Let $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ be a CPD of $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$. Also, let $\mathbf{Z} = \text{Diag}(\text{vec}(\mathcal{D}))$ and $\mathbf{W}^{(1,2)}$ be the matrix $\mathbf{D}_{(3)}^{\mathrm{T}}(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \in \mathbb{R}^{IJ \times R^2}$ reshaped into an $(IR \times JR)$-matrix. Then the matrix

$$\mathbf{G}^{(1,2)} = \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right)^{\mathrm{T}} \mathbf{Z} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J\right) \in \mathbb{R}^{IR \times JR}$$

can be computed as follows:

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{b}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{b}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,2)}.$$

Proof. Consider the case $R = 1$, where the goal is to compute $\mathbf{G}^{(1,2)} = \left((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I\right)^{\mathrm{T}} \mathbf{Z}\mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{c} \odot \mathbf{a}) \otimes \mathbf{I}_J\right)$. This product can again be computed block by block, where the kth block contains all rows of the Jacobians $\mathbf{J_A}$ and $\mathbf{J_B}$ that contain the factor c_k and the kth column of $\mathbf{D}_{(3)}^{\mathrm{T}}$:

$$\left((\mathbf{c}\odot\mathbf{b}) \otimes \mathbf{I}_I\right)^{\mathrm{T}} \text{Diag}\left(\left[\mathbf{d}_{(3)_{:1}}^{\mathrm{T}}, \cdots, \mathbf{d}_{(3)_{:K}}^{\mathrm{T}}\right]^{\mathrm{T}}\right) \mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{c}\odot\mathbf{a}) \otimes \mathbf{I}_J\right)$$

$$= c_1^2 \left(\mathbf{b}^{\mathrm{T}} \otimes \mathbf{I}_I\right) \text{Diag}\left(\mathbf{d}_{(3)_{:1}}^{\mathrm{T}}\right) \left(\mathbf{I}_J \otimes \mathbf{a}\right)$$

$$+ \cdots + c_K^2 \left(\mathbf{b}^{\mathrm{T}} \otimes \mathbf{I}_I\right) \text{Diag}\left(\mathbf{d}_{(3)_{:K}}^{\mathrm{T}}\right) \left(\mathbf{I}_J \otimes \mathbf{a}\right).$$

The structure of these terms can then be exploited to obtain:

$$\mathbf{G}^{(1,2)} = c_1^2 \text{ unvec}\left(\mathbf{d}_{(3)_{:1}}^{\mathrm{T}}\right) * \left(\mathbf{a}\mathbf{b}^{\mathrm{T}}\right) + \cdots + c_K^2 \text{ unvec}\left(\mathbf{d}_{(3)_{:K}}^{\mathrm{T}}\right) * \left(\mathbf{a}\mathbf{b}^{\mathrm{T}}\right).$$

Computing $\mathbf{D}_{(3)}^{\mathrm{T}}\left(\mathbf{c}^{\bullet 2}\right) \in \mathbb{R}^{IJ \times 1}$ and reshaping this matrix into the $(I \times J)$-matrix $\mathbf{W}^{(1,2)}$ then gives:

$$\mathbf{G}^{(1,2)} = \left(\mathbf{a}\mathbf{b}^{\mathrm{T}}\right) * \mathbf{W}^{(1,2)}.$$

For the general case where $R > 1$, again, all combinations of columns can be handled separately. One now needs all entry-wise products of two columns of $\mathbf{C}$ and each of these terms should be matched with its corresponding outer product of a column of $\mathbf{A}$ and a column of $\mathbf{B}$, thus $\mathbf{c}_{:x}\mathbf{c}_{:y}$ should be matched with $\mathbf{a}_{:x}\mathbf{b}_{:y}^{\mathrm{T}}$. The matrix $\mathbf{W}^{(1,2)} \in \mathbb{R}^{IR \times JR}$ is then obtained by reshaping $\mathbf{D}_{(3)}^{\mathrm{T}}\left(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}\right) \in \mathbb{R}^{IJ \times R^2}$ into an $(I \times J \times R \times R)$-tensor, permuting its modes

to get an $(I \times R \times J \times R)$-tensor and finally reshaping this tensor into $\mathbf{W}^{(1,2)}$. The full $(1,2)$ off-diagonal block then becomes:

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{b}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{b}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,2)}.$$

□

7 Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions

ABSTRACT

The canonical polyadic decomposition (CPD) allows one to extract compact and interpretable representations of tensors. Several optimization-based methods exist to fit the CPD of a tensor for the standard least-squares (LS) cost function. Extensions have been proposed for more general cost functions such as β-divergences as well. For these non-LS cost functions, a generalized Gauss–Newton (GGN) method has been developed. This is a second-order method that uses an approximation of the Hessian of the cost function to determine the next iterate and with this algorithm, fast convergence can be achieved close to the solution. While it is possible to construct the full Hessian approximation for small tensors, the exact GGN approach becomes too expensive for tensors with larger dimensions, as found in typical applications. In this chapter, we therefore propose to use an inexact GGN method and provide several strategies to make this method scalable to large tensors. First, the approximation of the Hessian is only used implicitly and its multilinear structure is exploited during Hessian-vector products, which greatly improves the scalability of the method. Next, we show that by using a compressed instance of the GGN Hessian approximation, the computation time of the inexact GGN method can be lowered even more, with only limited influence on the convergence speed. We also propose dedicated preconditioners for the problem. Further, the maximum likelihood estimator for Rician distributed data is examined in detail as an example of an alternative cost function. This cost function is useful for the analysis of the moduli of complex data, as in functional magnetic resonance imaging, for instance. We compare the proposed method to the existing CPD methods and demonstrate the method's speed and effectiveness on synthetic and simulated real-life data. Finally, we show that the method can scale by using randomized block sampling.

This chapter is based on M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions," IEEE Journal of Selected Topics in Signal Processing, vol. 15, pp. 491–505, 4 2021. The specific contributions of the doctoral candidate are listed in Section 11.1.

7.1 Introduction

Many applications involve data with inherent higher-order structure, i.e., depending on more than two variables. Most of this higher-order structure is lost, however, when the information is stored in the traditional matrix format. Therefore, tensors, higher-order generalizations of vectors and matrices, have been proposed as a way to analyze such data with preservation of their higher-order structure. Tensors appear extensively within the fields of machine learning and signal processing and different kinds of tensor decompositions have been developed to enable better analysis of higher-order data [35, 174]. Even more than matrix decompositions, tensor decompositions give a compact representation of the data while maintaining their important features. This often has a denoising effect as the noise is generally not modeled by the decomposition. For large tensors, decompositions can also significantly reduce the number of parameters that are required to represent the data. One specific type of tensor decomposition is the canonical polyadic decompostion (CPD). This decomposition is a powerful tool in, for instance, signal separation applications, as it allows meaningful components to be extracted from the data. The CPD is particularly appealing due to its simplicity and mild uniqueness conditions [58, 123]. Aside from algebraic algorithms—see, e.g., [62]—several optimization-based methods have been developed to compute the CPD of a given tensor, as described in, e.g., [157, 183, 223]. These methods generally assume that the rank of the CPD is known in advance. Rank estimates typically follow from domain knowledge or experimentation. Alternatively, automatic rank determination methods can find the appropriate rank [7, 22, 32, 144].

Typically, optimization-based methods use the least-squares (LS) distance as cost function to fit the CPD. The LS cost function is well suited for applications involving additive, independent and identically distributed (i.i.d.) errors from a Gaussian distribution, which is (approximately) the case in many applications [146]. However, these assumptions do not always hold and in specific settings better alternatives for the cost function exist. These alternative cost functions allow other statistical assumptions, resulting in new methods that can compute a CPD of a tensor with a range of non-LS cost functions. Non-LS cost functions are particularly useful when working with nonnegative data, as more meaningful results can usually be achieved with cost functions that are specifically tailored to such data. For example, the Kullback–Leibler (KL) and Itakura–Saito (IS) divergence cost functions—β-divergence [100] cost functions with $\beta = 1$ and $\beta = 0$, respectively—have been shown to give the maximum likelihood estimation under the assumption of Poisson-distributed entries or multiplicative Gamma noise, respectively [79]. The former typically arises when analyzing count data [40], while the latter occurs in audio [79] and

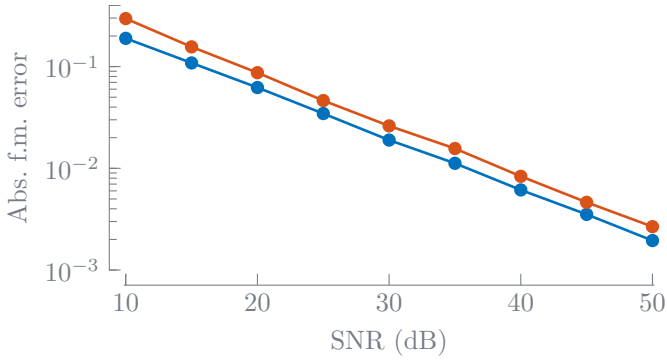

Figure 7.1: When a tensor is perturbed by multiplicative Gamma noise, fitting a CPD with the IS-divergence cost function consistently leads to a better solution than when the LS cost function is used. Median absolute factor matrix errors are shown over 50 trials after matching and normalizing columns. The tensors have dimensions $20 \times 20 \times 20$ and the unperturbed tensors and the CPDs both have rank 5.

image analysis [194], for instance in synthetic aperture radar images perturbed by speckle noise [18]. In Fig. 7.1, it is demonstrated that for tensors perturbed by multiplicative Gamma noise, the CPD obtained with the IS-divergence cost function consistently provides a better fit than the CPD obtained with the LS cost function. As another example, when fitting Rician distributed data, the maximum likelihood estimate of the underlying parameters is obtained using a particular non-LS cost function [126]. Rician distributed data arises, for instance, when the real and imaginary parts of a complex signal are captured independently and equal-variance additive noise is present on both parts, which is typical in medical applications such as MRI analysis [90].

In [34,38,74,78,94,105,235,240] first-order, gradient-based methods to compute a CPD with general cost functions are given. To improve the convergence speed of the algorithms, second-order information can be used as well [81,141,150]. Second-order methods find the next step direction $\mathbf{p}$ by solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$, wherein $\mathbf{H}$ is (a good approximation of) the Hessian matrix and $\mathbf{g}$ is the gradient. Although the per-iteration cost of second-order methods is usually higher than that of first-order methods, this is often more than compensated by their lower number of iterations. For non-LS cost functions, cost function and gradient evaluations are generally much more expensive in a relative sense than for the LS cost function, which makes it easier for second-order methods to be competitive. In Chapter 6, a second-order method is proposed in the form of a generalized Gauss–Newton (GGN) method. Instead of the Gauss–Newton (GN)

Hessian approximation $\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{J}$, which only depends on the Jacobian matrix $\mathbf{J}$, an approximation of the form $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is used, wherein the diagonal matrix $\mathbf{Z}$ depends on the specific cost function that is used. For the CPD, the GGN Hessian approximation lies very close to the true Hessian. At the same time, it is highly structured, which can, and should, be exploited to minimize the per-iteration cost. Consequently, the GGN method often outperforms first-order methods with respect to computation time; see Chapter 6. Unfortunately, this method requires the construction and inversion of a full approximate Hessian, which take $\mathcal{O}(I^3R^2)$ and $\mathcal{O}(I^3R^3)$ flop, respectively, for an $I \times I \times I$ tensor of rank R. Although this cost is manageable for small to medium sized tensors, this is no longer the case for larger tensors.

To improve the scalability of second-order methods for non-LS CPD fitting, we propose to use an inexact GGN method. Instead of constructing a full Hessian approximation, its information is accessed implicitly and an iterative solver such as (preconditioned) conjugate gradient (CG) [195,223] or MINRES [153] is used to find the next step direction $\mathbf{p}$. As a result, the computational cost of the GGN iterations is significantly reduced, which makes the method applicable for larger tensors than those that can be handled by the exact GGN method. Still, a naive implementation of the inexact GGN method is too costly for larger tensors. To make the inexact GGN approach scalable, we propose several strategies to reduce both the per-iteration cost and the number of CG/MINRES iterations. The properties of the proposed method are illustrated in detail for a non-LS cost function which is specifically tailored for Rician distributed data [126].

The main contributions in this chapter are threefold. First, to limit the computational cost of solving the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ iteratively via CG/MINRES iterations, we carefully exploit the structure of $\mathbf{J}$ to significantly lower the cost of products of the form $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$. Second, we reduce the cost of the CG/MINRES iterations even further by approximating the second-order information with a low multilinear rank approximation (LMLRA) [208]. Note that the LMLRA is applied in a way that is fundamentally different from the standard multilinear compression CANDELINC [27]. While methods exist that approximate the entire Hessian approximation matrix $\hat{\mathbf{H}}$ [81], which in our case is equal to $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, we are—at least to our knowledge—the first to approximate only the $\mathbf{Z}$-matrix holding the second-order information. In this way, the cost of each CG/MINRES iteration is further decreased, while all first-order information in $\mathbf{J}$ is maintained and its structure can still be leveraged in the CG/MINRES computations. Finally, we tailor general-purpose preconditioners into dedicated preconditioners for this problem. Due to the exploitation of the specific structure of these preconditioners, they are much cheaper to apply than generic preconditioners, while they can significantly decrease the number of CG/MINRES iterations that are required to solve

$\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$. Together, these contributions make the inexact GGN approach applicable to larger tensors: they allow fast non-stochastic computation of the CPD with arbitrary twice-differentiable cost functions when the construction of the full Hessian approximation is prohibitively expensive. As the proposed techniques are independent of the specific solver that is used, they can be applied in other second-order algorithms to increase their scalability as well. We illustrate this by applying the techniques to a randomized block sampling (RBS) [222] CPD algorithm. This method finds the CPD in a stochastic way by repeatedly sampling small blocks from the tensor. Combining RBS with our proposed techniques enables it to work with non-LS cost functions as well, opening up a whole new class of applications.

In the remainder of this section, notation is fixed and basic definitions are given. The inexact GGN algorithm is proposed in Sec. 7.2, while in Sec. 7.3, we examine Rician distributed data and the cost function that gives the maximum likelihood solution for such data. Lastly, numerical experiments are shown in Sec. 7.4. For simplicity of notation, third-order tensors are considered in this chapter, but with minor modifications, the results extend to higher orders; see supplemental material.

7.1.1 Notation

We denote scalars, vectors, matrices and tensors by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase letters ($\mathbf{A}$) and calligraphic letters ($\mathcal{A}$), respectively. Powers, divisions and logarithms of matrices are entry-wise. For matrices $\mathbf{X}$ and $\mathbf{Y}$, the former is written as $\mathbf{X}^{\bullet a}$, for the others their scalar notation is used: $\frac{\mathbf{X}}{\mathbf{Y}}$ and $\log(\mathbf{X})$, respectively. The transpose of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\mathrm{T}}$ and $\mathrm{Diag}(\mathbf{x})$ forms a square matrix with $\mathbf{x}$ as its diagonal. When using subscripts to specify entries, a colon refers to all entries of a specific mode, e.g., $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. The $M \times M$-identity matrix is written as $\mathbf{I}_M$ and the Moore–Penrose pseudoinverse of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\dagger}$. As a higher-order extension of the vectorization $\mathrm{vec}(\mathbf{X}) \in \mathbb{R}^{IJ\times 1}$ of a matrix $\mathbf{X} \in \mathbb{R}^{I\times J}$, the vectorization $\mathbf{t} = \mathrm{vec}(\mathcal{T}) \in \mathbb{R}^{IJK\times 1}$ of the tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ maps t_{ijk} to t_q, with $q = i+(j-1)I+(k-1)IJ$. Its inverse (with context-implied dimensions) is denoted by $\mathrm{unvec}(\mathbf{t})$. A mode-$n$ vector of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfolding $\mathbf{T}_{(n)}$ of a tensor $\mathcal{T}$ is constructed by stacking all its mode-n vectors as columns of a matrix. For example, $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{i_2,q}$, with $q = i_1 + (i_3 - 1)I$. We also introduce the permutation matrices $\mathbf{\Pi}^{(n)} \in \{0,1\}^{IJK\times IJK}$ for vectorizations of a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$. The matrix $\mathbf{\Pi}^{(n)}$ permutes the modes of $\mathcal{T}$ directly in its

vectorization by bringing the nth mode to the first position. Thus $\mathbf{\Pi}^{(1)} = \mathbf{I}_{IJK}$ and $\mathbf{\Pi}^{(2)}$ and $\mathbf{\Pi}^{(3)}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(3,1,2)$, respectively. The inverse permutation is denoted by $\mathbf{\Pi}^{(n)^{\mathrm{T}}}$, thus bringing the first mode to the nth position. Again, $\mathbf{\Pi}^{(1)^{\mathrm{T}}} = \mathbf{I}_{IJK}$, while $\mathbf{\Pi}^{(2)^{\mathrm{T}}}$ and $\mathbf{\Pi}^{(3)^{\mathrm{T}}}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(2,3,1)$, respectively. The Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK \times JL}.$$

For $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$, the column-wise Khatri–Rao product is defined as $\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_{:1} \otimes \mathbf{b}_{:1} & \cdots & \mathbf{a}_{:K} \otimes \mathbf{b}_{:K} \end{bmatrix}$, while the row-wise Khatri–Rao product of $\mathbf{A} \in \mathbb{R}^{K \times I}$ and $\mathbf{B} \in \mathbb{R}^{K \times J}$ is $\mathbf{A} \odot^{\mathrm{T}} \mathbf{B} = \left(\mathbf{A}^{\mathrm{T}} \odot \mathbf{B}^{\mathrm{T}}\right)^{\mathrm{T}}$. The Hadamard, or entry-wise, product of $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I \times J}$ is written as $\mathbf{A} * \mathbf{B}$. The outer product of vectors is denoted by $\otimes$ and the mode-n tensor-matrix product of $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\mathbf{X} \in \mathbb{R}^{J \times I_n}$ is written as $\mathcal{T} \cdot_n \mathbf{X}$. This product replaces every mode-n vector $\mathbf{t}$ of $\mathcal{T}$ by $\mathbf{X}\mathbf{t}$. For example, $\mathcal{T} \cdot_1 \mathbf{X} \in \mathbb{R}^{J \times I_2 \times I_3}$ is obtained by replacing every mode-1 vector $\mathbf{t}_{:jk}$ of $\mathcal{T}$ by $\mathbf{X}\mathbf{t}_{:jk}$.

A third-order tensor is a rank-1 tensor if it is the outer product of three nonzero vectors. The minimal number of terms needed to write the tensor as a sum of rank-1 tensors, is the rank R. Such a decomposition is called a CPD:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r =: [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

The column vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ are collected as columns of factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively. For a CPD $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, the mode-n unfoldings $\mathbf{T}_{(1)}$, $\mathbf{T}_{(2)}$ and $\mathbf{T}_{(3)}$ are

$$\mathbf{A}\left(\mathbf{C} \odot \mathbf{B}\right)^{\mathrm{T}}, \ \mathbf{B}\left(\mathbf{C} \odot \mathbf{A}\right)^{\mathrm{T}}, \text{ and } \mathbf{C}\left(\mathbf{B} \odot \mathbf{A}\right)^{\mathrm{T}}, \tag{7.1}$$

respectively. Finally, an LMLRA [35, 174, 208] of a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is defined as

$$\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W} =: [\![\mathcal{S}; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!],$$

with $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U} \in \mathbb{R}^{I_1 \times R_1}$, $\mathbf{V} \in \mathbb{R}^{I_2 \times R_2}$, $\mathbf{W} \in \mathbb{R}^{I_3 \times R_3}$ and $R_1 \leq I_1$, $R_2 \leq I_2$, $R_3 \leq I_3$. The multilinear rank of a tensor is the collection of ranks R_n of the mode-n unfoldings of the tensor and is written as (R_1, R_2, R_3).

7.2 CPD of tensors with general cost functions

Optimization methods that use second-order information generally perform well at fitting the CPD of a tensor [35, 152, 159, 174, 183, 204, 206, 223]. Although the per-iteration cost of second-order methods is higher than that of first-order methods, the asymptotic convergence behavior of the former is superlinear (or even quadratic), compared to linear for the latter [150]. In practice, one generally observes a significant reduction in the number of iterations required to obtain the CPD compared to first-order methods, even before the asymptotic behavior kicks in [214, 223]. Unfortunately, the per-iteration computational cost of second-order methods can become prohibitively high for larger tensors. Consequently, strategies have been developed to decrease this cost. For example, for the GN algorithm, exploiting the multilinear structure of the CPD can drastically reduce the per-iteration cost [183]. As a result, the method can leverage its lower number of iterations compared to first-order methods to obtain the solution faster [183, 223].

Reducing the number of iterations is even more important for non-LS cost functions, as their cost function and gradient evaluations are relatively more costly; see Chapter 6. For example, for β-divergences, the entry-wise cost functions require the evaluation of logarithms and non-integer powers, while for the Rician cost function, they require the evaluation of modified Bessel functions. Both are much more expensive than the quadratic error of the LS cost function. Additionally, the standard tricks to avoid computing all tensor entries that are used in the LS case no longer work for non-LS cost functions. For example, first computing a multilinear rank-(R, R, R) LMLRA of a tensor, then computing a rank-R CPD of the $R \times R \times R$ core and finally expanding the factors to the original tensor dimensions [21] only works for the LS distance. Consequently, compared to the LS cost function, even first-order methods are much more expensive for large tensors.

The CPD GN algorithm for the LS cost function has been generalized to arbitrary twice differentiable cost functions in Chapter 6. This GGN method requires much fewer iterations than first-order methods, with only a moderate increase of the per-iteration cost for tensors of small to moderate dimensions. Unfortunately, forming the approximate Hessian becomes prohibitively expensive for larger tensors. Therefore, we propose an inexact GGN method that does not require the full Hessian to be computed, but salvages its good convergence properties.

7.2.1 GGN for non-LS cost functions

Optimization-based algorithms typically fit a rank-R CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ to a given tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ by minimizing the LS distance between the model and the tensor. With the vector $\mathbf{z} = [\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$ holding all optimization variables, the CPD can thus be obtained by solving $\min_{\mathbf{z}} f(\mathbf{z})$, with

$$f(\mathbf{z}) = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \frac{1}{2} \left(m_{ijk} - t_{ijk}\right)^2 .$$

For convenience, we replace the triple sum over all tensor entries by the single sum $\sum_{n=1}^{IJK}$. As discussed in Chapter 6, the LS cost function for the CPD can be generalized to

$$f(\mathbf{z}) = \sum_{n=1}^{IJK} f_n(m_n(\mathbf{z})), \tag{7.2}$$

with $m_n(\mathbf{z})$ mapping the optimization variables in $\mathbf{z}$ to the tensor elements in the current CPD estimate $\mathcal{M}$ of the optimization algorithm. Each f_n is a twice differentiable fit function for the corresponding model entry m_n. Aside from the LS fit function $f_n = (m_n - t_n)^2$, other examples of cost functions are given by β-divergences [100]:

$$f_n = \frac{1}{\beta(\beta-1)} \left(t_n^{\beta} + (\beta-1)m_n^{\beta} - \beta \left(t_n m_n^{(\beta-1)}\right)\right),$$

with $\beta = 1$ (KL-divergence) and $\beta = 0$ (IS-divergence) as special cases:

$$f_n = t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \beta = 1,$$

$$f_n = \left[\frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1\right], \quad \beta = 0.$$

The correntropy function [133] is another alternative, as is the maximum likelihood estimator for data that follows a Rician distribution [126]; see (7.9) in Sec. 7.3. For a more extensive overview of non-LS cost functions, see, for instance, [105].

Gradient descent methods use the direction of the negative gradient to obtain a new intermediate solution $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha\mathbf{p}$. They thus require only the gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ of (7.2) to be computed. In contrast, second-order optimization methods find the next model estimate by solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ for the step direction $\mathbf{p}$, where $\mathbf{H} = \nabla^2 f \in \mathbb{R}^{(I+J+K)R \times (I+J+K)R}$ is the Hessian matrix of f. By also using information from the Hessian, one typically obtains a better

local approximation of the cost function and therefore better steps are taken and fewer iterations are required to reach the solution. As the computation of the full Hessian is usually prohibitively expensive, an approximate Hessian $\hat{\mathbf{H}}$ is often used in practice. The new step direction $\mathbf{p}$ is then obtained from the approximate system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$. This direction can be combined with the (first-order) Cauchy point within a (dogleg) trust-region approach [150] to guarantee global convergence of the method to a stationary point. With the trust region, the step is restricted to a small region around the previous iterate, wherein the local second-order approximation of the cost function can be trusted.

In [223], expressions for the gradient and Hessian are derived. The gradient $\mathbf{g}$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g}_\mathbf{A}; \mathbf{g}_\mathbf{B}; \mathbf{g}_\mathbf{C}]$. By exploiting the multilinear structure, the subgradients $\mathbf{g}_\mathbf{A}$, $\mathbf{g}_\mathbf{B}$ and $\mathbf{g}_\mathbf{C}$ can be computed as

$$\mathbf{g}_\mathbf{A} = \operatorname{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right),$$

$$\mathbf{g}_\mathbf{B} = \operatorname{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right),$$

$$\mathbf{g}_\mathbf{C} = \operatorname{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right),$$

where the entries of the tensor $\mathcal{R} \in \mathbb{R}^{I \times J \times K}$ are defined as:

$$r_n = \frac{\partial f_n}{\partial m_n},$$

for $1 \leq n \leq IJK$. For example, for the LS cost function, $\mathcal{R}$ is simply the residual tensor $\mathcal{M} - \mathcal{T}$. The subgradients $\mathbf{g}_\mathbf{A}$, $\mathbf{g}_\mathbf{B}$ and $\mathbf{g}_\mathbf{C}$ all have the same structure: they are products of an unfolded tensor and a Khatri–Rao product of matrices. This matricized tensor times Khatri–Rao product (`MTKRPROD`) is well-studied and can be computed efficiently using specialized routines, either sequentially or in parallel [96, 218, 223, 226].

For the CPD fitting problem with general cost functions (7.2), the Hessian matrix $\mathbf{H}$ is defined as

$$\mathbf{H} = \sum_{n=1}^{IJK} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \mathbf{J}_{n:} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n} \left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right). \tag{7.3}$$

The first term in (7.3) involves the Jacobian matrix $\mathbf{J} \in \mathbb{R}^{IJK \times (I+J+K)R}$. The nth row of $\mathbf{J}$ holds all first order partial derivatives of $f_n(\mathbf{z})$. Using the permutation matrices $\mathbf{\Pi}^{(i)}$, as defined in Sec. 7.1.1, $\mathbf{J}$ is written as $[\mathbf{J}_\mathbf{A}; \mathbf{J}_\mathbf{B}; \mathbf{J}_\mathbf{C}]$,

with

$$\mathbf{J}_{\mathbf{A}} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I,$$

$$\mathbf{J}_{\mathbf{B}} = \mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J \right),$$

$$\mathbf{J}_{\mathbf{C}} = \mathbf{\Pi}^{(3)^{\mathrm{T}}} \left((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K \right).$$

The second term in (7.3) consists of two factors. The factor $\frac{\mathrm{d}f_n}{\mathrm{d}m_n}$, i.e., the change in cost function, becomes very small as we move closer to the (global) optimum. The other factor, $\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right)$, is sparse for the CPD, because each tensor entry only depends on a few model entries [183]. The generalized GGN method, which approximates the Hessian by only its first term

$$\hat{\mathbf{H}} = \sum_{n=1}^{IJK} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \mathbf{J}_{n:},$$

can thus be expected to perform well for fitting a CPD [169, 221, 223]. The tensor $\mathcal{D} \in \mathbb{R}^{I \times J \times K}$, with

$$d_n = \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2},$$

can be constructed to write $\hat{\mathbf{H}}$ as follows:

$$\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}} \mathrm{Diag}\left(\mathrm{vec}(\mathcal{D})\right) \mathbf{J}.$$

Finally, by defining

$$\mathbf{Z} = \mathrm{Diag}\left(\mathrm{vec}(\mathcal{D})\right) \in \mathbb{R}^{IJK \times IJK},$$

the GGN Hessian approximation can be written more compactly as:

$$\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}} \mathbf{Z} \mathbf{J}.$$

For the LS cost function, $\mathbf{Z} = \mathbf{I}_{IJK}$ and the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ of the GN method is recovered. If the f_n are nonconvex, negative values can occur in $\mathbf{Z}$, especially far from the optimum. This can possibly introduce negative eigenvalues in $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ as well. In such cases, one should ensure that a descent direction is chosen and continued convergence of the optimization algorithm is guaranteed; see [214, 223].

Computing the full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can become expensive as the tensor dimensions are increased, as $\mathbf{J}$ is of dimensions $IJK \times (I+J+K)R$ and $\mathbf{Z}$ holds IJK non-zero entries on its diagonal. Fortunately, its explicit computation

is only required if one wants to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ exactly. Similar to inexact GN, we use an inexact GGN method to reduce the cost: an iterative solver such as (preconditioned) CG [195, 223] or MINRES [153] is used to solve $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ approximately. CG is only applicable when $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is positive semi-definite (PSD). For convex f_n, the Hessian approximation is always PSD, but for nonconvex f_n, this is not guaranteed. In such cases, MINRES can be used instead. Both methods do not require the full matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, but only need to efficiently evaluate the product of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ with an arbitrary vector $\mathbf{x}$, where

$$\mathbf{x} = [\mathrm{vec}(\mathbf{X_A}); \mathrm{vec}(\mathbf{X_B}); \mathrm{vec}(\mathbf{X_C})] = [\mathbf{x_A}; \mathbf{x_B}; \mathbf{x_C}]$$

and $\mathbf{X_A}$, $\mathbf{X_B}$ and $\mathbf{X_C}$ have the same dimensions as $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively. As the computational complexity of CG/MINRES is dominated by the complexity of the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-products, the $\mathcal{O}(I^3R^3)$ flop cost of solving the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ exactly is replaced by a cost of $\mathcal{O}(kl)$ flop. Herein, k is the number of iterations of the inexact solver (50 to 100 typically) and l is its per-iteration cost, i.e., the cost of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$. Below, we provide three strategies to reduce both k and l. Together, these strategies make the inexact GGN method applicable to much larger tensors than when a naive implementation is used. In Sec. 7.2.2, we explain how the structure of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can be exploited to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ efficiently using the CG/MINRES algorithm. Next, we approximate the tensor $\mathcal{D}$ with an LMLRA in Sec. 7.2.3 to further reduce the cost of each CG/MINRES iteration without sacrificing the structure of $\mathbf{J}$. Finally, in Sec. 7.2.4, we provide dedicated preconditioners for the computation of a CPD with general cost functions. These preconditioners reduce the number of CG/MINRES iterations that are necessary to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ up to a certain accuracy and by leveraging the structure of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, they can be applied cheaply.

7.2.2 Computing $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ efficiently

In the product $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$, the vector $\mathbf{y}$ can be split into three parts of the same dimensions as $\mathbf{x_A}$, $\mathbf{x_B}$ and $\mathbf{x_C}$, respectively. The parts $\mathbf{y_A}$, $\mathbf{y_B}$ and $\mathbf{y_C}$ of the product $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ can be obtained separately [223]. For the vector

$$\begin{aligned}\mathbf{y_A} &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_A}\mathbf{x_A} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_B}\mathbf{x_B} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_C}\mathbf{x_C} \\ &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\left(\mathbf{J_A}\mathbf{x_A} + \mathbf{J_B}\mathbf{x_B} + \mathbf{J_B}\mathbf{x_C}\right), \end{aligned} \tag{7.4}$$

the terms of the last factor can be rewritten as follows:

$$\begin{aligned}
\mathbf{J}_{\mathbf{A}}\mathbf{x}_{\mathbf{A}} &= \left[(\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right]\mathbf{x}_{\mathbf{A}} \\
&= \operatorname{vec}\left(\mathbf{X}_{\mathbf{A}}\left(\mathbf{C}\odot\mathbf{B}\right)^{\mathrm{T}}\right), \\
\mathbf{J}_{\mathbf{B}}\mathbf{x}_{\mathbf{B}} &= \boldsymbol{\Pi}^{(2)^{\mathrm{T}}}\left[(\mathbf{C}\odot\mathbf{A})\otimes\mathbf{I}_J\right]\mathbf{x}_{\mathbf{B}} \\
&= \boldsymbol{\Pi}^{(2)^{\mathrm{T}}}\operatorname{vec}\left(\mathbf{X}_{\mathbf{B}}\left(\mathbf{C}\odot\mathbf{A}\right)^{\mathrm{T}}\right), \\
\mathbf{J}_{\mathbf{C}}\mathbf{x}_{\mathbf{C}} &= \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}\left[(\mathbf{B}\odot\mathbf{A})\otimes\mathbf{I}_K\right]\mathbf{x}_{\mathbf{C}} \\
&= \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}\operatorname{vec}\left(\mathbf{X}_{\mathbf{C}}\left(\mathbf{B}\odot\mathbf{A}\right)^{\mathrm{T}}\right).
\end{aligned}$$

Herein, the identity

$$(\mathbf{U}\otimes\mathbf{V})\operatorname{vec}(\mathbf{W}) = \operatorname{vec}\left(\mathbf{V}\mathbf{W}\mathbf{U}^{\mathrm{T}}\right) \tag{7.5}$$

is used. By comparing with (7.1), it can be seen that each of the three terms can be seen as an unfolding of a CPD. The factor $(\mathbf{J}_{\mathbf{A}}\mathbf{x}_{\mathbf{A}} + \mathbf{J}_{\mathbf{B}}\mathbf{x}_{\mathbf{B}} + \mathbf{J}_{\mathbf{B}}\mathbf{x}_{\mathbf{C}})$ from (7.4) thus holds the sum of three unfolded CPDs. They are merged into one CPD $\mathcal{E} = \left[\!\left[\mathbf{E}^{\mathbf{A}}, \mathbf{E}^{\mathbf{B}}, \mathbf{E}^{\mathbf{C}}\right]\!\right]$ by defining

$$\begin{aligned}
\mathbf{E}^{\mathbf{A}} &= \begin{bmatrix}\mathbf{X}_{\mathbf{A}} & \mathbf{A} & \mathbf{A}\end{bmatrix} \in \mathbb{R}^{I\times 3R}, \\
\mathbf{E}^{\mathbf{B}} &= \begin{bmatrix}\mathbf{B} & \mathbf{X}_{\mathbf{B}} & \mathbf{B}\end{bmatrix} \in \mathbb{R}^{J\times 3R}, \\
\mathbf{E}^{\mathbf{C}} &= \begin{bmatrix}\mathbf{C} & \mathbf{C} & \mathbf{X}_{\mathbf{C}}\end{bmatrix} \in \mathbb{R}^{K\times 3R}.
\end{aligned}$$

Exploiting the diagonal structure of $\mathbf{Z}$ then leads to the following expression for $\mathbf{y}_{\mathbf{A}}$:

$$\mathbf{y}_{\mathbf{A}} = \left[(\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right]^{\mathrm{T}}\left[\operatorname{vec}\left(\mathcal{D}\right) * \operatorname{vec}\left(\mathcal{E}\right)\right].$$

Writing $\mathcal{Q} = \mathcal{D} * \mathcal{E}$ and applying (7.5) one more time, yields

$$\mathbf{y}_{\mathbf{A}} = \operatorname{vec}\left(\mathbf{Q}_{(1)}(\mathbf{C}\odot\mathbf{B})\right). \tag{7.6}$$

Analogously, $\mathbf{y}_{\mathbf{B}}$ and $\mathbf{y}_{\mathbf{C}}$ can be found as

$$\mathbf{y}_{\mathbf{B}} = \operatorname{vec}\left(\mathbf{Q}_{(2)}(\mathbf{C}\odot\mathbf{A})\right), \text{ and } \mathbf{y}_{\mathbf{C}} = \operatorname{vec}\left(\mathbf{Q}_{(3)}(\mathbf{B}\odot\mathbf{A})\right).$$

The complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is dominated by the generation of the tensor $\mathcal{E}$ and the computation of the three `MTKRPROD`s. Assuming $I = J = K$ for simplicity, these each require $\mathcal{O}(I^3R)$ flop. In the following section, we discuss a strategy to reduce this complexity.

7.2.3 Approximating $\mathcal{D}$

Within a single GGN iteration, the new step direction $\mathbf{p}$ is obtained from the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ by repeatedly evaluating products of the form $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ using CG/MINRES. Only the vector $\mathbf{x}$ varies between the different CG/MINRES evaluations. Consequently, only the tensor $\mathcal{E}$ changes when $\mathbf{y}$ is computed as in (7.6), with $\mathcal{Q} = \mathcal{D} * \mathcal{E}$. One of the most costly steps in the computation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is the construction of the tensor $\mathcal{Q}$, as it requires a Hadamard product of $\mathcal{D}$ and $\mathcal{E}$. As the same $\mathcal{D}$ is used throughout the execution of the full CG or MINRES algorithm, it is often worthwhile to spend some time computing an LMLRA $\hat{\mathcal{D}}$ of $\mathcal{D}$. The information in $\mathcal{D}$ can be maintained up to a chosen precision, while the structure of its decomposition can be leveraged to efficiently obtain an approximation $\hat{\mathcal{Q}}$ of $\mathcal{Q}$. The second costly step, the `MTKRPROD`s of the form

$$\mathbf{y}_{\mathbf{A}} = \operatorname{vec}\left(\hat{\mathbf{Q}}_{(1)}(\mathbf{C} \odot \mathbf{B})\right),$$

can also be computed much more efficiently by exploiting the structure of $\hat{\mathcal{Q}}$, as explained in the remainder of this section.

Let $\hat{\mathcal{D}} = [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ be an LMLRA of $\mathcal{D}$, with $\mathcal{S} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$. Then, the Hadamard product $\hat{\mathcal{Q}} = \hat{\mathcal{D}} * [\![\mathbf{E}^{\mathbf{A}}, \mathbf{E}^{\mathbf{B}}, \mathbf{E}^{\mathbf{C}}]\!]$ is structured as well:

$$\begin{aligned}
\hat{\mathcal{Q}} &= [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{E}^{\mathbf{A}}, \mathbf{E}^{\mathbf{B}}, \mathbf{E}^{\mathbf{C}}]\!] \\
&= \sum_{s=1}^{3R} [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{e}^{\mathbf{A}}_{:s}, \mathbf{e}^{\mathbf{B}}_{:s}, \mathbf{e}^{\mathbf{C}}_{:s}]\!] \\
&= \sum_{s=1}^{3R} [\![\mathcal{S}; \mathbf{E}^{\mathbf{A}}_s \mathbf{U}, \mathbf{E}^{\mathbf{B}}_s \mathbf{V}, \mathbf{E}^{\mathbf{C}}_s \mathbf{W}]\!], \qquad (7.7)
\end{aligned}$$

wherein $\mathbf{E}^{\mathbf{A}}_s = \operatorname{Diag}\left(\mathbf{e}^{\mathbf{A}}_{:s}\right)$, $\mathbf{E}^{\mathbf{B}}_s = \operatorname{Diag}\left(\mathbf{e}^{\mathbf{B}}_{:s}\right)$ and $\mathbf{E}^{\mathbf{C}}_s = \operatorname{Diag}\left(\mathbf{e}^{\mathbf{C}}_{:s}\right)$. Because of the structure of $\mathbf{E}^{\mathbf{A}}$, some matrices $\mathbf{E}^{\mathbf{A}}_s\mathbf{U}$ are duplicates. Naturally, these have to be computed only once. The same holds for $\mathbf{E}^{\mathbf{B}}_s\mathbf{V}$ and $\mathbf{E}^{\mathbf{C}}_s\mathbf{W}$. Using (7.7), $\mathbf{Y}_{\mathbf{A}} = \operatorname{unvec}(\mathbf{y}_{\mathbf{A}}) = \operatorname{unvec}\left(\hat{\mathbf{Q}}_{(1)}(\mathbf{C} \odot \mathbf{B})\right)$ can be efficiently computed as

$$\begin{aligned}
\mathbf{Y}_{\mathbf{A}} &= \sum_{s=1}^{3R} (\mathbf{E}^{\mathbf{A}}_s \mathbf{U}) \mathbf{S}_{(1)} \left((\mathbf{E}^{\mathbf{C}}_s \mathbf{W}) \otimes (\mathbf{E}^{\mathbf{B}}_s \mathbf{V})\right)^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B}) \\
&= \sum_{s=1}^{3R} (\mathbf{E}^{\mathbf{A}}_s \mathbf{U}) \mathbf{S}_{(1)} \left(\left[(\mathbf{E}^{\mathbf{C}}_s \mathbf{W})^{\mathrm{T}} \mathbf{C}\right] \odot \left[(\mathbf{E}^{\mathbf{B}}_s \mathbf{V})^{\mathrm{T}} \mathbf{B}\right]\right), \qquad (7.8)
\end{aligned}$$

with $\mathbf{E}_s^{\mathbf{A}}\mathbf{U} \in \mathbb{R}^{I\times R_1}$, $(\mathbf{E}_s^{\mathbf{C}}\mathbf{W})^{\mathrm{T}}\mathbf{C} \in \mathbb{R}^{R_3\times R}$ and $(\mathbf{E}_s^{\mathbf{B}}\mathbf{V})^{\mathrm{T}}\mathbf{B} \in \mathbb{R}^{R_2\times R}$. The vector $\mathbf{y}_{\mathbf{A}}$ is obtained by vectorization of $\mathbf{Y}_{\mathbf{A}}$. Compared to (7.6), (7.8) is again a `MTKRPROD`, but now involving smaller tensors and matrices. The same strategy can be applied in the other two modes to obtain $\mathbf{y}_{\mathbf{B}} = \hat{\mathbf{Q}}_{(2)}(\mathbf{C}\odot\mathbf{A})$ and $\mathbf{y}_{\mathbf{C}} = \hat{\mathbf{Q}}_{(3)}(\mathbf{B}\odot\mathbf{A})$, wherein terms of the form $\mathbf{E}_s^{\mathbf{A}}\mathbf{U}$ and $\left(\mathbf{E}_s^{\mathbf{B}}\mathbf{V}\right)^{\mathrm{T}}\mathbf{B}$ can be reused from the computation of $\mathbf{y}_{\mathbf{A}}$.

Using specialized methods, e.g., a sequentially truncated multilinear singular value decompostion (ST-MLSVD) algorithm with randomized SVD computation [93, 224], a multilinear rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of an $I \times J \times K$-tensor $\mathcal{D}$ can be computed efficiently, with the approximation having good numerical properties due to the orthogonality constraints. If forming or storing the full tensor $\mathcal{D}$ is too expensive, one can sample entries or fibers from it and compute an LMLRA of this incomplete tensor instead [23, 192, 227, 234].

With a multilinear rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of $\mathcal{D}$, the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is dominated by the products of the form $\mathbf{M} = \mathbf{S}_{(1)}\left(\left[(\mathbf{E}_r^{\mathbf{C}}\mathbf{W})^{\mathrm{T}}\mathbf{C}\right]\odot\left[(\mathbf{E}_r^{\mathbf{B}}\mathbf{V})^{\mathrm{T}}\mathbf{B}\right]\right)$ and $\mathbf{y}_{\mathbf{A}} = (\mathbf{E}_r^{\mathbf{A}}\mathbf{U})\mathbf{M}$ from (7.8). Again assuming $I = J = K$, these have a complexity of $\mathcal{O}(RR_{\mathrm{ML}}^3))$ flop and $\mathcal{O}(RR_{\mathrm{ML}}I)$ flop, respectively. This leads to a total complexity of $\mathcal{O}(R^2R_{\mathrm{ML}}\max(I, R_{\mathrm{ML}}^2))$ flop for the entire $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-evaluation. Compared to the complexity of $\mathcal{O}(I^3R)$ flop when using the full $\mathcal{D}$-tensor, we see that the approximation decreases the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ whenever $RR_{\mathrm{ML}} < I^2$ and $RR_{\mathrm{ML}}^3 < I^3$. Taking into account that in applications typically $I \gg R$, while by definition $I \geq R_{\mathrm{ML}}$, these conditions are easily satisfied for realistic values of R_{ML}, R and I. Of course, one should also take the computation time for the LMLRA itself into account. The LMLRA is thus only useful if enough $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-evaluations are required in the CG or MINRES computations. In addition, one should be careful not to take R_{ML} so small that it yields a poor approximation, which can lead to a bad step in the optimization procedure. This can then increase the number of GGN iterations, negating the computational gain that replacing $\mathcal{D}$ by its LMLRA yields. In Sec. 7.4.2, we discuss strategies for picking an adequate value of R_{ML}.

7.2.4 Preconditioners

The convergence rate of an iterative solver for the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ is generally higher when the eigenvalues of the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ are well clustered. To improve the spectral properties of this matrix, a suitable preconditioner [207] can be used. A good preconditioner is cheap to apply and clusters the eigenvalues of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ around one, which should reduce the required number of CG or MINRES

iterations. Using the preconditioner $\mathbf{M}$, the modified system $\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{M}^{-1}\mathbf{g}$ is solved instead of the original system.

In iterative methods, the preconditioner $\mathbf{M} \in \mathbb{R}^{(I+J+K)R\times(I+J+K)R}$ is applied implicitly in every iteration. Instead of the multiplication of $\mathbf{M}^{-1}$ with the full $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix, only systems of the form $\mathbf{M}\mathbf{s} = \mathbf{t}$ need to be solved for $\mathbf{s} \in \mathbb{R}^{(I+J+K)R}$, with $\mathbf{t} \in \mathbb{R}^{(I+J+K)R}$ an arbitrary vector. A good preconditioner clusters the eigenvalues of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, which results in a reduction of the number of CG/MINRES iterations that are required to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ [150]. If the preconditioner is sufficiently cheap to apply, this also reduces the required time to solve the system. In order obtain a preconditioner $\mathbf{M}$ that can be applied efficiently, $\mathbf{M}$ is typically constructed to be sparse and/or structured. This is also the case for the block-Jacobi and Jacobi preconditioners that we propose below. Both preconditioners are sparse and have exploitable higher-order structure which makes them much cheaper to apply than generic preconditioners. When showing complexities, we assume $I = J = K$, as before. In Sec. 7.4.1, the efficiency and effectiveness of both preconditioners is compared.

Block-Jacobi

A block-Jacobi preconditioner $\mathbf{M}$ for the optimization problem is a block-diagonal matrix with $\mathbf{G}^{(n,n)} = \mathbf{J}_n^{\mathrm{T}}\mathbf{Z}\mathbf{J}_n$ as its blocks, where $\mathbf{G}^{(1,1)} \in \mathbb{R}^{IR\times IR}$, $\mathbf{G}^{(2,2)} \in \mathbb{R}^{JR\times JR}$ and $\mathbf{G}^{(3,3)} \in \mathbb{R}^{KR\times KR}$. To solve $\mathbf{M}\mathbf{s} = \mathbf{t}$, one thus needs an efficient method to solve $\mathbf{G}^{(n,n)}\mathbf{s}^{(n)} = \mathbf{t}^{(n)}$. The method from Chapter 6 can be applied to obtain the highly structured diagonal blocks $\mathbf{G}^{(n,n)}$. Matrices $\mathbf{G}_{\mathrm{nz}}^{(n,n)}$ holding only the nonzero entries of the $\mathbf{G}^{(n,n)}$-matrices are efficiently computed as

$$\mathbf{G}_{\mathrm{nz}}^{(1,1)} = \mathbf{D}_{(1)}\left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B})\right),$$

$$\mathbf{G}_{\mathrm{nz}}^{(2,2)} = \mathbf{D}_{(2)}\left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A})\right),$$

$$\mathbf{G}_{\mathrm{nz}}^{(3,3)} = \mathbf{D}_{(3)}\left((\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A})\right),$$

and the zeros are added afterwards. The construction of one diagonal block requires $\mathcal{O}(I^3R^2)$ flop. If a multilinear rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of $\mathcal{D}$ is available, its structure can be exploited to reduce the computational load of the construction of the blocks to $\mathcal{O}(\max(I, R_{\mathrm{ML}}^2)R_{\mathrm{ML}}R^2)$ flop. After forming the diagonal blocks, applying the preconditioner, i.e., solving $\mathbf{M}\mathbf{s} = \mathbf{t}$, can be done efficiently by exploiting the particular structured sparsity of the blocks. This is done by first computing an LDL factorization [87] of $\mathbf{M}$, wherein the $\mathbf{L}$-matrix is lower triangular and the $\mathbf{D}$-matrix is diagonal, and then performing back substitution. For the proposed block-Jacobi preconditioner,

an LDL factorization of $\mathbf{M}$ can be obtained by straightforwardly combining the LDL factorizations of the different $\mathbf{G}^{(n,n)}$-blocks, so we can focus on each block independently. The LDL factorization of a $\mathbf{G}^{(n,n)}$-block can be computed through I independent LDL factorizations of $R \times R$-matrices. These I independent LDL factorizations can then be combined into an LDL factorization of the entire $\mathbf{G}^{(n,n)}$-block. As an illustration, in Fig. 7.2, the nonzero stucture of a $\mathbf{G}^{(n,n)}$-block is shown for $I = 5$ and $R = 3$ and the entries handled by the second LDL factorization are highlighted. This dedicated approach for the CPD problem requires only $\mathcal{O}(IR^3)$ operations per diagonal block, which is much cheaper than sparse LDL factorization methods that ignore the structure and require at least $\mathcal{O}(I^2R^4)$ operations per block [41]. Additionally, there are no issues with unwanted fill-in—entries that are zero and should remain zero obtain a nonzero value due to unrelated operations and need to be restored—from which sparse LDL algorithms suffer. The back substitution then involves only R iterations per block, working on length-I vectors. The LDL factorization has to be computed only once and afterwards, the computed decomposition of $\mathbf{M}$ can be reused in every CG/MINRES iteration to solve $\mathbf{G}^{(n,n)}\mathbf{s}^{(n)} = \mathbf{t}^{(n)}$ via back substitution in $\mathcal{O}(IR^2)$ flop per block. In contrast, the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ product itself already requires $\mathcal{O}(I^3R)$ flop.

The block-Jacobi preconditioner reduces the computational cost if the lower number of CG or MINRES iterations compensates for the overhead of computing and decomposing the $\mathbf{G}^{(n,n)}$-matrices. In Sec. 7.4.1, we show that, for typical values of I, R and R_{ML}, a significant reduction is achieved.

Jacobi

For a Jacobi preconditioner, $\mathbf{M}$ is the diagonal of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$. To compute $\mathbf{M}$, the inner product of each row of $\mathbf{J}$ with itself has to be computed, whereby each entry has to be weighted with its corresponding entry of $\mathcal{D}$. This leads to the following expression for the diagonal $\mathbf{m}_{\mathrm{diag}}$ of $\mathbf{M}$:

$$\mathbf{m}_{\mathrm{diag}} = \begin{bmatrix} \mathrm{vec}\left(\mathbf{D}_{(1)}\left(\mathbf{C}^{\bullet 2} \odot \mathbf{B}^{\bullet 2}\right)\right) \\ \mathrm{vec}\left(\mathbf{D}_{(2)}\left(\mathbf{C}^{\bullet 2} \odot \mathbf{A}^{\bullet 2}\right)\right) \\ \mathrm{vec}\left(\mathbf{D}_{(3)}\left(\mathbf{B}^{\bullet 2} \odot \mathbf{A}^{\bullet 2}\right)\right) \end{bmatrix}.$$

The system $\mathbf{M}\mathbf{s} = \mathbf{t}$ can then be solved by computing $\mathbf{s} = \frac{\mathbf{t}}{\mathbf{m}_{\mathrm{diag}}}$. Forming $\mathbf{m}_{\mathrm{diag}}$ requires $\mathcal{O}(I^3R)$ flop. As before, the same $\mathbf{m}_{\mathrm{diag}}$ can be used for all evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$. Only $\mathbf{x} = \frac{\mathbf{y}}{\mathbf{m}_{\mathrm{diag}}}$ (costing $\mathcal{O}(IR)$ flop) has to be computed in every evaluation, so the preconditioner needs to save only one $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluation to reduce the computation time. If a multilinear rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of $\mathcal{D}$ is available, this reduces the complexity of

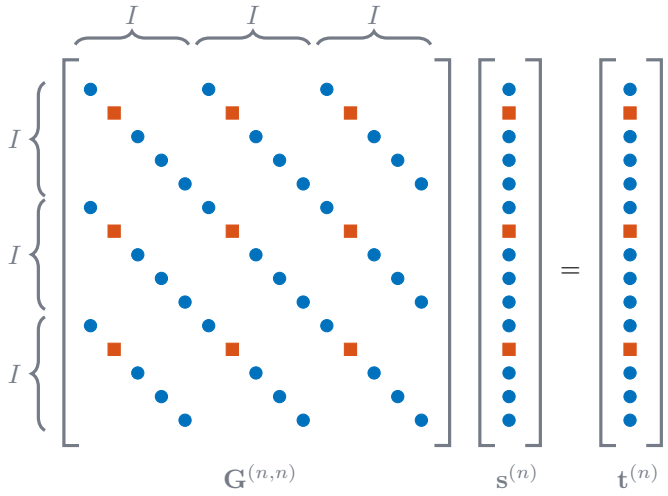

Figure 7.2: Thanks to the structure of each $\mathbf{G}^{(n,n)}$-block, applying the block-Jacobi preconditioner involves solving I independent linear systems. To achieve an even lower computational cost, I independent LDL factorizations are computed such that only I back and forward substitution steps are required to apply the preconditioner. Entries corresponding to the second factorization are highlighted with red squares in the spy plot for $I = 5$ and $R = 3$.

computing $\mathbf{m}_{\text{diag}}$ to $\mathcal{O}(R_{\text{ML}} R \max(I, R_{\text{ML}}^2))$ flop, making the preconditioner even more efficient.

7.3 Alternative cost functions

As mentioned in Sec. 7.2, many alternatives exist for the standard LS cost function. Depending on the application and the distribution of the noise, they will give better or worse results. β-divergences are a popular choice of alternative cost function. Their behavior depends on the value of β: for small values of β, smaller values are given more importance compared to the LS cost function, while the converse is true for large values of β. Additionally, they interpolate continuously between three popular cost functions: the Itakura–Saito divergence ($\beta = 0$), the Kullback–Leibler divergence ($\beta = 1$) and the LS distance ($\beta = 2$). The β-divergence is discussed extensively in the context of CPD fitting in Chapter 6.

In this section, we consider another non-LS cost function for the CPD. It is particularly useful when the tensor entries follow a Rician distribution. In such cases, modeling with the Gaussian distribution introduces a bias [56], especially when the variance of the noise is large. Consider a complex measurement $z = N(\mu_1, \sigma^2) + jN(\mu_2, \sigma^2)$, with $N(\mu, \sigma^2)$ the Gaussian (or normal) distribution with mean μ and standard deviation σ. Herein, $\mu_1 + j\mu_2$ is the underlying unperturbed measurement and σ is the standard deviation of the zero-mean additive Gaussian noise on both the real and imaginary parts. Consequently $A = \sqrt{\mu_1^2 + \mu_2^2}$ corresponds to the unperturbed magnitude of the measurement. The distribution of its modulus $x = |z|$ then has a Rician probability density function:

$$p(x|A, \sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2 + A^2}{2\sigma^2}\right) I_0\left(\frac{Ax}{\sigma^2}\right),$$

with I_k the modified Bessel function of the first kind with order k. For $A = 0$, i.e., when the unperturbed measurement is zero, the Rician distribution is equal to the Rayleigh distribution. The optimum of the LS cost function does not coincide with the MLE of Rician distributed data. Instead, the MLE for a variable x following a Rician distribution with noise variance σ^2, is the solution of

$$\min_y \frac{y^2}{2\sigma^2} - \log\left(I_0\left(\frac{yx}{\sigma^2}\right)\right).$$

Some methods to find the MLE and alternative estimators are given in [126, 178, 201]. The value of σ^2 can typically be obtained by averaging the squared magnitude over a region where the unperturbed measurement is zero [177]. For a tensor $\mathcal{T}$ with Rician distributed data and its CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, one thus obtains the entry-wise cost function

$$f_n = \sum_{n=1}^{IJK} \frac{m_n^2}{2\sigma^2} - \log\left(I_0\left(\frac{m_n t_n}{\sigma^2}\right)\right). \tag{7.9}$$

As the t_n are moduli, we restrict $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative. For this cost function, the tensors $\mathcal{R}$ and $\mathcal{D}$, which are defined in Sec. 7.2.1 and used in the computation of the gradient and the Hessian approximation, respectively, satisfy:

$$\mathcal{R} = \frac{\mathcal{M}}{\sigma^2} - \frac{\mathcal{T}}{\sigma^2} * \frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})},$$

and

$$\mathcal{D} = \frac{1}{\sigma^2} - \left(\frac{\mathcal{T}^{\bullet 2}}{\sigma^4} - \frac{\mathcal{T}}{\sigma^2 \mathcal{M}} * \frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})} - \frac{\mathcal{T}^{\bullet 2}}{\sigma^4} * \left(\frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}\right)^{\bullet 2}\right).$$

The f_n are not necessarily convex, so $\mathcal{D}$ may have negative values, especially when $\frac{t_n}{m_n}$ is large. As explained in Sec. 7.2.1, this can lead to an indefinite Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ which impedes fast convergence of the algorithm. Fortunately, as for β-divergence cost functions, the LS solution often offers a decent initialization of the algorithm, which ideally implies the nonnegativity of $\mathcal{D}$.

Rician distributed data occurs for instance in medical applications where analysis focuses on the magnitude of a complex signal, such as functional magnetic resonance imaging (fMRI) [56]. The real and imaginary parts of the complex-valued Fourier spectra can often both be modeled as Gaussians with different means but the same variance. To avoid problems due to incidental phase variations, magnitude images are predominantly used in fMRI analysis instead of the complex-valued Fourier spectra themselves [56]. As the magnitude of each measurement is not Gaussian distributed, but Rician, using the LS cost function introduces a bias which can influence the interpretability of the results. For such fMRI images, the Rician-dedicated cost function is more suitable.

7.4 Experiments

Experiments are performed on synthetic and simulated real-life data. First, we compare the performance of different preconditioners on the computation time and number of iterations to convergence that are required. Second, the influence on the computational cost of using an approximation of the tensor $\mathcal{D}$ is studied. Next, we show how the method performs compared to the small-scale version and standard first-order methods from the literature. We also apply our proposed techniques in the RBS framework to decompose a large tensor. Finally, we use our method to separate simulated fMRI sources. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0 [228]. For experiments on synthetic data, entries of factor matrices are sampled independently from the uniform distribution on the interval (a, b), denoted by $U(a, b)$. Writing $\bar{\mathcal{T}}$ as the perturbed version of the tensor $\mathcal{T}$, we can define the perturbation tensor itself as $\mathcal{N} = \bar{\mathcal{T}} - \mathcal{T}$. The signal to noise ratio (SNR) is then $\mathbb{E}\left(20\log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)\right)$. When the entries of a tensor $\mathcal{T}$ are perturbed by multiplicative Gamma noise, we have $\bar{\mathcal{T}} = \mathcal{T} * \mathcal{N}_\gamma$. The entries of the noise tensor $\mathcal{N}_\gamma$ are sampled from a Gamma distribution with shape α and scale $\frac{1}{\alpha}$, hence, the SNR (in dB) is expected to be $10\log_{10}(\alpha)$.

Table 7.1: Preconditioners can reduce the number of CG or MINRES steps in the GGN method. With the block-Jacobi preconditioner, fewer steps are required than with the Jacobi preconditioner, but the latter can be computed much more efficiently, leading to overall faster iterations for smaller tensors. A β-divergence cost function with $\beta = 1.5$ is used to decompose the $I \times I \times I$-tensors and all noise is multiplicative Gamma noise. The preconditioning time is included in the GGN iteration time. Results are medians over 20 trials.

	Preconditioner		
Problem	None	Jacobi	Block-J.
$I = 50$, $R = 5$, no noise			
Median number of CG steps	110	73.5	44
Time per GGN iteration (s)	0.4427	0.3346	0.2526
Prec. time per GGN iteration (s)	0	0.0053	0.0321
$I = 50$, $R = 5$, 20 dB noise			
Median number of CG steps	88	65	41
Time per GGN iteration (s)	0.4461	0.3274	0.2446
Prec. time per GGN iteration (s)	0	0.0053	0.0321
$I = 30$, $R = 3$, no noise			
Median number of CG steps	75	33	23
Time per GGN iteration (s)	0.1813	0.0772	0.1069
Prec. time per GGN iteration (s)	0	0.0025	0.0133
$I = 30$, $R = 3$, 20 dB noise			
Median number of CG steps	49	33	21
Time per GGN iteration (s)	0.1218	0.0743	0.0765
Prec. time per GGN iteration (s)	0	0.0025	0.0133

7.4.1 Preconditioning

We have proposed two different preconditioners for the GGN method: a diagonal Jacobi preconditioner and a block-diagonal block-Jacobi preconditioner. In Table 7.1, the median number of CG iterations required to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ up to a tolerance of 10^{-6} is compared for the GGN method with and without preconditioning. Preconditioning clearly decreases the number of CG iterations that are required to find the next iterate of the GGN method. With the block-Jacobi preconditioner, even fewer CG steps are needed than with the Jacobi preconditioner. However, the higher computation time of the former makes it less attractive than the latter for smaller tensors. For all problems in Table 7.1, both preconditioners are effective in decreasing the computation time.

7.4.2 Approximating $\mathcal{D}$

Using non-LS cost functions adds complexity to the CPD algorithm, which naturally results in a larger computational cost of the method. In Fig. 7.3, the computation time of 100 $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations is shown in function of the tensor size. In our experiments, around 100 $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations are generally required when using the CG or MINRES algorithms to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$, although with preconditioning, around 50 can often be achieved. From Fig. 7.3, we can see that the computation of the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ becomes very costly for the larger tensors. For such tensors, the approach of Sec. 7.2.3 can be used to approximate the tensor $\mathcal{D}$ with an LMLRA. This decomposition is then used instead of the full $\mathcal{D}$ in the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations of the CG or MINRES algorithm. Exploiting the LMLRA structure speeds up these evaluations significantly. In Fig. 7.3, one can also see the computation time of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ with compression of $\mathcal{D}$. For small tensors, using a compressed $\mathcal{D}$ actually increases the computation time, because of the higher number of `MTKRPROD`s that are needed compared to the uncompressed variant ($9R$ compared to 3). This makes the compressed format more expensive to compute than simply forming all entries of $\mathcal{Q} = \mathcal{D} * \mathcal{E}$. However, the computation time for $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ grows exponentially with increasing tensor dimensions when $\mathcal{D}$ is not compressed. In contrast, the computation time stays about the same when $\mathcal{D}$ is compressed (and R_{ML} is kept constant), leading to considerable efficiency gains as the tensor dimensions are increased. Computing the compression takes time by itself, so it is mostly useful when many $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ products have to be computed with the same $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix, as is typically the case.

To assess the effect of the compression on the convergence speed, results for different compression factors are compared in Table 7.2 for rank-4 $I \times I \times I$ tensors with $I = 50$, $I = 100$ and $I = 200$, perturbed by multiplicative Gamma noise with an SNR of 40 dB. A β-divergence cost function is used with $\beta = 1.5$ and the factor matrices are initialized randomly with entries sampled from $U(0, 1)$. The mean number of GGN iterations and computation time until convergence are shown when $\mathcal{D}$ is approximated by an LMLRA with core dimensions $S \times S \times S$. The algorithm stops when either a relative step size $\frac{\|\mathbf{p}\|}{\|\mathbf{z}\|}$ or a relative cost function change compared to the initial cost function value below 10^{-15} is encountered. Only successful runs are taken into account, where the algorithm converged to a solution with a function value $f < 10^{-4}I^3$ within 500 GGN iterations. The compression factors I/S that are considered are 1, 5, 10 and 25. The tolerance of the CG/MINRES algorithm is 10^{-8} and the Jacobi preconditioner is used. First, it can be seen that for $I = 100$ and $I = 200$, the computation time is significantly reduced by compressing $\mathcal{D}$. For a compression factor of 5, the number of iterations until convergence is

actually lower than when the full $\mathcal{D}$-tensor is used, which evidently reduces the computation time as well. For the large compression factors 10 and 25, the lower quality of the approximation sometimes leads to an increase of the number of GGN iterations, but the compression still offers a time gain compared to using the full $\mathcal{D}$-tensor. For smaller tensors with $I = 50$, the approximation of $\mathcal{D}$ increases the computation time and number of iterations, so no computational benefit is achieved.

To ensure that the convergence speed is not hindered by an inadequate approximation rank, an adaptive compression factor can be used. By trimming only multilinear singular values under a certain threshold, a sufficient accuracy of the approximation can be guaranteed. As an example, we perform the experiment from Table 7.2 for $I = 100$ with an adaptive compression factor. A compression factor of 25 is chosen and, if necessary, the multilinear rank is gradually increased by adding multilinear singular vectors in each mode until the sum of the squared multilinear singular values of $\hat{\mathcal{D}}$ sums to at least 99.9% of $\|\mathcal{D}\|_{\mathrm{F}}^2$. To have access to the extra multilinear singular vectors, an LMLRA of slightly higher multilinear rank is computed and the rank $(I/25 \times I/25 \times I/25)$-LMLRA is extracted from it. This leads to a mean time of 10 s and a mean number of GGN iterations of 43, while all 20 runs converge to the optimal solution (given the noise level). Strategically increasing the multilinear rank of the compression can thus improve the convergence behavior compared to fixing the compression factor at 25. In this experiment, the adaptive method performs even better than the best method from Table 7.2, the one using a fixed compression factor of 5, without needing to specify the compression factor a priori.

In practice, the LS solution, which can be obtained with a cheaper method, is often a good initialization for the non-LS GGN method. Therefore, we can focus on the convergence behavior of the GGN method close to the solution. To analyze this behavior, the same experiment as above is performed, but now using the LS solution as initialization. This initialization is obtained by running the GN LS method until $\|\mathcal{M} - \mathcal{T}\|_{\mathrm{F}}^2$ drops below a value of $I^3 \cdot 10^{-4}$. In Table 7.3, the results can be seen for a β-divergence cost function with $\beta = 1.5$ and $\beta = 0.5$. Again, the number of GGN iterations and the computation time until convergence are shown when $\mathcal{D}$ is approximated by an LMLRA with core dimensions $S \times S \times S$. The mean number of CG (for $\beta = 1.5$) or MINRES (for $\beta = 0.5$) iterations used to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{g}$ is also shown. Comparing the column for compression factor 5 to the column for compression factor 1 (no compression), shows that compressing $\mathcal{D}$ can significantly reduce the runtime of the algorithm. For the largest tensors, the number of GGN iterations was only moderately affected by the compression and for $\beta = 1.5$ compression factors up to 25 were viable.

There is a limit on how much time can be gained by compressing $\mathcal{D}$. A first limiting factor is that the cost function and gradient evaluations already take a significant amount of time for β-divergences. Even if the time required for the $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ evaluations could be eliminated entirely, this cost would remain. Second, from the experiments in Table 7.3 with $\beta = 0.5$, for which the f_n are nonconvex, it can be seen that the approximation becomes progressively worse if the compression factor is increased. If the approximation becomes too bad, for example, when $\beta = 0.5$, $I = 100$ and $I/S = 10$, this increases the number of iterations that are needed to reach the solution which in turn increases the runtime of the algorithm. Also, while $\mathcal{D}$ has all entries equal to one when $\beta = 2$, the entries change proportionally with the distance $|\beta - 2|$. As a result, a higher multilinear rank and thus a larger core tensor is required to approximate this tensor well with an LMLRA when $|\beta - 2|$ is large, which can be seen by comparing the results for $\beta = 1.5$ and $\beta = 0.5$.

The system $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx} = -\mathbf{g}$ is altered when $\mathcal{D}$ is approximated. Even if this approximated system is solved exactly, its solution will be (slightly) different from the solution of the true system. The approximation could also influence the number of CG or MINRES iterations that are needed to obtain a solution with the requested accuracy. In this experiment, however, this number is largely unaffected by the compression.

7.4.3 Other β-divergence CPD algorithms

In this subsection, we compare the inexact GGN method to other methods to compute CPDs with general cost functions. The stopping criteria for the algorithms are again a relative step size below 10^{-15} or a relative cost function change below 10^{-15}. First, a comparison with the exact GGN method in function of the tensor dimensions can be seen in Fig. 7.4. Rank-10 tensors of dimensions $I \times I \times I$ are generated with factor matrix entries sampled from $U(0, 1)$ and the tensor $\mathcal{D}$ is computed. For the exact GGN method, the mean time (over 20 trials) to compute $\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$ is shown, alongside the mean time to compute $\hat{\mathbf{H}}^{\dagger}$. For the inexact GGN method, the mean time is displayed to compress the tensor $\mathcal{D}$ to a multilinear rank-$(10, 10, 10)$ tensor, next to the mean computation time of 100 products $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ using this compressed version of $\mathcal{D}$, which in this case is plentiful to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{ZJp} = -\mathbf{g}$ up to a precision of 10^{-6}. From Fig. 7.4, it can be seen that the time required for the construction of the GGN Hessian approximation $\hat{\mathbf{H}}$ grows at a manageable pace for increasing tensor dimensions. However, the computation time of the pseudoinverse $\hat{\mathbf{H}}^{\dagger}$ grows quickly when the tensor dimensions are increased, making the exact GGN method infeasible for larger tensors. Consequently, for

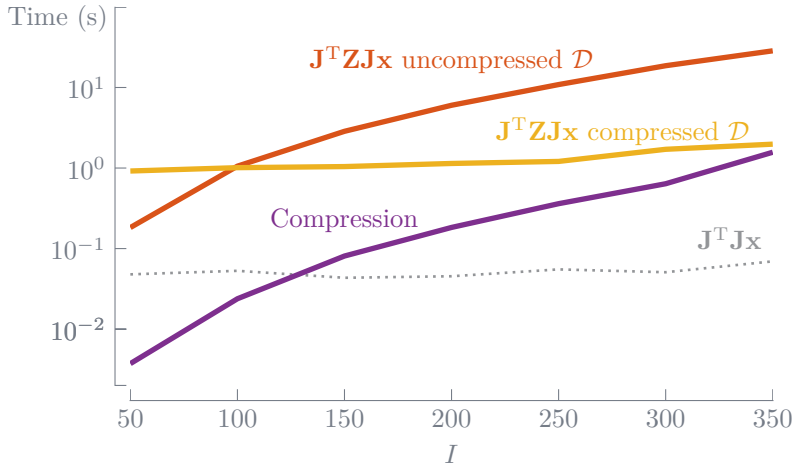

Figure 7.3: When $\mathcal{D}$ is compressed, the computation time for an evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ (without the compression time) remains relatively constant for increasing I, while it increases exponentially without compression. Mean time (over 20 trials) for 100 evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ with the same $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$-matrix (a typical value using CG) for the β-divergence cost function ($\beta = 1.5$) in function of the tensor size. The tensors have dimensions $I \times I \times I$ and rank 10. The CPD model also has rank 10. A multilinear rank-$(I/25, I/25, I/25)$ approximation was used for $\mathcal{D}$. Aside from the largest dimensions, the computation time of the LMLRA is negligible compared to that of the $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ products. The computation time for 100 evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{Jx}$ for the LS distance is added for reference.

such tensors, it is necessary to switch to the inexact GGN method, wherein $\mathbf{J}^{\mathrm{T}}\mathbf{ZJp} = -\mathbf{g}$ is solved using the CG/MINRES method.

The full Hessian approximation $\hat{\mathbf{H}}$ could be used to perform products of the form $\hat{\mathbf{H}}\mathbf{x}$ during the CG/MINRES iterations. The mean computation time of 100 such products is also shown in Fig. 7.4 for comparison. It can be seen that the computation time of $\hat{\mathbf{H}}\mathbf{x}$ increases steadily when the tensor dimensions are increased. In contrast, the computation time of the product $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ stays relatively constant when the Hessian approximation is not formed in full, using the approach from Sec. 7.2. Evidently, the one-time costs for the formation of $\hat{\mathbf{H}}$ and for the compression of $\mathcal{D}$ should also be taken into account, with the latter remaining comparatively lower as the tensor dimensions are increased. Thus, for larger tensors, the CG/MINRES iterations can be more efficiently performed by using the GGN Hessian approximation implicitly.

Table 7.2: Compressing $\mathcal{D}$ decreases the computation time and number of iterations for sufficiently large tensors. Mean (over 20 trials) computation time and number of GGN iterations until convergence using a β-divergence cost function with $\beta = 1.5$ when $\hat{\mathcal{D}}$ is a multilinear rank-(S, S, S) LMLRA. The noise is multiplicative Gamma noise with an SNR of 40 dB.

	Compression factor I/S			
	1	5	10	25
$I = 50$				
Optimal solution /20	20	20	20	20
# GGN it.	26.1	29.2	27.9	34.2
Time (s)	4.21	5.06	4.47	5.43
$I = 100$				
Optimal solution /20	19	19	19	19
# GGN it.	34.7	27.7	40.3	50.0
Time (s)	23.34	12.08	15.22	12.84
$I = 200$				
Optimal solution /20	16	18	18	18
# GGN it.	66.9	64.7	66.8	57.11
Time (s)	138.94	85.06	64.66	48.55

For a CPD fitting problem, the inexact GGN method (with Jacobi preconditioning) is compared to the exact GGN method and two first-order methods, applied in the generalized CPD framework [105]. The first-order methods are the L-BFGS-B method [245], which is a bound-constrained quasi-Newton method and the Adam method [117,120], which is a stochastic gradient descent method that uses both the first and second moments of the gradient to adjust the learning rate in every iteration. The first moment is used to maintain momentum in the direction of the solution and to overcome local minima, while the second moment is used to focus the update on variables that have not been changed significantly in recent iterations. The second-moment approach is called RMSProp [205], which was designed to improve on the speed of AdaGrad [64]. Adam performs a number of epochs until its stopping criterion is reached. The cost function is evaluated once per epoch, but the gradient is updated many times in every epoch, always using a different sample of tensor entries. In the experiment, the cost function is approximated in every epoch by uniformly sampling $\min\left(10^6, I^3/10\right)$ tensor entries, while 1000 gradient updates are applied per update, with 1000 entries per gradient update, following the default values from [9,120]. Adam is used with the standard parameter values [117] and its stopping criterion is the second failed epoch, i.e., the second epoch in which the cost function value increases instead of decreases [120].

Table 7.3: Proper compression of $\mathcal{D}$ decreases the computation time for tensors of large enough dimensions. Mean (over 20 trials) computation time and number of GGN iterations until convergence starting close to the LS solution. A β-divergence cost function is used with $\beta \in \{0.5, 1.5\}$. $\hat{\mathcal{D}}$ has multilinear rank (S, S, S). The mean (over all GGN iterations) number of CG (for the convex f_n of $\beta = 1.5$) and MINRES (for the nonconvex f_n of $\beta = 0.5$) iterations to solve $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{g}$ is also listed. Compression factors I/S are compared for an $I \times I \times I$-tensor of rank 4. The noise is multiplicative Gamma noise with an SNR of 40 dB.

	Compression factor I/S			
Criterion ($\beta = 1.5$)	1	5	10	25
$I = 50$				
Number of GGN it.	3.2	3.3	4.2	5.4
Mean number of CG it.	56.3	56.7	56.9	56.8
Time (s)	0.34	0.59	0.70	0.89
$I = 100$				
Number of GGN it.	3.0	3.0	3.1	4.2
Mean number of CG it.	77.3	77.3	77.3	78.0
Time (s)	2.07	1.60	1.49	1.42
$I = 200$				
Number of GGN it.	3.0	3.0	3.0	3.3
Mean number of CG it.	76.5	76.7	76.7	77.1
Time (s)	13.36	5.62	4.55	4.54
Criterion ($\beta = 0.5$)				
$I = 50$				
Number of GGN it.	3.9	7.7	12.2	12.3
Mean number of MINRES it.	51.9	53.3	57.1	53.8
Time (s)	0.37	1.43	4.72	3.73
$I = 100$				
Number of GGN it.	3.8	4.1	9.3	9.6
Mean number of MINRES it.	121.1	123.5	132.6	156.9
Time (s)	3.71	2.95	8.96	11.15
$I = 200$				
Number of GGN it.	3.8	3.9	4.5	10.3
Mean number of MINRES it.	135.1	137.0	139.7	164.7
Time (s)	26.62	8.17	7.54	25.99

Rank-5 tensors with dimensions $I \times I \times I$ are generated with factor matrix entries sampled from $U(0, 1)$ and the tensors are perturbed by multiplicative Gamma noise with an SNR of 20 dB. All methods are initialized by sampling random factor matrices with entries in $U(0, 1)$ and performing five iterations of the GN LS algorithm from Tensorlab [228] (`cpd_gndl`). For the inexact GGN method, the $\mathcal{D}$-tensor is compressed to a multilinear rank-$(I/5, I/5, I/5)$ tensor. The cases $I \geq 200$ are omitted for the exact GGN method, as the use of the full $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix caused out-of-memory problems on the available hardware. The results can be found in Fig. 7.5. In the center plot it is shown that, regardless of the tensor dimensions, the GGN and L-BFGS-B methods find the same solution for the CPD, while the solution found by Adam is worse by a factor of about 2. From the left plot of Fig. 7.5, one can see that the computational costs of the L-BFGS-B and Adam methods grow faster with increasing tensor dimensions than the inexact GGN method, especially for $80 \leq I \leq 160$. Consequentially, for larger tensors, the inexact GGN method is significantly more efficient. For these tensors, objective function and gradient evaluations are relatively more expensive than for smaller tensors. As a result, it is critical to limit the number of iterations of the optimization algorithm. From the right plot of Fig. 7.5, one can see that the GGN methods require much fewer iterations than the L-BFGS-B method. The Adam method cannot be directly compared with the other methods, as it uses epochs instead of iterations. Adam generally requires about 10 epochs to converge in our experiments, while in every epoch 1000 iterations are performed on a part of the data. Combined with the faster iterations of the inexact GGN method compared to the exact GGN method, its lower number of iterations explains why the inexact GGN method scales much better than the other methods.

In Fig. 7.6, the same experiment is repeated, but now the rank R is systematically increased and $I = 120$ is kept constant. From the left plot of Fig. 7.6, we can see that the computation time of the exact GGN method grows very quickly with increasing rank, so only ranks $R \leq 8$ were computed for this method. In contrast, the inexact GGN method has a much slower increase in computation time and can easily handle higher ranks. Still, for $R \geq 16$, the L-BFGS-B method becomes more computationally efficient than the GGN method. The GGN method is thus particularly effective when I is high and the rank R is low. Fortunately, these conditions frequently occur in practice. For example, in signal processing applications, it is often important to recover a small number of latent components from a large dataset with high precision [35]. For such cases, the GGN method can thus be significantly faster than first-order methods.

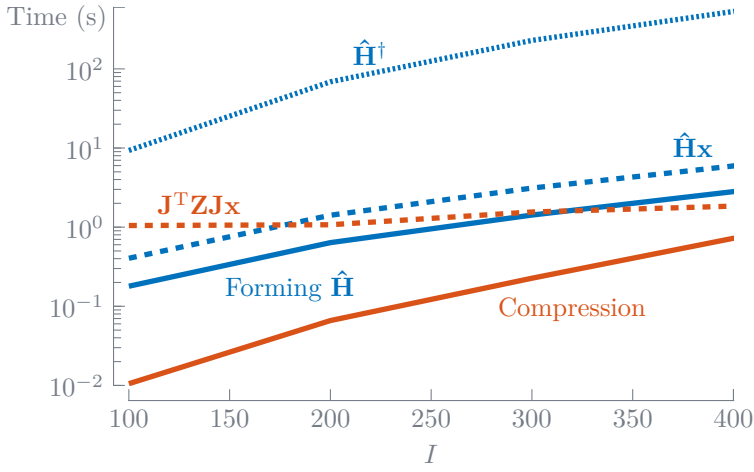

Figure 7.4: The inexact GGN method with compression is faster than the exact GGN method for sufficiently large tensors. Mean time (over 20 trials) for the compression of $\mathcal{D}$ and for 100 evaluations of $\mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}\mathbf{x}$ after compression. For comparison, the mean time for the construction of the Hessian approximation $\hat{\mathbf{H}} = \mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}$, for the computation of the pseudoinverse $\hat{\mathbf{H}}^\dagger$ and for 100 products $\hat{\mathbf{H}}\mathbf{x}$ are also shown for the exact GGN method. The tensors have dimensions $I \times I \times I$ and rank 10. The CPD model has rank 10 as well. A multilinear rank-$(I/10, I/10, I/10)$ approximation is used for $\mathcal{D}$.

7.4.4 Large-scale tensors

For tensors of dimensions $I \times I \times I$, with $I > 300$, the inexact GGN method is still too computationally expensive to obtain a solution in a reasonable amount of time. Fortunately, the proposed techniques are independent of the specific solver that is used. Consequentially, they can also be applied in other second-order methods such as stochastic algorithms to tackle large-scale problems. Below, we apply the techniques from this chapter to inexact GGN within the RBS framework [222]. Instead of using all tensor entries in every iteration of the algorithm, the RBS method samples a subtensor and only updates the model with information from that subtensor. As a result, the RBS method can quickly find the CPD of a large-scale tensor with high accuracy. For this experiment, we have extended the RBS strategy so that it can handle non-LS cost functions as well: in every iteration, instead of the GN method, the GGN method is used to update the tensor with information from the sampled subtensor. In this way, it becomes feasible to apply the GGN method for the decomposition of large-scale tensors.

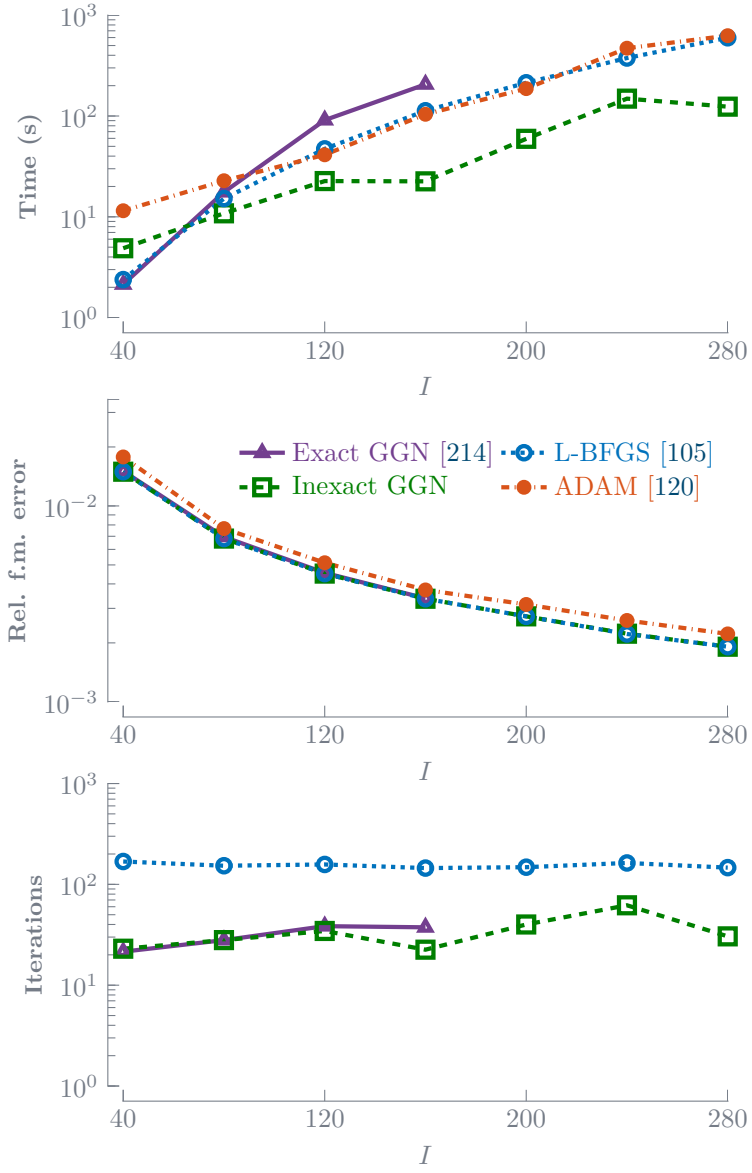

Figure 7.5: The inexact GGN method with compression scales much better than the exact GGN method and the L-BFGS and Adam methods for increasing tensor dimensions, while the solution accuracy is at least as good. Median (over 20 trials) time (top), relative factor matrix error (center) and number of iterations (bottom) for the computation of a CPD with a β-divergence cost function with $\beta = 1.5$. The tensors have dimensions $I \times I \times I$ and rank 5. The CPD model has rank 5 as well. A multilinear rank-$(I/5, I/5, I/5)$ approximation was used for $\hat{\mathcal{D}}$.

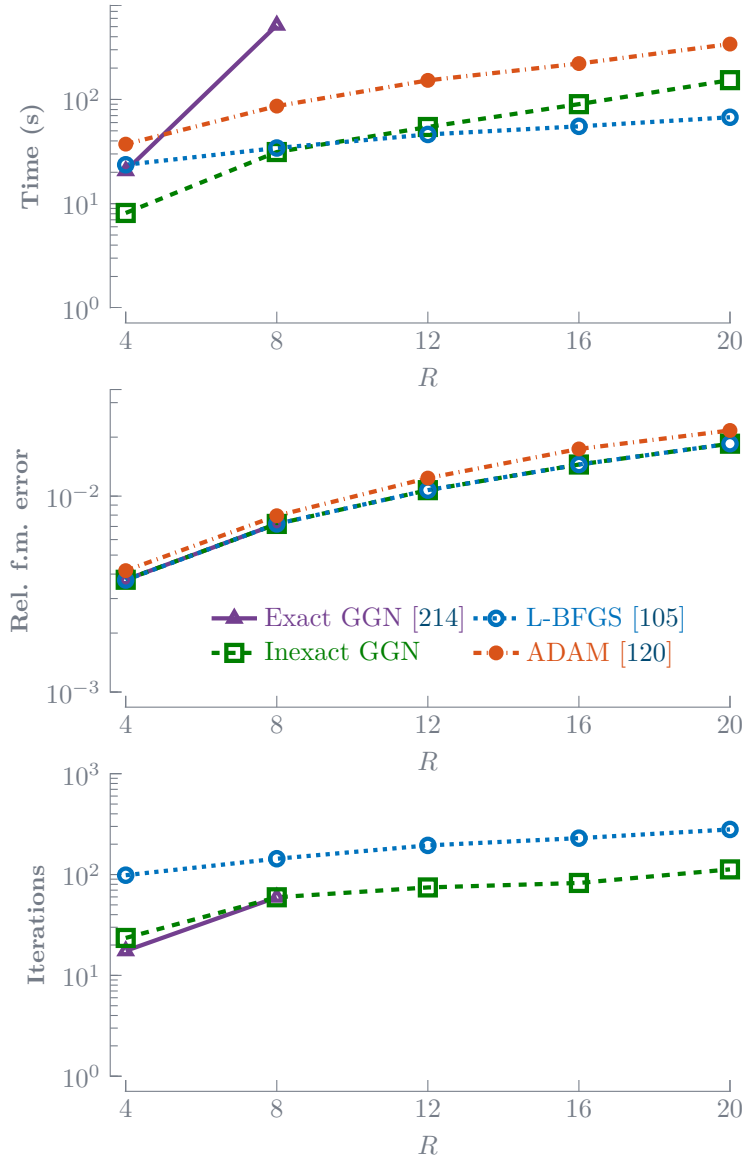

Figure 7.6: The inexact GGN method with compression scales much better than the exact GGN method for increasing rank R. Still, for $R \geq 16$, the first-order method L-BFGS is more efficient. The Adam method is less accurate than the other methods. Median (over 20 trials) time (top), relative factor matrix error (center) and number of iterations (bottom) for the computation of a CPD with a β-divergence cost function with $\beta = 1.5$. The tensors have dimensions $120 \times 120 \times 120$ and rank R. The CPD model has rank R as well. A multilinear rank-$(20, 20, 20)$ approximation was used for $\hat{\mathcal{D}}$.

Table 7.4: Using the RBS framework, the GGN method can be applied to obtain CPDs of large tensors with non-LS cost functions. Compared to the first-order optimization method Adam, its solution is both more accurate and obtained in a shorter amount of time. All results are medians over 10 trials.

	Time (s)	Rel. f.m. error
GGN RBS	268.29	$2.68 \cdot 10^{-3}$
GCPD with Adam	649.02	$4.22 \cdot 10^{-3}$
GN RBS	25.93	$2.86 \cdot 10^{-3}$

We tested the GGN RBS method on a dense $500 \times 500 \times 500$ tensor of rank 10 with factor matrix entries sampled from $U(0, 1)$. Multiplicative Gamma noise is added with an SNR of 20 dB. Three methods are compared: the GGN RBS method with a β-divergence cost function with $\beta = 1.5$, the first-order stochastic Adam method [120] within the generalized CPD framework [105] with the same cost function and the GN RBS method which uses the LS cost function. The RBS methods use a block size of $80 \times 80 \times 80$ and stop when the relative step becomes smaller than 10^{-5}. They use a maximal step size $\|\mathbf{p}\|$ of 10 for the first 350 iterations and thereafter, a maximal step size of $0.005 \cdot 0.99^{(k-350)}$ is used, with $k > 350$ the iteration number. The Adam method samples 1 000 000 values when evaluating the cost function and in each of the 1000 iterations per epoch, it samples 100 000 values for the gradient. The method stops when a second failed epoch occurs.

In Table 7.4, the results can be seen for the three methods. From the right column, one can see that GGN RBS obtains the best solution. The solution of the GN RBS method is worse because it uses a less suitable cost function, while the solution of Adam is simply less accurate. From the left column, it can be seen that the GGN RBS method is slower than GN RBS, which is to expected, as the latter uses a cost function that is much less expensive to evaluate. More importantly, the GGN RBS method is significantly faster than the first-order Adam method, while obtaining a more accurate solution.

The GGN method can thus be applied in a stochastic context to obtain an accurate solution for the CPD fitting problem with a non-LS cost function in a reasonable amount of time. Compared to an existing first-order stochastic method, it is both more accurate and faster.

7.4.5 Separating fMRI sources

In neuro-imaging, fMRI is a noninvasive technique that indirectly studies brain activity by measuring fluctuations of the Blood Oxygenation Level Dependent (BOLD) signal [130]. The localization of the activated brain areas from fMRI data is a challenging blind source separation (BSS) problem. The sources consist of a combination of spatial maps (activated areas) and time courses (activation times). Following the methodology of [31], we use a CPD to perform blind fMRI source unmixing to recover eight simulated spatio-temporal sources. The spatial and temporal modes of these sources are shown in Fig. 7.7. They are generated based on knowledge of the statistical characteristics of the underlying neuronal sources. The spatial signatures have dimensions 60×60 and are vectorized into vectors of length 3600. Over time, 100 different samples are collected. For fMRI data, the phase difference between activated and resting states of a voxel induced by the task activation, is usually less than $\pi/9$ [106], so the phases of each pixel are sampled from $U(\pi/18, \pi/18)$. Four subjects are considered that each generate the same sources, but with different amplitudes depending on the person. After mixing, a $3600 \times 100 \times 4$-tensor $\mathcal{T}$ is obtained.

In practice, the real and imaginary parts of fMRI data are generally obtained separately and are independently perturbed by additive Gaussian noise with equal variance during data collection. As explained in Sec. 7.3, the magnitude of the fMRI data, which is used for analysis, follows a Rician distribution. The cost function from Sec. 7.3 is thus expected to perform better than the LS cost function for this case. Hereafter, this cost function will be called the Rician-dedicated cost function.

Additive Gaussian noise with variance σ is added to both the real and imaginary parts of the tensor $\mathcal{T}$ such that the SNR of its real part is 25 dB. CPDs are then found for the tensor of moduli $|\mathcal{T}|$ using two different cost functions: the LS cost function, and the Rician-dedicated cost function. In Figs. 7.8 and 7.9, the unmixed sources are shown for these two cost functions, respectively. They have been reordered to match the ordering of the original sources. Aside from some small artifacts leaking through from other sources, most sources are found with relatively high accuracy by both methods. Visually, the results from both methods cannot be distinguished. Comparing the mean correlations of the unmixed sources with the true sources for the two cost functions, one also finds both to be very similar: 0.9873 for the LS cost function, and 0.9875 for the Rician-dedicated cost function. In Table 7.5, the peak SNR (PSNR) and structural similarity index (SSI) [231] of the spatial components of the 8 recovered sources are displayed. The PSNR of two images $\mathbf{X}, \mathbf{Y} \in [0, 1]^{I \times J}$ is

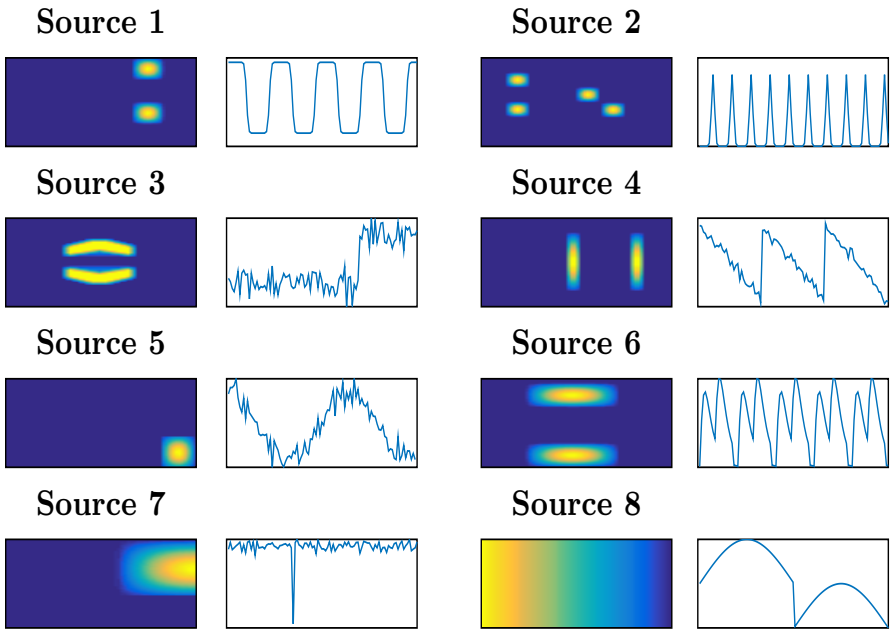

Figure 7.7: Spatial (left) and temporal (right) modes of eight simulated fMRI sources.

defined as

$$\mathrm{PSNR}(\mathbf{X},\mathbf{Y}) = 10\log_{10}\left(\frac{\rho^2 IJ}{\|\mathbf{X}-\mathbf{Y}\|_{\mathrm{F}}^2}\right),$$

with ρ the maximal value of the image. The SSI of $\mathbf{X}$ and $\mathbf{Y}$ is defined as

$$\mathrm{SSI}(\mathbf{X},\mathbf{Y}) = \frac{(2\mu_{\mathbf{X}}\mu_{\mathbf{Y}}+C_1)\,(2\sigma_{\mathbf{XY}}+C_2)}{(\mu_{\mathbf{X}}^2+\mu_{\mathbf{Y}}^2+C_1)\,(\sigma_{\mathbf{X}}^2+\sigma_{\mathbf{Y}}^2+C_2)},$$

wherein $\mu_{\mathbf{X}}$, $\mu_{\mathbf{Y}}$, $\sigma_{\mathbf{X}}$, $\sigma_{\mathbf{Y}}$, and $\sigma_{\mathbf{XY}}$ are the local means, standard deviations, and cross-covariance, respectively. The constants C_1 and C_2 are set to the default values 0.0001 and 0.0009, respectively [229]. The SSI takes the differences in luminance, contrast and structure of both images into account. It lies between 0 and 1, with 1 indicating a perfect match. For both metrics, it can be seen in Table 7.5 that the results for the Rician-dedicated cost function are indeed somewhat better than those for the LS cost function.

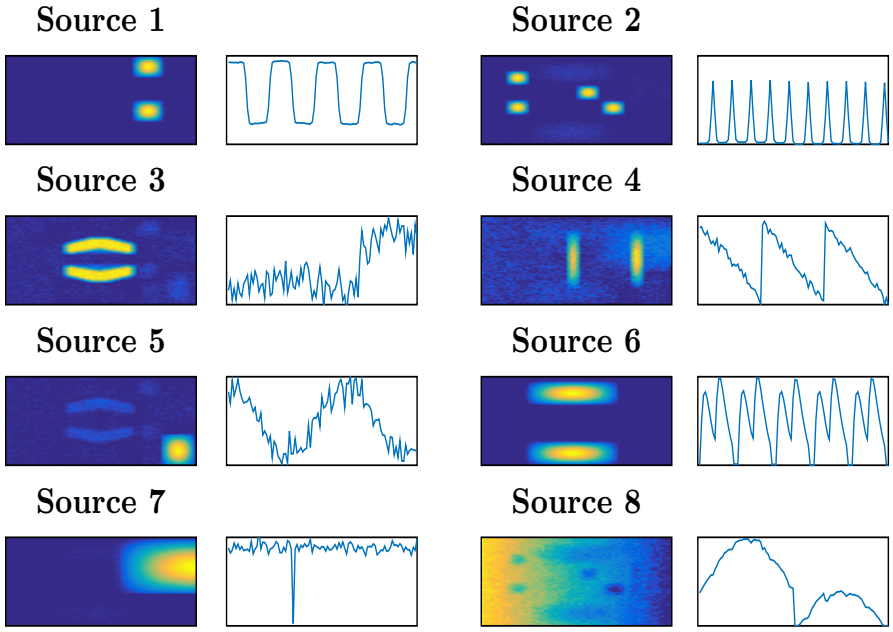

Figure 7.8: Unmixed fMRI sources when a CPD with a standard LS cost function is used.

Table 7.5: The recovered spatial sources are slightly more accurate with the Rician-dedicated cost function compared to the LS cost function. PSNRs and SSIs of the eight spatial sources recovered using a CPD with an LS and with a Rician-dedicated cost function, together with their mean values.

	Source								
PSNR	1	2	3	4	5	6	7	8	Mean
LS	64.82	53.07	54.64	42.00	48.78	74.14	66.65	53.32	57.18
Rice	67.26	54.36	59.10	42.14	50.19	74.40	64.81	53.22	58.19
SSI									
LS	0.9982	0.9651	0.9777	0.7749	0.9306	0.9998	0.9985	0.9919	0.9545
Rice	0.9992	0.9737	0.9940	0.8009	0.9530	0.9999	0.9975	0.9906	0.9636

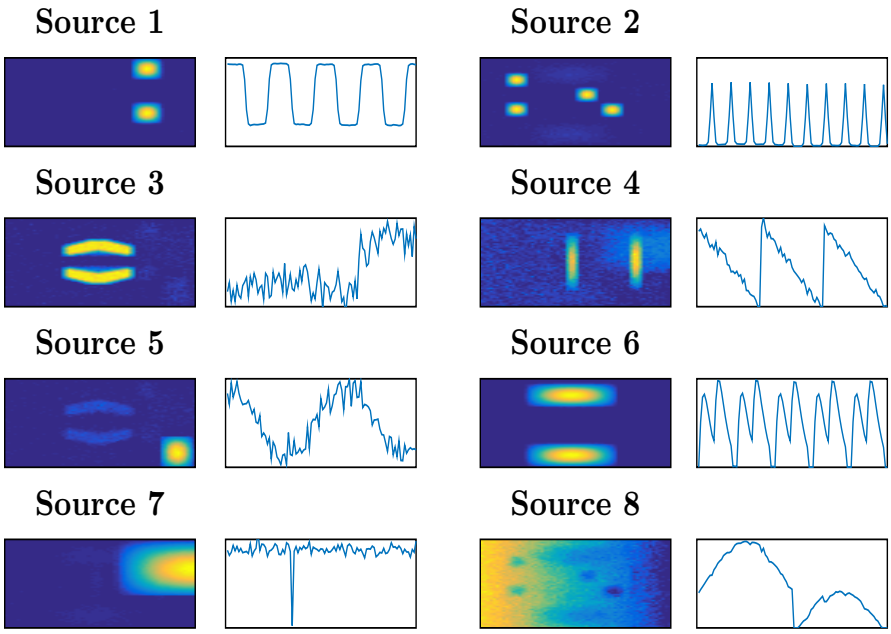

Figure 7.9: Unmixed fMRI sources when a CPD with a Rician-dedicated cost function is used.

7.5 Conclusion

An inexact GGN method was proposed to fit the CPD of a tensor for non-LS entry-wise cost functions. Rician distributed data is used as an example where the application of an alternative cost function can lead to more meaningful results. The proposed inexact GGN method obtains the new step direction $\mathbf{p}$ by solving the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ iteratively. A considerable speedup is achieved by exploiting the multilinear structure of the Hessian approximation, by compressing the second-order information with an LMLRA and by applying a specialized preconditioner. The inexact GGN method scales much better than the exact GGN method and other existing methods without compromising the fast convergence that the use of second-order information yields. The proposed techniques can be applied in other second-order methods as well, as shown for a large-scale stochastic algorithm. The performance of the proposed method is demonstrated on synthetic data and on a simulated biomedical experiment.

Rank-one tensor approximation with beta-divergence cost functions

8

ABSTRACT

β-divergence cost functions generalize three popular cost functions for low-rank tensor approximation by interpolating between them: the least-squares (LS) distance, the Kullback—Leibler (KL) divergence and the Itakura—Saito (IS) divergence. For certain types of data and specific noise distributions, beta-divergence cost functions can lead to more meaningful low-rank approximations than those obtained with the LS cost function. Unfortunately, much of the low-rank structure that is heavily exploited in existing second-order LS methods, is no longer exploitable when moving to general β-divergences. In this chapter, we show that, unlike in the general rank-R case, rank-1 structure can still be exploited. We therefore propose an efficient method that uses second-order information to compute nonnegative rank-1 approximations of tensors for general β-divergence cost functions.

This chapter is based on M. Vandecappelle, N. Vervliet, and L. De Lathauwer,"Rank-1 tensor approximation with beta-divergence cost functions," in 2019 27th European Signal Processing Conference (EUSIPCO19), Sep. 2019, pp. 1–5. The figures have been updated for consistency. The specific contributions of the doctoral candidate are listed in Section 11.1.

8.1 Introduction

With the recent explosion of interest in data and data analytics, tensors have found their way into many applications in machine learning and signal processing [35, 174]. These higher-order generalizations of vectors and matrices represent higher-order datasets while preserving their higher-order structure. Unfortunately, as the number of entries of an Nth-order tensor with dimensions $I \times I \times \ldots \times I$ grows as I^N, its size quickly becomes problematic. By working with a low-rank approximation of the tensor instead of the full tensor, the required storage space and computation time can often be reduced dramatically, while still maintaining all the important features of the data in the tensor. A rank-R approximation of a tensor consists of R rank-1 terms, which themselves are tensors that can be written as an outer product of N nonzero vectors. To obtain a rank-R approximation, several algebraic and optimization-based methods have been developed [2, 62, 157, 183, 223].

By using β-divergences d_β as cost functions, the standard least-squares cost function can be generalized. These divergences, a special class of Bregman divergences [100], are defined for $\beta \in \mathbb{R}$ and interpolate continuously between the Itakura–Saito (IS) divergence ($\beta = 0$), Kullback–Leibler (KL) divergence ($\beta = 1$) and the least-squares distance ($\beta = 2$). When used as a cost function, values of $\beta < 2$ penalize errors on small entries more heavily compared to the least-squares distance, while the converse is true for values of $\beta > 2$. β-divergences are thus particularly useful when the data consists of entries of different magnitudes. As such, audio data is a prime example where β-divergence cost functions (with $\beta < 2$) outperform the least-squares cost function as these manage to capture the low intensity components of the signals better; see, for example, [72] and references therein for a profound discussion of the use of β-divergences in nonnegative matrix factorization (NMF) of audio spectra. Notably, an NMF with the KL- or IS-divergence corresponds to maximum likelihood estimation under the assumption of Poisson distributed data and data perturbed by multiplicative Gamma noise, respectively [79], while the least-squares cost function assumes additive independent and identically distributed (i.i.d.) Gaussian noise. The β-divergence $d_\beta(x, y)$ is defined as follows:

$$d_\beta(x,y) = \begin{cases} \frac{x^\beta + (\beta-1)y^\beta - \beta\left(xy^{(\beta-1)}\right)}{\beta(\beta-1)} & \beta \in \mathbb{R} \setminus \{0,1\} \\ x \log\left(\frac{x}{y}\right) - x + y & \beta = 1 \\ \frac{x}{y} - \log\left(\frac{x}{y}\right) - 1 & \beta = 0 \end{cases} \tag{8.1}$$

Where the least-squares distance considers only the absolute difference between a target value x and its estimate y, other β-divergences also take the size of

x and y themselves into account. If $\beta = 0$, the divergence is scale invariant, meaning that $d_0(x, 2x)$ is the same for any value of x. If $\beta > 0$, $d_\beta(x, 2x)$ is larger for large values of x, while for $\beta < 0$, $d_\beta(x, 2x)$ is larger for small values of x.

8.1.1 Notation

We denote scalars, vectors, matrices and tensors by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase ($\mathbf{A}$), and calligraphic letters ($\mathcal{A}$), respectively. For simplicity, we will only consider third-order tensors. Powers ($\mathbf{X}^{\bullet a}$), divisions and logarithms of matrices are entry-wise throughout the text. The Kronecker, column-wise Khatri–Rao, and Hadamard products of matrices and the outer product of vectors are denoted by $\otimes$, $\odot$, $*$ and $\circledast$ respectively, while the mode-n tensor contraction (i.e., tensor-vector product) of a tensor $\mathcal{T}$ and a vector $\mathbf{x}$ is written as $\mathcal{T} \cdot_n \mathbf{x}$. The transpose of a matrix $\mathbf{X}$ is denoted by $\mathbf{X}^{\mathrm{T}}$ and $\mathrm{diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its diagonal. When using subscripts to specify certain entries of a tensor $\mathcal{T}$, a colon (:) refers to all entries of a specific mode. The vectorization of the tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ maps $x_{i_1 i_2 i_3}$ to $\mathrm{vec}(\mathcal{X})_q$, with $q = i_1 + (i_2 - 1)I + (i_3 - 1)IJ$. Its inverse is denoted by $\mathrm{unvec}(\mathbf{x})$. A mode-$n$ fiber of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfolding $\mathbf{T}_{(n)}$ of a tensor $\mathcal{T}$ is constructed by stacking all its mode-n fibers as columns of a matrix. For example: $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{i_2, q}$, with $q = i_1 + (i_3 - 1)I$. The $(K \times K)$-identity matrix is written as $\mathbf{I}_K$ and $\mathbf{1}$ is a column vector of appropriate dimensions, consisting of only ones. For a rank-R approximation $\mathcal{T} \approx \sum_{r=1}^{R} \mathbf{a}_r \circledast \mathbf{b}_r \circledast \mathbf{c}_r$, the column vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ can be collected into factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. The rank-R approximation is then written as $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

8.2 β-divergence rank-R approximation

For tensor data, β-divergence cost functions have been used to compute nonnegative rank-R approximations of tensors. The proposed algorithms [34, 38, 94, 105], however, are all either first-order methods, are alternating schemes, only consider KL- or IS-divergences, or are combinations of these. In contrast, we propose a method that uses second-order information to achieve near second-order convergence, is all-at-once and applicable for any value of β. For a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ and its low-rank approximation $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, the β-divergence cost function (8.1) leads to the following optimization problem:

$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f(\mathbf{x})$, with $\mathbf{x} = [\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})]$ and

$$f(\mathbf{x}) = \frac{1}{\beta(\beta-1)} \sum_{n=1}^{N} t_n^{\beta} + (\beta-1) m_n^{\beta} - \beta \left(t_n m_n^{(\beta-1)} \right), \tag{8.2}$$

where $N = IJK$ and n runs over all tensor entries. Using (8.1), the cost functions for the degenerate special cases $\beta = 1$ (KL divergence) and $\beta = 0$ (IS divergence) are:

$$f(\mathbf{x}) = \sum_{n=1}^{N} t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \text{and} \tag{8.3}$$

$$f(\mathbf{x}) = \sum_{n=1}^{N} \frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1, \tag{8.4}$$

respectively. Note that the case $\beta = 2$ corresponds to the least-squares cost function. β-divergences are only defined for nonnegative values, so we constrain $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative matrices. For any value of β, the gradient $\mathbf{g} = [\mathbf{g}_{\mathbf{A}}; \mathbf{g}_{\mathbf{B}}; \mathbf{g}_{\mathbf{C}}]$ is:

$$\begin{aligned} \mathbf{g}_{\mathbf{A}} &= \text{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) \\ \mathbf{g}_{\mathbf{B}} &= \text{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) \\ \mathbf{g}_{\mathbf{C}} &= \text{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right), \end{aligned} \tag{8.5}$$

with $\mathcal{R} = \frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}$. Note that $\mathbf{g}$ can equivalently be written as $\mathbf{g} = \mathbf{J}^{\mathrm{T}}\text{vec}(\mathcal{R})$, with $\mathbf{J}$ the Jacobian of the low-rank model $\mathcal{M}$ w.r.t. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. Only $\mathcal{R}$ depends on the specific cost function that is chosen. For $\beta = 2$, $\mathcal{R}$ is simply the residual $\mathcal{M} - \mathcal{T}$. For a second-order method, (a good approximation of) the Hessian is required. In [223], an approximation of the form $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is proposed. This approximation is further generalized to arbitrary entry-wise cost functions in Chapter 7, using similar ideas as in [169]. The diagonal matrix $\mathbf{Z}$ has $\text{vec}(\mathcal{D})$ as its diagonal, where $\mathcal{D}$ is the entry-wise second-order derivative of the β-divergence cost function w.r.t. to $\mathcal{M}$:

$$\mathcal{D} = \frac{(\beta-1)\mathcal{M} - (\beta-2)\mathcal{T}}{\mathcal{M}^{\bullet(3-\beta)}} = \frac{(\beta-2)(\mathcal{M}-\mathcal{T}) + \mathcal{M}}{\mathcal{M}^{\bullet(3-\beta)}}. \tag{8.6}$$

Note that for $\beta \notin [1,2]$, $\mathcal{D}$ can have negative values when the low-rank approximation $\mathcal{M}$ is not close enough to the tensor $\mathcal{T}$. Consequently, the Hessian approximation is not guaranteed to be positive semidefinite (PSD), i.e., to have only eigenvalues ≥ 0. Positive semidefiniteness of the Hessian

approximation guarantees that the local quadratic approximation of the cost function has a minimum. Without this property, one can end up in a saddle point instead. If $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is not PSD, one needs to use an optimization algorithm that can handle indefinite Hessian approximations. An alternative is to first find a good initialization, e.g., the solution for $\beta = 2$, before moving to the β-divergence algorithm.; see Chapter 7 for an extended discussion. For $\beta = 2$, the matrix $\mathbf{Z}$ is equal to the identity matrix and one recovers the Gauss-Newton approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ of the Hessian, which is the Gramian of the Jacobian.

Due to the shape of β-divergence cost functions, simply projecting the unconstrained Gauss–Newton point to the nonnegative orthant generally leads to poor results. In the kth iteration of the algorithm, the current model is $\mathbf{x}_k$ and a nonlinear projected gradient method is used to find the next step $\mathbf{p}$ such that $\mathbf{x}_k + \mathbf{p} = \mathbf{x}_{k+1}$. The cost function $f(\mathbf{x})$ is approximated by its second-order Taylor expansion $m(\mathbf{x})$. To assure convergence of the algorithm, we first find the Cauchy point $\mathbf{p}_C$ of $m(\mathbf{x})$ within an l_∞-norm trust region that incorporates the nonnegativity constraints on $\mathbf{x}$. Starting from this point, we then approximately solve the quadratic program $\min_{\mathbf{p}} m(\mathbf{x} + \mathbf{p})$ subject to $\mathbf{x} + \mathbf{p} \geq 0$ within the trust region and use $\mathbf{p}$ as the next step of the algorithm.

8.3 Rank-1 approximations

The low-rank structure of the model can be exploited when $\beta = 2$. For example: products of the form $\mathbf{M}_{(1)}(\mathbf{C} \odot \mathbf{B}) = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{B})$ can be efficiently computed as $\mathbf{A}\left((\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})\right)$. Unfortunately, this structure is no longer exploitable in the computation of the objective function, gradient and Hessian approximation for other values of β. The model has to be expanded to the full tensor before computing products, sums or powers of its entries $m_n = \sum_{r=1}^{R} a_{ir}b_{jr}c_{kr}$, which negatively influences both time and storage complexity of the method. Fortunately, for $R = 1$, we have $\mathcal{M} = [\![\mathbf{a}, \mathbf{b}, \mathbf{c}]\!]$ and thus:

$$m_n^\beta = (a_i b_j c_k)^\beta = a_i^\beta b_j^\beta c_k^\beta, \tag{8.7}$$

which allows us to exploit the multilinear structure of the model and simplifies the computations significantly compared to the rank-R case. Dedicated, efficient expressions for the objective function, gradient and Hessian approximation are derived in this section. In various applications, e.g., [171, 238], one only needs a rank-1 approximation of a tensor in order to extract its most important component. Using similar ideas as in [85, 116, 170], the method could also be used to extract more than one nonnegative rank-1 term from a nonnegative tensor by projecting the deflated tensor to $\mathbb{R}^+$ (if necessary) between the

extraction of successive rank-1 terms. Note that for the KL-divergence, there exists a closed-form solution in the rank-1 case [108].

8.3.1 Cost function

Writing $s_{\mathbf{x}}^y = \sum_{q=1}^{Q} x_q^y$ for a vector $\mathbf{x} \in \mathbb{R}^Q$, we can rewrite $f(\mathbf{x})$ from (8.2) to obtain the optimization problem $\min_{\mathbf{a},\mathbf{b},\mathbf{c}} f(\mathbf{x})$ with f equal to

$$\frac{1}{\beta^2-\beta}\left[\left(\sum_{n=1}^{N} t_n^\beta\right) + (\beta-1)s_{\mathbf{a}}^\beta s_{\mathbf{b}}^\beta s_{\mathbf{c}}^\beta - \beta\mathcal{T} \cdot_1 \mathbf{a}^{\bullet(\beta-1)} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right]. \quad (8.8)$$

The first term $\left(\sum_{n=1}^{N} t_n^\beta\right)$ does not involve the model parameters and can be precomputed. Although adding this term does not change the optimal $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, it gives us a frame of reference to assess the quality of the model, as it assures that $f = 0$ when $m_n = t_n, \forall n \in \{1, \ldots, N\}$. The second term follows from (8.7). It allows us to only compute the (non-integer) power of the entries of the three vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ instead of the power of all tensor entries m_n, which offers a considerable efficiency gain: $\mathcal{O}(I + J + K)$ flops and storage compared to $\mathcal{O}(IJK)$. In the last term, we again use (8.7) to avoid forming the full tensor $\mathcal{M}$ and computing (non-integer) powers of all its entries. For the KL-divergence, i.e., $\beta = 1$, the cost function (8.3) becomes:

$$\begin{aligned} f &= \left(\sum_{n=1}^{N} t_n \log(t_n) - t_n\right) - \left(\sum_{n=1}^{N} t_n \log(m_n)\right) + s_{\mathbf{a}}^1 s_{\mathbf{b}}^1 s_{\mathbf{c}}^1 \\ &= \left(\sum_{n=1}^{N} t_n \log(t_n) - t_n\right) - \mathbf{1}^{\mathrm{T}} \begin{bmatrix} \mathbf{T}_{(1)}^{\mathrm{T}} \log(\mathbf{a}) \\ \mathbf{T}_{(2)}^{\mathrm{T}} \log(\mathbf{b}) \\ \mathbf{T}_{(3)}^{\mathrm{T}} \log(\mathbf{c}) \end{bmatrix} + s_{\mathbf{a}}^1 s_{\mathbf{b}}^1 s_{\mathbf{c}}^1. \end{aligned}$$

The first term can again be precomputed. For the second term, because of the property $\log(abc) = \log(a) + \log(b) + \log(c)$, we only need to compute logarithms of the three vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ instead of logarithms of all entries of $\mathcal{M}$. For the last term, the sum over all tensor entries is converted to three vector sums, as before. For the IS-divergence, i.e., $\beta = 0$, the cost function (8.4) can be rewritten as follows:

$$f = -N - \left(\sum_{n=1}^{N} \log(t_n)\right) + \mathcal{T} \cdot_1 \frac{1}{\mathbf{a}} \cdot_2 \frac{1}{\mathbf{b}} \cdot_3 \frac{1}{\mathbf{c}} + \mathbf{1}^{\mathrm{T}} \begin{bmatrix} JK \log(\mathbf{a}) \\ IK \log(\mathbf{b}) \\ IJ \log(\mathbf{c}) \end{bmatrix}.$$

The same ideas can be used as in the previous two cases.

8.3.2 Gradient

The gradient $[\mathbf{g_a}; \mathbf{g_b}; \mathbf{g_c}]$ can be simplified in a similar manner as the cost function. For general β, $\mathbf{g_a}$ from (8.5) can be written as

$$\begin{aligned}\mathbf{g_a} &= \sum_{j=1}^{J}\sum_{k=1}^{K} \frac{\mathbf{m}_{:jk} - \mathbf{t}_{:jk}}{\mathbf{m}_{:jk}^{\bullet(2-\beta)}} b_j c_k \\ &= s_{\mathbf{b}}^{\beta} s_{\mathbf{c}}^{\beta} \mathbf{a}^{\bullet(\beta-1)} - \mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right]. \end{aligned} \tag{8.9}$$

Analogously, we find $\mathbf{g_b}$ and $\mathbf{g_c}$. Note that we never need to explicitly form the $(I \times J \times K)$-tensor $\left(\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}\right)$ and thus also avoid computing (non-integer) powers of all entries of $\mathcal{M}$. For $\beta \notin \{0,1\}$, all factors of the first term have already been obtained in (8.8), so only a scalar-vector multiplication is required. For the second term, both factors of the Hadamard product are new, so we need to compute an entry-wise vector power and a contraction. Both factors are reused in the computation of the Hessian approximation, however.

8.3.3 Hessian approximation

The full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ consists of 3×3 blocks $\mathbf{G}^{(p,q)}$, with $p, q \in \{1,2,3\}$. Using $\mathcal{D}$ from (8.6), the diagonal and off-diagonal blocks are of the form

$$\mathbf{G}^{(1,1)} = \operatorname{diag}\left(\mathcal{D} \cdot_2 \mathbf{b}^{\bullet 2} \cdot_3 \mathbf{c}^{\bullet 2}\right) \quad \text{and} \quad \mathbf{G}^{(1,2)} = \left[\mathbf{a}\mathbf{b}^{\mathrm{T}}\right] * \left(\mathcal{D} \cdot_3 \mathbf{c}^{\bullet 2}\right),$$

respectively; see Chapter 7. Due to symmetry, only the blocks with $p \leq q$ need to be computed. To compute these blocks efficiently, we exploit the rank-1 structure to avoid forming the tensor $\mathcal{D}$ and taking (non-integer) powers of all entries of a tensor. The diagonal blocks can be computed using the same strategies as for the gradient. As mentioned above, we precompute and reuse factors from the gradient when possible. Note that for rank-1 approximations, the diagonal blocks are diagonal matrices. Letting $\mathbf{G}_{\text{diag}}^{(1,1)}$ be the diagonal of $\mathbf{G}^{(1,1)}$, we find:

$$\begin{aligned}\mathbf{G}_{\text{diag}}^{(1,1)} &= \sum_{j=1}^{J}\sum_{k=1}^{K} \frac{(\beta-1)\mathbf{m}_{:jk} - (\beta-2)\mathbf{t}_{:jk}}{\mathbf{m}_{:jk}^{\bullet(3-\beta)}} b_j^2 c_k^2 \\ &= (\beta-1) s_{\mathbf{b}}^{\beta} s_{\mathbf{c}}^{\beta} \mathbf{a}^{\bullet(\beta-2)} - (\beta-2)\, \mathbf{a}^{\bullet(\beta-3)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right]. \end{aligned}$$

The matrices $\mathbf{G}^{(2,2)}$ and $\mathbf{G}^{(3,3)}$ can be found similarly. For the off-diagonal blocks, we have

$$\begin{aligned}\mathbf{G}^{(1,2)} &= \left[\mathbf{a}\mathbf{b}^{\mathrm{T}}\right] * \sum_{k=1}^{K} \frac{(\beta-1)\mathbf{M}_{::k} - (\beta-2)\mathbf{T}_{::k}}{\mathbf{M}_{::k}^{\bullet(3-\beta)}} c_k^2 \\ &= \left[\mathbf{a}^{\bullet(\beta-2)} \left(\mathbf{b}^{\bullet(\beta-2)}\right)^{\mathrm{T}}\right] * \left[(\beta-1)s_{\mathbf{c}}^{\beta}\mathbf{a}\mathbf{b}^{\mathrm{T}} - (\beta-2)\left(\mathcal{T} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right)\right].\end{aligned}$$

Analogously, one can find $\mathbf{G}^{(1,3)}$ and $\mathbf{G}^{(2,3)}$. Again, results from the gradient (8.9) are reused in the first term, leaving only a scaled vector outer product to compute. The second term still has to be computed. Note, however, that the factor $\mathcal{T} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}$ can be obtained as an intermediate result of computing $\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}$ in the gradient.

For large-scale problems, one can use the (precondioned) conjugate gradient Steihaug [195] (for convex cost functions) or minres [153] (for nonconvex cost functions) method to approximately solve $\min_{\mathbf{p}} m(\mathbf{x}_k + \mathbf{p})$ with $\mathbf{x}_k + \mathbf{p} \geq 0$. Using this approach, we only need to compute products of the form $[\mathbf{y}_{\mathbf{a}}; \mathbf{y}_{\mathbf{b}}; \mathbf{y}_{\mathbf{c}}] = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}[\mathbf{x}_{\mathbf{a}}; \mathbf{x}_{\mathbf{b}}; \mathbf{x}_{\mathbf{c}}]$, where $\mathbf{y}_{\mathbf{a}}$ and $\mathbf{x}_{\mathbf{a}}$ have the same size as $\mathbf{a}$, etc. Using the previously derived expressions, we can compute such products $\mathbf{y}_{\mathbf{a}} = \mathbf{G}^{(1,1)}\mathbf{x}_{\mathbf{a}} + \mathbf{G}^{(1,2)}\mathbf{x}_{\mathbf{b}} + \mathbf{G}^{(1,3)}\mathbf{x}_{\mathbf{c}}$, without having to form the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, in the following way:

$$\mathbf{G}^{(1,1)}\mathbf{x}_{\mathbf{a}} = \left[\mathbf{x}_{\mathbf{a}} * \mathbf{a}^{\bullet(\beta-3)}\right] * \left[(\beta-1)s_{\mathbf{b}}^{\beta}s_{\mathbf{c}}^{\beta}\mathbf{a} - (\beta-2)\,\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right],$$

$$\mathbf{G}^{(1,2)}\mathbf{x}_{\mathbf{b}} = (\beta-1)s_{\mathbf{c}}^{\beta}\mathbf{a}^{\bullet(\beta-1)}\left[\mathbf{x}_{\mathbf{b}}^{\mathrm{T}}\mathbf{b}^{\bullet(\beta-1)}\right] - (\beta-2)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T} \cdot_2 \mathbf{b}_{\mathbf{x}_{\mathbf{b}}} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right],$$

$$\mathbf{G}^{(1,3)}\mathbf{x}_{\mathbf{c}} = (\beta-1)s_{\mathbf{b}}^{\beta}\mathbf{a}^{\bullet(\beta-1)}\left[\mathbf{x}_{\mathbf{c}}^{\mathrm{T}}\mathbf{c}^{\bullet(\beta-1)}\right] - (\beta-2)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}_{\mathbf{x}_{\mathbf{c}}}\right],$$

where $\mathbf{b}_{\mathbf{x}_{\mathbf{b}}} = \mathbf{x}_{\mathbf{b}} * \mathbf{b}^{\bullet(\beta-2)}$ and $\mathbf{c}_{\mathbf{x}_{\mathbf{c}}} = \mathbf{x}_{\mathbf{c}} * \mathbf{c}^{\bullet(\beta-2)}$. Note that the first terms of $\mathbf{G}^{(1,2)}$ and $\mathbf{G}^{(1,3)}$ both have the factor $(\beta-1)\mathbf{a}^{\bullet(\beta-1)}$, so their sum can be computed as

$$(\beta-1)\mathbf{a}^{\bullet(\beta-1)}\left(s_{\mathbf{c}}^{\beta}\mathbf{x}_{\mathbf{b}}^{\mathrm{T}}\mathbf{b}^{\bullet(\beta-1)} + s_{\mathbf{b}}^{\beta}\mathbf{x}_{\mathbf{c}}^{\mathrm{T}}\mathbf{c}^{\bullet(\beta-1)}\right).$$

The sum of their second terms can also be rewritten as

$$(2-\beta)\,\mathbf{a}^{\bullet(\beta-2)} * \left[\mathcal{T} \cdot_2 \mathbf{b}^{\bullet(\beta-1)} \cdot_3 \mathbf{c}_{\mathbf{x}_{\mathbf{c}}} + \mathcal{T} \cdot_2 \mathbf{b}_{\mathbf{x}_{\mathbf{b}}} \cdot_3 \mathbf{c}^{\bullet(\beta-1)}\right].$$

Computing the product $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}[\mathbf{x_a};\mathbf{x_b};\mathbf{x_c}]$ only requires storing vectors of length I, except for the intermediate steps in the computation of the tensor contractions. Note that even this can be avoided by computing the tensor contractions slice by slice, i.e., computing only one entry at the time. When $\mathcal{T}$ is sparse or structured (e.g., Hankel, Löwner, low multilinear rank), this structure can also be exploited during the computations [226], leading to lower computation times and storage requirements. Similar expressions can be derived for $\mathbf{y_b}$ and $\mathbf{y_c}$. As $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$ is diagonally dominant and its diagonal blocks are diagonal matrices, a Jacobi preconditioner can be expected to perform well. The preconditioner $\mathbf{M}^{-1}$ can be obtained cheaply by computing $\mathbf{G}^{(1,1)}_{\mathrm{diag}}$, $\mathbf{G}^{(2,2)}_{\mathrm{diag}}$ and $\mathbf{G}^{(3,3)}_{\mathrm{diag}}$:

$$\mathbf{M}^{-1} = \mathrm{diag}\left(\left[\frac{1}{\mathbf{G}^{(1,1)}_{\mathrm{diag}}}; \frac{1}{\mathbf{G}^{(2,2)}_{\mathrm{diag}}}; \frac{1}{\mathbf{G}^{(3,3)}_{\mathrm{diag}}}\right]\right).$$

Compared to the expressions for general R, those for the rank-1 case offer a significant efficiency gain. On the one hand, more of the multilinear structure can be exploited. On the other hand, more intermediate results from the objective function and gradient can be reused in the gradient and Hessian approximation, respectively.

8.4 Experiments

Two experiments are performed on synthetic data, using a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM, MATLAB R2016b and Tensorlab 3.0 [228].

8.4.1 Experiment 1

We analyze the decrease in computation time that the exploitation of the rank-1 structure brings by computing rank-1 decompositions of 10 random rank-1 tensors $\mathcal{T} \in \mathbb{R}_{+}^{10\times10\times10\times10\times10\times10}$. The factors are sampled uniformly from $[0,1]$. We initialize with random factors with entries sampled uniformly from $[0,1]$ and perform 10 iterations of the general rank-R and the specialized rank-1 algorithms. In Figure 8.1, we see the time needed for the computation of the cost function, gradient and Hessian approximation for different values of β, averaged over the 10 tensors. One can see that the rank-1 algorithm requires about 2 to 6 times less computation time compared to the general β-divergence algorithm. The main improvements lie in the computation of the gradient, due to the fact that almost all factors can be reused from the objective function.

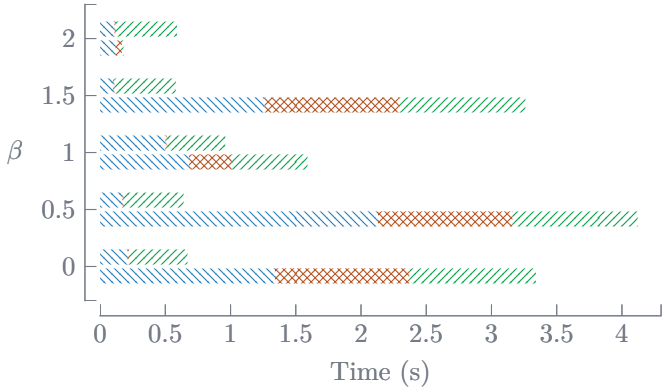

Figure 8.1: Time spent computing cost function (▧), gradient (▩) and Hessian approximation (▨) during 10 iterations of the NLS method with (top bars) and without (bottom bars) rank-1 exploitation. The rank-1 method offers a large speedup, except for the case $\beta = 2$ (LS cost function).

Except for $\beta = 1$ the computation of the cost function also becomes much more efficient, partly because the full tensor $\mathcal{M}$ never has to be formed explicitly. Of course, in this noiseless case, the optimal factor matrices are the same for any value of β. Thus in practice, one should use the least-squares algorithm instead, as it is still faster than the rank-1 β-divergence method.

8.4.2 Experiment 2

We compare the number of successful rank-1 approximations for three different methods: the proposed second-order rank-1 NLS method, the first-order generalized canonical polyadic decomposition (GCPD) method [105], and a first-order method using multiplicative updates (MU) [38]. A relative step size of $\epsilon = 2.22 \cdot 10^{-16}$ is used as a stopping criterion for the NLS and MU methods and a relative cost function improvement of ϵ is used as as stopping criterion for the GCPD method. The methods were additionally limited to 100 iterations, but this upper bound is only reached by the GCPD method for $\beta < 1$. We perturb 20 rank-1 tensors $\mathcal{T} \in \mathbb{R}_{+}^{50 \times 50 \times 50}$ with factor matrices sampled uniformly from $[0, 1]$ by multiplicative Gamma noise with an SNR of 40 dB. These tensors are approximated by a rank-1 tensor. We call an approximation successful if the relative error on the obtained factor matrices lies below 10^{-3}. In Table 8.1, we can see that the rank-1 NLS method and the MU algorithm find the solution in all cases, while the GCPD method fails for values of $\beta < 1$, where convexity of d_β

Table 8.1: Fraction of successful approximations, computation time and memory requirements for three rank-1 β-divergence approximation methods. The NLS algorithm is slightly faster than the MU algorithm and requires less storage space. The GCPD method generally performs badly for $\beta < 1$.

		β					
	Method	-0.5	0	0.5	1	1.5	2.5
Success rate	Rank-1 NLS	1	1	1	1	1	1
	MU	1	1	1	1	1	1
	GCPD	0	0.05	0.65	1	1	1
Time (s)	Rank-1 NLS	1.62	1.25	0.47	0.37	0.33	0.25
	MU	5.26	4.85	1.66	0.11	0.36	0.49
	GCPD	7.53	1.73	3.90	0.73	1.04	1.15
Memory	Rank-1 NLS	$\mathcal{O}(I)$ + tensor contractions					
	MU	$\mathcal{O}(I^3)$					
	GCPD	$\mathcal{O}(I^3)$					

is not guaranteed. The NLS algorithm is slightly faster than the MU algorithm. Further, for an $(I \times I \times I)$-tensor $\mathcal{T}$, the MU and GCPD algorithms require the formation of a full (dense) $(I \times I \times I)$-approximation tensor in every iteration, even when $\mathcal{T}$ is sparse. In contrast, the rank-1 NLS method only needs to store some vectors of length I and compute contractions with the original tensor $\mathcal{T}$.

8.5 Conclusion

As rank-1 approximations are often used in practice, we propose a dedicated algorithm to compute a rank-1 β-divergence approximation to a tensor. The algorithm uses a good approximation of the Hessian and thus offers fast local convergence, especially when starting from the LS solution, while it avoids the construction of large intermediate tensors and can exploit available structure in the tensor. In future work, we will look at other classes of cost functions and possible extensions to higher ranks.

Tensor-based blind fMRI source separation without the Gaussian noise assumption

9

ABSTRACT

The advantages of tensor- over matrix-based methods have been recently demonstrated in the context of functional magnetic resonance imaging (fMRI) blind source unmixing. However, these methods rely on the assumption of a Gaussian distribution for the noise, which suggests a least squares criterion for the tensor decomposition. One can instead argue that a Rician model for the fMRI noise is much more accurate and hence alternative cost functions should also be investigated. In this chapter, β-divergences are used to parametrize the Canonical Polyadic Decomposition (CPD) fitting to fMRI data and the effect of β on the source separation performance is evaluated, for different values of signal to noise ratio (SNR). Our results confirm that the commonly used squared error is not the best choice, particularly at low SNRs.

This chapter is based on C. Chatzichristos, M. Vandecappelle, E. Kofidis, S. Theodoridis, L. De Lathauwer, and S. Van Huffel, "Tensor-based Blind fMRI Source Separation Without the Gaussian Noise Assumption — A β-Divergence Approach," in 7th IEEE Global Conference on Signal and Information Processing (GlobalSIP) , Nov. 2019, pp. 1–5. The figures have been updated for consistency. The specific contributions of the doctoral candidate are listed in Section 11.1.

9.1 Introduction

Functional Magnetic Resonance Imaging (fMRI) is a noninvasive neuro-imaging technique, which indirectly studies brain activity, by measuring fluctuations of the Blood Oxygenation Level Dependent (BOLD) signal [130]. The fMRI concept builds upon the Magnetic Resonance Imaging (MRI) technology and the properties of oxygen-rich blood. The signal detected from the MR scanner is the sum of the signals from dipoles (nuclei) all over the region under examination. In order to form an image, the dipoles must somehow be spatially resolved. To this end, MRI uses strong magnetic fields (varying in phase and frequency) in order to define the different voxels from where the signal is transmitted [233].

The signal is measured through a quadrature detector that produces real and imaginary signals (based on the different phases and frequencies of the applied magnetic gradients). The collected raw complex-valued data, which lie in the frequency domain, form the so-called k-space. Each point in that space contains spatial frequency and phase information about all the voxels of the image. The noise that corrupts the data is white Gaussian with zero mean [164, 233].

The image space can thus be acquired through an Inverse Fourier Transform (IFT) of the k-space and, since the IFT is a linear transformation, the noise distribution in the image space is also Gaussian. Furthermore, the variance of the noise will be uniform over the whole field of view and, for a circularly symmetric k-space noise, the real and imaginary parts of the noise can be assumed uncorrelated [90].

However, MR scanners typically provide magnitude images, mainly due to memory limitations and also for avoiding the problem of phase artifacts (by deliberately discarding the phase information) [178, 219]. By only taking the magnitude of the complex data, the noise that results follows a Rician instead of a Gaussian model [90].

The localization of the activated brain areas from the fMRI data is a challenging Blind Source Separation (BSS) problem [202], in which the sources consist of a combination of spatial maps (areas activated) and time courses (timings of activation). fMRI data involve multiple modes, such as trial, session and subject, in addition to the intrinsic modes of time and space [6]. Up to recently, multivariate bi-linear (i.e., matrix-based) methods, based on the concatenation of different modes, have been the state-of-the-art in fMRI BSS [24,111]. However, by definition, such methods fall short in exploiting the inherently multi-way nature of fMRI data.

In contrast, this is preserved in multi-linear (tensor) models, which, in general, a) produce unique (modulo scaling and permutation ambiguities)

representations under mild conditions [176], b) can improve the ability of extracting spatiotemporal modes of interest [6, 98, 193], and c) facilitate neurophysiologically meaningful interpretations [6]. The state-of-the-art in tensorial methods for analyzing multi-subject fMRI data include (among others [11, 29, 138]) the Canonical Polyadic Decomposition (CPD)-based analysis [6], which views the fMRI data as a 3rd-order tensor, with modes space $\times$ time $\times$ subjects.

However, in all of those tensor-based blind fMRI source unmixing methods, a Gaussian approximation of the noise is used, which suggests the adoption of the least squares (LS) criterion for the estimation of the tensor decomposition, but is known to be inaccurate in view of the above. In this chapter, we relax the Gaussian assumption for the fMRI noise with the aid of β-divergence cost functions, and investigate whether, for certain types of data and specific squared noise distributions, these can lead to more meaningful low-rank approximations than those obtained with the LS cost function. This is done by simulating fMRI data, adding noise with different Signal-to-Noise Ratios (SNRs) and decomposing the resulting tensor with a β-divergence CPD for different values of β. It is shown that, indeed, the LS criterion can be outperformed by choosing a suitable β-divergence cost function. For overlapping sources, higher values of β work well, while for sources that have different noise variances in their real and imaginary parts, lower β values offer the best results.

9.1.1 Notation

Vectors, matrices and higher-order tensors are denoted by bold lower-case, upper-case and calligraphic upper-case letters, respectively. The outer product of two or more vectors is denoted by $\circ$.

9.2 Noise Model

The magnitude images are formed by calculating the magnitude, pixel by pixel, from the real and the imaginary images. This is a nonlinear mapping and therefore the resulting noise distribution is no longer Gaussian (and is signal dependent). Assuming that $A(p)$ is the MR image in the absence of noise (with $A_r(p)$ and $A_i(p)$ its real and imaginary parts, respectively) at voxel location p, the magnitude signal is:

$$M(p) = \sqrt{(A_r(p) + N_r(p))^2 + (A_i(p) + N_i(p))^2},$$

with N_r, $N_i \sim \mathcal{N}(0, \sigma^2)$, where σ denotes the standard deviation of the Gaussian noise in the real and the imaginary images. Hence, in the presence of noise, the probability density function (pdf) of the magnitude image is given by [90, 219]:

$$p(M) = \frac{M}{\sigma^2} e^{-\frac{M^2+A^2}{2\sigma^2}} I_0\left(\frac{AM}{\sigma^2}\right),$$

with M and A being the mean magnitude signal intensity and the mean measured signal intensity, respectively, and $I_0(\cdot)$ is the modified zeroth–order Bessel function of the first kind. This is known as the Rice density and for low SNRs it is quite different from the Gaussian pdf.

While the stationarity of the Rician noise (in the sense of having the same variance throughout space) has been the keystone of the statistical signal processing in MRI for years, may not hold when multiple coils and parallel imaging reconstruction, are considered [5]. Parallel imaging methods are used to increase the acquisition rate via subsampling of the k-space. In those methods, the known placement and sensitivities of the receiver coils are used to assist spatial localization of the MR signal, and hence a reduction in the number of phase-encoding steps during image acquisition is achieved. When parallel imaging techniques are applied, due to the use of fewer phase gradients, a reconstruction process is needed, for combining the signals from each individual coil into a single composite image, and accommodating for possible aliasing artifacts. As a result, the previous assumption of a single value of σ to characterize the whole data set is no longer valid, the variance of noise becomes space-dependent, and of different value for real and imaginary parts [17].

9.3 CPD with β-divergence

CPD (or PARAFAC) [6, 95, 174], approximates the 3rd-order (space × time × subject) tensor of fMRI data, $\mathcal{T} \in \mathbb{R}^{I_{xyz} \times I_t \times I_s}$, by a sum of R (= estimated number of sources) rank-1 tensors, namely

$$\mathcal{T} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r.$$

Letting $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_R]$ be the matrix that contains the R spatial components (I_{xyz} voxels per component) and defining $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_R]$ and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_R]$ as corresponding to the associated time courses (I_t time points) and the subject activation levels (I_s subjects), respectively, the CPD

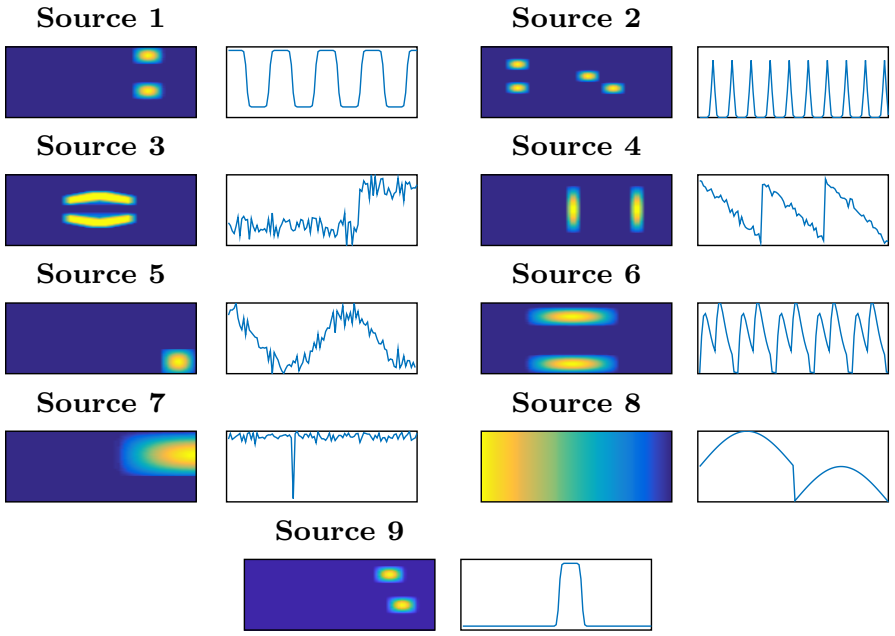

Figure 9.1: Magnitude spatial maps and time courses of the complex-valued sources used.

of $\mathcal{T}$, denoted by $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, allows us to recover the space-time sources in a completely blind manner.

Its main advantage, besides its simplicity, is the fact that it is unique (up to permutation and scaling) under mild conditions [58, 123]. Uniqueness of CPD is crucial to its application in fMRI. It was demonstrated [30, 98, 193] that CPD with fMRI data is robust to overlaps (spatial and/or temporal) of the sources. On the other hand, the result of CPD is largely dependent on the correct estimation of the tensor rank, R [22, 28].

Numerous algebraic and optimization-based algorithms have been proposed to estimate the CPD of a tensor [174]. Most of these methods fit a CPD to the tensor in a LS sense, i.e. by minimizing the Frobenius norm of the residual tensor $\mathcal{T} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$. Assuming that the tensor entries are perturbed by independent and identically distributed (i.i.d.) Gaussian noise, this gives the maximum likelihood estimate. However, magnitude images are instead expected to have a Rician noise distribution, as explained above. As a result, fitting a CPD with a different criterion may lead to better results for this type of data and investigating this possibility is the subject of this chapter.

For nonnegative data, such as magnitude images, a possible alternative to the squared error is the class of β-divergence cost functions, a special class of Bregman divergences [100], defined for $\beta \in \mathbb{R}$. This class of cost functions encompasses not only the squared error itself ($\beta = 2$), but also the Itakura—Saito (IS) divergence ($\beta = 0$) and the Kullback—Leibler (KL) divergence ($\beta = 1$) and interpolates between them continuously. An important property of β-divergence cost functions is that values of $\beta < 2$ penalize errors in small entries more heavily compared to the squared distance. On the other hand, for values of $\beta > 2$, errors in large entries are penalized more heavily. β-divergences can therefore be a good choice when the tensor has regions of significantly different magnitudes and one is primarily interested in fitting either the smaller or larger entries better compared to the LS approach. These cost functions have been applied successfully to the decomposition of tensors containing count data or spectral data and gave more meaningful low-rank approximations than those obtained with the squared cost [79]. The KL divergence has been demonstrated to outperform the squared distance as a criterion for different model choices in matrix- and general linear model-based fMRI analysis [70, 125, 232].

For each tensor entry x, the β-divergence $d_\beta(x, y)$ between x and its estimate y from the CPD is defined as follows:

$$d_\beta(x,y) = \begin{cases} \frac{x^\beta + (\beta-1)y^\beta - \beta(xy^{(\beta-1)})}{\beta(\beta-1)} & \beta \in \mathbb{R}\backslash\{0,1\} \\ x\log(\frac{x}{y}) - x + y & \beta = 1 \\ \frac{x}{y} - \log(\frac{x}{y}) - 1 & \beta = 0 \end{cases}$$

Note that the squared distance depends only on the absolute difference between x and y, while other β-divergences also consider the values x and y themselves. A number of methods have been proposed to compute a nonnegative (which is relevant in our problem) CPD of a tensor with a β-divergence cost function [38, 80, 105]. In this chapter, we will use the algorithm from [214], which uses second-order information to compute a nonnegative CPD of a tensor for general cost functions and thus also for the β-divergence. Following a generalized Gauss–Newton approach, the algorithm uses curvature information from the Hessian matrix of the cost function and thus offers fast local convergence, while it exploits any available structure in the tensor.

9.4 Results

In order to test the performance of the separation algorithm for Rician noise, complex-valued data were generated, as the Rician noise is obtained from the

modulus of complex data that are perturbed by Gaussian noise. A set of 8 complex spatial components was generated and mixed with a set of time courses to obtain a simulated fMRI-like dataset [25]. The sources are generated based on basic knowledge of the statistical characteristics of the underlying neuronal sources.

The complex-valued spatial maps are of 60×60 voxels each and were simulated along with their corresponding time courses of 100 samples each. The magnitude spatial maps and the time courses of each source are as shown in Figure 9.1. In an fMRI experiment, the phase difference between activated and resting states of a voxel, induced by the task activation, is typically less than $\pi/9$ [106]. Therefore, we keep the phase of each pixel uniformly distributed in the range $[-\pi/18, \pi/18]$. The phase of each complex-valued time point is generated proportional to its magnitude [233], but is again restricted to the range mentioned. The spatial sources are unfolded into one-dimensional vectors and mixed by the corresponding time courses. The same procedure is followed for S different subjects, and for each subject every source occurs with different random (real) amplitude. A tensor $\mathcal{T} \in \mathbb{R}^{3600 \times 100 \times S}$ with complex-valued entries is thus generated. Gaussian complex noise is added and the CPD of the magnitude (image) tensor $\tilde{\mathcal{T}}$ provides the source estimates.

Three different simulation setups are tested. The first with the initial eight sources used in [25], the second with the same sources but with noise with different standard deviation in the real and imaginary parts (simulating the result of parallel imaging methods) and the last one with an extra source (Source 9, Figure 9.1), which has high spatial and temporal overlap with Source 1. Four different SNRs, ranging from low to high values, are considered in each case, namely: 95.4 dB, 47.6 dB, 22.4 dB and 11.2 dB, with noise variance equal to 0.5. The rank, R, was set equal to the number of sources for all the simulation setups. The Pearson correlation of the obtained sources with the ground truth is presented in Figures 9.2, 9.3 and 9.4 for the three different scenarios, respectively.

As we can note from Figure 9.2, where only sources with low overlap are considered, the best β value is equal to 2.5 and not the commonly used $\beta = 2$ (LS) case. In the high SNR case, for different β values, the results are approximately the same while at lower SNR values the difference between the performances with $\beta = 2.5$ and $\beta = 2$ increases. In Figure 9.5 (left), a histogram is shown of observed values for one spatial point with a constant time course, where both the real and imaginary parts have been perturbed by Gaussian distributed noise with equal variance. The Rician nature of the data is apparent.

In the second case (Figure 9.3), with different noise variance in the real and imaginary parts, we note that $\beta = 1$ (KL divergence) gives the best performance,

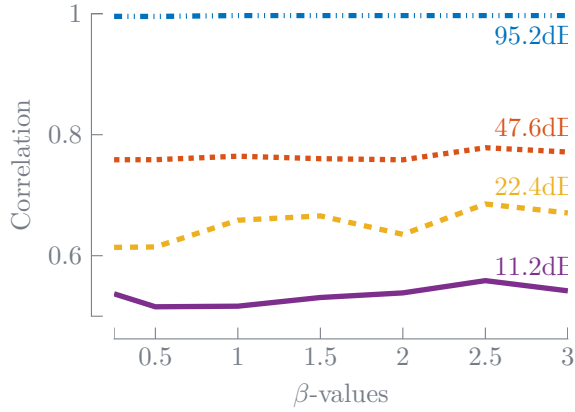

Figure 9.2: First simulation with the 8 sources used in [25].

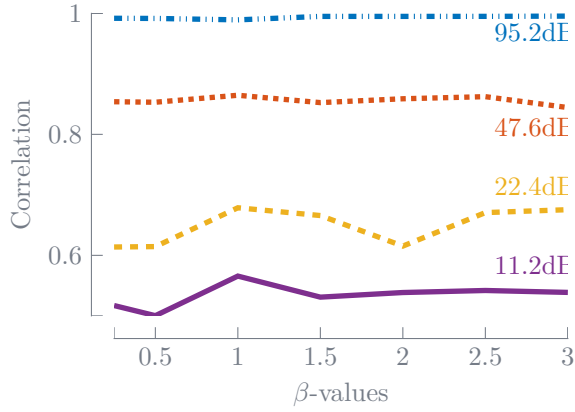

Figure 9.3: Second simulation with the 8 sources used in [25] and different noise variance in the real and imaginary parts of the complex-valued source signals.

while also $\beta = 2.5$ still has better performance than $\beta = 2$. In Figure 9.5 (right), a histogram is shown for one spatial point where the imaginary part has been perturbed by Gaussian distributed noise with five times the variance of the noise of the real part. Most entries are very small, but the right tail of the distribution is long, pointing to a large number of outliers.

Last, when an extra source with high spatial and temporal overlap is included in the first scenario (Source 9 in Figure 9.1), the gain in performance with

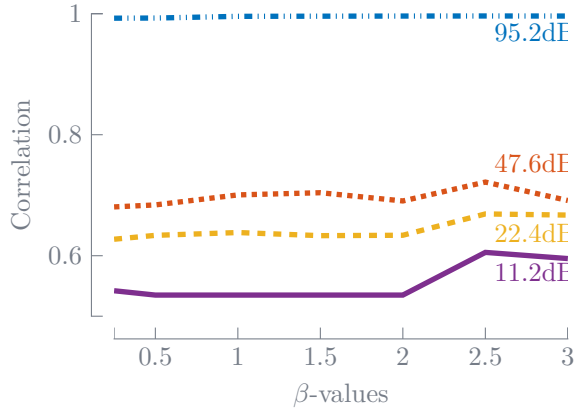

Figure 9.4: Third simulation with 9 sources, where Source 9 with high overlap is included in the scenario of Figure 9.1.

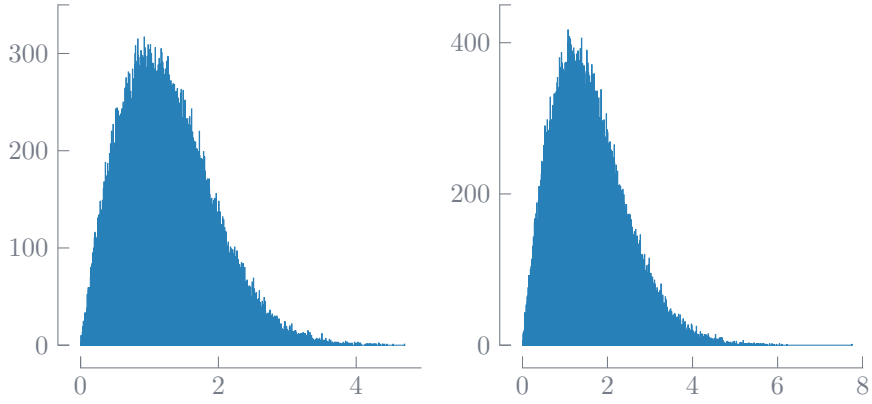

Figure 9.5: Histograms of the observed values at one spatial point. Left: real and imaginary noise variances are equal. Right: the noise variance of the imaginary part is five times that of the real part.

$\beta=2.5$ (and $\beta=3$) compared to $\beta=2$ increases at low SNR values, as can be noted in Figure 9.4. The overlapping sources result in an increase of the signal magnitude in the specific area, hence these values are fitted relatively better than the non-overlapping ones. As mentioned before, for $\beta<2$, errors in smaller entries are penalized more heavily than for the squared distance, while for $\beta>2$ the converse is true. Hence in the case of overlapping sources, more importance is given to fitting these entries well compared to the squared distance [147].

9.5 Conclusion

To the best of the authors' knowledge, this is the first time that the Gaussian noise assumption and its influence on the fMRI BSS performance are tested in a tensorial framework. Given that the use of a LS criterion (implied by the Gaussian assumption) for fitting a CPD to fMRI data is not optimal in the presence of Rician noise, we have considered alternative cost functions in this chapter, based on β-divergences. Different values of β have been tested and it has been shown that $\beta=1$ (KL divergence) performs best in cases where different noise variances affect the real and imaginary data. Furthermore, $\beta=2.5$ outperforms the LS criterion ($\beta=2$) by producing better separation results in all other cases.

As future work, real data will be also considered, while additional constraints that are meaningful for fMRI BSS will be tested as well. For example, the adoption of Block Term Decomposition [30,44] (BTD) and the use of regularizers that force independence or sparsity in the spatial maps will be investigated.

A recursive eigenspace computation for the canonical polyadic decomposition

10

ABSTRACT

The canonical polyadic decomposition (CPD) is a compact decomposition which expresses a tensor as a sum of its rank-1 components. A common step in the computation of a CPD is computing a generalized eigenvalue decomposition (GEVD) of the tensor. A GEVD provides an algebraic approximation of the CPD which can then be used as an initialization in optimization routines.

While in the noiseless setting GEVD exactly recovers the CPD, it has recently been shown that pencil based computations such as GEVD are not stable. In this chapter we present an algebraic method for approximation of a CPD which greatly improves on the accuracy of GEVD. Our method is still fundamentally pencil based; however, rather than using a single pencil and computing all of its generalized eigenvectors, we use many different pencils and in each pencil compute generalized eigenspaces corresponding to sufficiently well separated generalized eigenvalues. The resulting 'generalized eigenspace decomposition' is significantly more robust to noise than the classical GEVD.

Accuracy of the generalized eigenspace decomposition is examined both empirically and theoretically. In particular, we provide a deterministic perturbation theoretic bound which is predictive of the error in the computed factorization

10.1 Introduction

Tensors, or multiindexed arrays, play an important role in fields such as machine learning and signal processing [35, 174]. These higher-order generalizations of matrices allow for preservation of higher-order structure present in data and

This chapter is based on E. Evert, M. Vandecappelle, and L. De Lathauwer, "A recursive eigenspace computation for the canonical polyadic decomposition," submitted to SIAM Journal on Matrix Analysis and Applications. The specific contributions of the doctoral candidate are listed in Section 11.1.

low rank decompositions of tensors allow for compression of data and recovery of underlying information [35,39,174]. One of the most popular decompositions for tensors is the canonical polyadic decomposition (CPD) which expresses a tensor as a sum of rank-1 tensors. An important feature of the CPD is that, with mild assumptions [57,123,176], the CPD of a low rank tensor is unique. Furthermore, for a low rank tensor with a unique CPD, the CPD can often be found algebraically. Such an algebraic solution can typically be obtained with limited computation time, hence is often used as an initialization for optimization based methods when the tensor is noisy.

One of the most popular algorithms for algebraic computation of a CPD is the generalized eigenvalue decomposition (GEVD) [60,62,127,168]. The key idea behind the classical GEVD is that a factor matrix of a tensor may be obtained by computing the generalized eigenvectors of any subpencil of the tensor; see Algorithm 2.1. However, it has recently been shown that pencil based algorithms such as GEVD are not stable [13]. That is, the condition number for computing a generalized eigenvalue decomposition of a subpencil can be arbitrarily larger than the condition number [19] for computing the CPD of the tensor. In this chapter we present an extension of the GEVD algorithm which significantly improves the accuracy of algebraic computation of the CPD. This increase in accuracy is obtained by using many subpencils of the tensor to compute the CPD. In each subpencil we only compute generalized eigenvectors and eigenspaces which correspond to sufficiently well separated generalized eigenvalues.

The ability to accurately compute generalized eigenvectors of a matrix pencil is heavily dependent on the separation between the generalized eigenvalues and the generalized eigenvectors of the pencil [86,148,198]. In the case that the generalized eigenvalues and eigenvectors are well separated, a GEVD may be accurately computed. However, when either a pair of generalized eigenvalues or generalized eigenvectors is near parallel, the accuracy of the computed generalized eigenvectors deteriorates. As such, the GEVD algorithm performs well when there is a subpencil for the tensor in which all generalized eigenvalues are well separated; however, GEVD runs into challenges if one is unable to find a subpencil in which all generalized eigenvalues are well separated.

When computing a GEVD of a rank-R tensor, one computes the generalized eigenvalues and eigenvectors of a pair of $R \times R$ matrices. The generalized eigenvalues of this matrix pair can be naturally interpreted as a collection of R points on the unit circle [198], so it is not difficult to see that the generalized eigenvalues cannot be well separated when the tensor rank R is very large.

We propose an alternative approach in which we exploit the fact that a tensor has many subpencils, and that a factor matrix of the tensor may be computed

by combining information from many different subpencils. Using the language of the joint generalized eigenvectors (JGE vectors) [67], the primary goal of the GEVD algorithm is in fact to compute the JGE vectors of the tensor.

Let $\mathbb{K}=\mathbb{R}$ or $\mathbb{C}$. Suppose the tensor $\mathcal{T}\in\mathbb{K}^{R\times R\times K}$ is a rank-R tensor with JGE vectors $\{\mathbf{x}_r\}_{r=1}^{R}$. Then for every subpencil of $\mathcal{T}$, the generalized eigenspaces are of the form $\operatorname{span}(\{\mathbf{x}_{r_1},\dots,\mathbf{x}_{r_k}\})$ for some subcollection of JGE vectors of $\mathcal{T}$. Here $\{r_1,\dots,r_k\}$ is a subcollection of indices from $\{1,\dots,R\}$ with cardinality k. Intuitively it follows that one may compute the JGE vectors of $\mathcal{T}$ by computing generalized eigenspaces for many different subpencils of $\mathcal{T}$.

An important feature of this approach is that the accuracy of computing a particular generalized eigenspace does not depend on the separation of the generalized eigenvalues which correspond to that generalized eigenspace. Rather the accuracy depends on the separation of the set of generalized eigenvalues corresonding to the generalized eigenspace of interest from the other generalized eigenvalues of the pencil as well as the separation between the generalized eigenspace itself and the other generalized eigenvectors of the pencil [196]. Thus by only computing generalized eigenspaces which correspond to collections of generalized eigenvalues that are well separated from the other generalized eigenvalues of the pencil, one source of inaccuracy for the generalized eigenvalue decomposition, i.e., inaccuracy introduced by projecting a tensor to a matrix pencil [13], is significantly reduced.

10.1.1 Notation and definitions

We denote scalars, vectors, matrices, and tensors by lower case (a), bold lower case ($\mathbf{a}$), bold upper case ($\mathbf{A}$), and calligraphic script ($\mathcal{A}$), respectively. Additionally, fraktur font ($\mathfrak{A}$) is used to denote a subspace of $\mathbb{K}^I$. For a matrix $\mathbf{A}\in\mathbb{K}^{I\times I}$, we let $\mathbf{A}^{\mathrm{T}}$ denote the transpose of $\mathbf{A}$ while $\mathbf{A}^{\mathrm{H}}$ denotes the conjugate transpose of $\mathbf{A}$. If $\mathbf{A}$ is invertible, then we let $\mathbf{A}^{-\mathrm{T}}$ denote the inverse of $\mathbf{A}^{\mathrm{T}}$. A tensor is a multiindexed array with entries in $\mathbb{K}$. In this chapter we restrict our attention to tensors of order three. In other words, the tensors we consider are elements of $\mathbb{K}^{I_1\times I_2\times I_3}$.

Given nonzero vectors $\mathbf{a}\in\mathbb{K}^{I_1},\mathbf{b}\in\mathbb{K}^{I_2},\mathbf{c}\in\mathbb{K}^{I_3}$, let

$$\mathbf{a}\otimes\mathbf{b}\otimes\mathbf{c}\in\mathbb{K}^{I_1\times I_2\times I_3}$$

denote the $I_1\times I_2\times I_3$ tensor with i,j,k entry equal to $a_ib_jc_k$. A tensor of this form is called a rank-1 tensor. The minimal integer R such that $\mathcal{M}\in\mathbb{K}^{I_1\times I_2\times I_3}$

may be written as

$$\mathcal{M} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r$$

where the vectors $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$ have entries in $\mathbb{K}$ is called the $\mathbb{K}$-rank of the tensor $\mathcal{M}$, and a decomposition of this form is called a canonical polyadic decomposition (CPD) of $\mathcal{M}$. Compactly we write $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ where the matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ have $\mathbf{a}_r, \mathbf{b}_r$ and $\mathbf{c}_r$ as their rth columns, respectively. The matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are called the factor matrices for $\mathcal{M}$.

Given a tensor $\mathcal{M} \in \mathbb{K}^{I_1 \times I_2 \times I_3}$, let $\mathbf{M}_{[1;3,2]}$ denote an $I_1 \times I_2 I_3$ matrix with columns $\{\mathbf{m}(:,k,\ell)\}_{\forall k,\ell}$. Here, the columns of $\mathbf{M}_{[1;3,2]}$ are ordered so that the second index of $\mathbf{m}(:,k,\ell)$ increments faster than the third. The matrices $\mathbf{M}_{[2;3,1]}$ and $\mathbf{M}_{[3;2,1]}$ are defined similarly. A matrix of the form $\mathbf{M}_{[i;k,j]}$ is called a mode-i unfolding of $\mathcal{M}$.

Having defined the mode-i unfoldings of $\mathcal{M}$, we define

$$R_i(\mathcal{M}) := \operatorname{rank}(\mathbf{M}_{[i;k,j]}) \qquad \text{for } i = 1,2,3.$$

The integer triple $(R_1(\mathcal{M}), R_2(\mathcal{M}), R_3(\mathcal{M}))$ is called the multilinear rank of $\mathcal{M}$.

For tensors $\mathcal{M}$ occurring in applications, $\mathcal{M}$ is often thought of as a noisy measurement of some signal tensor of interest. That is, one can decompose $\mathcal{M} = \mathcal{T}' + \mathcal{N}'$ where $\mathcal{T}'$ is the true signal tensor of interest and $\mathcal{N}'$ is measurement error. Most commonly the signal tensor $\mathcal{T}'$ has low multilinear rank. That is, one has $R_i(\mathcal{T}') \ll I_i$ for each i. In this case, under an appropriate unitary change of bases one has $t'_{i,j,k} = 0$ if $i > R_1(\mathcal{T}')$ or $j > R_2(\mathcal{T}')$ or $k > R_3(\mathcal{T}')$. In this choice of bases it is clear that $\mathcal{T}'$ may be compressed to a tensor of size $R_1(\mathcal{T}') \times R_2(\mathcal{T}') \times R_3(\mathcal{T}')$ without loss of information. Such a compression is called an orthogonal compression of $\mathcal{T}'$ to its multilinear rank, and a popular choice of orthogonal compression is the multilinear singular value decomposition (MLSVD); see [47] for details.

Compression of a noisy measurement $\mathcal{M} = \mathcal{T}' + \mathcal{N}'$ to the multilinear rank of $\mathcal{T}'$ is a common step in computation of a CPD which both approximates the (core of) the signal tensor $\mathcal{T}'$ and reduces computational complexity. However, such a compression has very few implications for the methods discussed in this chapter. For this reason we often restrict to considering tensors $\mathcal{Y}$ with the property $R_i(\mathcal{Y}) = I_i$ for $i = 1,2,3$. For clarity, here I_i refers to the ith dimension of the compressed tensor $\mathcal{Y}$. Intuitively, the symbol $\mathcal{Y}$ is used to denote a tensor which can be thought of as a compression of $\mathcal{M} = \mathcal{T}' + \mathcal{N}'$ to the multilinear rank of $\mathcal{T}'$. In particular, we may write $\mathcal{Y} = \mathcal{T} + \mathcal{N}$ where $\mathcal{T}$ is an orthogonal compression $\mathcal{T}'$ to its multilinear rank and $\mathcal{N}$ is noise.

It is not difficult to show that the $\mathbb{K}$-rank of a tensor $\mathcal{Y}$ is bounded from below by each of the $R_i(\mathcal{Y})$, e.g., see [124]. For most tensors the inequality is strict. However, for the signal portion $\mathcal{T}$ of a tensor $\mathcal{Y}$ occurring in applications, one often has (up to a permutation of indices) $\text{rank}_{\mathbb{K}}(\mathcal{T}) = R_1(\mathcal{T}) = R_2(\mathcal{T}) \geq R_3(\mathcal{T})$. In light of this, the tensors $\mathcal{Y} = \mathcal{T} + \mathcal{N}$ we consider are taken to have size $R \times R \times K$ where $K \leq R$ and the signal portion $\mathcal{T}$ of $\mathcal{Y}$ is assumed to have $\mathbb{K}$-rank R. To summarize, throughout the chapter we let $\mathcal{M}$ denote an arbitrary tensor of size $I_1 \times I_2 \times I_3$, we let $\mathcal{Y}$ denote a full multilinear rank tensor of size $R \times R \times K$, and we let $\mathcal{T}$ denote a $R \times R \times K$ tensor with $\mathbb{K}$-rank equal to R and full multilinear rank.

Define the mode-1 product between a tensor $\mathcal{M} \in \mathbb{K}^{I_1 \times I_2 \times I_3}$ and a matrix $\mathbf{U} \in \mathbb{K}^{J_1 \times I_1}$ to be the tensor $\mathcal{M} \cdot_1 \mathbf{U} \in \mathbb{K}^{J_1 \times I_2 \times I_3}$ with mode-1 fibers given by

$$(\mathcal{M} \cdot_1 \mathbf{U})(:, k, \ell) = \mathbf{U} \cdot \mathbf{m}(:, k, \ell) \qquad \text{for all } k, \ell.$$

The mode-2 and mode-3 products are defined analogously. Say a tensor $\mathcal{Y} \in \mathbb{K}^{R \times R \times K}$ is slice mix invertible if there is a vector $\mathbf{v} \in \mathbb{R}^K$ such that the matrix $\mathcal{Y} \cdot_3 \mathbf{v}^{\mathrm{T}}$ is invertible. That is, $\mathcal{Y}$ is slice mix invertible if there exists a linear combination of the matrices $\mathbf{Y}(:, :, k)$ for $k = 1, \ldots, K$ which is invertible. We call the matrices $\mathbf{Y}(:, :, k)$ the frontal slices of $\mathcal{Y}$ and use $\mathbf{Y}_k$ to denote $\mathbf{Y}(:, :, k)$ for $k = 1, \ldots, K$.

Let $\mathcal{Y} \in \mathbb{K}^{R \times R \times K}$. An implicit subpencil of $\mathcal{Y}$ is a tensor of the form $\mathcal{Y} \cdot_3 \mathbf{V} \in \mathbb{K}^{R \times R \times 2}$ where $\mathbf{V} \in \mathbb{K}^{2 \times K}$ has rank 2. More generally, call a tensor of the form $\mathcal{Y} \cdot_3 \mathbf{V} \in \mathbb{K}^{R \times R \times L}$ where $\mathbf{V} \in \mathbb{K}^{L \times K}$ has full row rank an implicit subtensor of $\mathcal{Y}$. For emphasis we note that implicit subpencils and implicit subtensors differ from the usual notions in that they can be revealed by modal multiplication in the third mode of the tensor. Throughout the chapter, implicit subtensors and subpencils of a given tensor are denoted using the calligraphic letter $\mathcal{S}$. In the reminder of the chapter we drop the distinction 'implicit', as all subpencils and subtensors we consider are implicit.

10.1.2 Joint generalized eigenvalues and eigenvectors

Given a tensor $\mathcal{T} \in \mathbb{K}^{R \times R \times K}$, we say a nonzero vector $\mathbf{z} \in \mathbb{K}^R$ is a joint generalized eigenvector (JGE vector) of $\mathcal{T}$ if there exists a $\boldsymbol{\lambda} \in \mathbb{K}^K$ and a nonzero vector $\mathbf{q} \in \mathbb{K}^R$ such that

$$\mathbf{T}_k \mathbf{z} = \lambda_k \mathbf{q} \tag{10.1}$$

for all $k = 1, \ldots, K$. Note that assuming $\mathbf{q} \neq \mathbf{0}$ guarantees that $\lambda_k = 0$ if and only if $\mathbf{T}_k \mathbf{z} = \mathbf{0}$. If $(\boldsymbol{\lambda}, \mathbf{z})$ is a solution to (10.1), then the one-dimensional subspace $\mathfrak{L} := \text{span}(\boldsymbol{\lambda}) \subset \mathbb{K}^K$ is a joint generalized eigenvalue (JGE value) of $\mathcal{T}$ and $(\mathfrak{L}, \mathbf{z})$

is a joint generalized eigenpair (JGE pair) of $\mathcal{T}$.[1] Throughout the chapter we make frequent use of the following theorem [67, Theorem 2.3 (1)] which explains the connection between JGE vectors and tensor rank.

Theorem 10.1.1:
Let $\mathcal{T}' \in \mathbb{K}^{I_1 \times I_2 \times I_3}$ and assume $\mathcal{T}'$ has multilinear rank (R, R, K). Let $\mathcal{T} \in \mathbb{K}^{R \times R \times K}$ be an orthogonal compression of $\mathcal{T}'$. Then $\mathcal{T}'$ has $\mathbb{K}$-rank R if and only if the JGE vectors of $\mathcal{T}$ form a basis for $\mathbb{K}^R$. Furthermore if $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ has $\mathbb{K}$-rank R then the JGE values of $\mathcal{T}$ are equal to the spans of columns of the factor matrix $\mathbf{C}$, and the JGE vectors of $\mathcal{T}$ are given by the columns of $\mathbf{B}^{-\mathrm{T}}$.

Given a subspace $\mathfrak{Z} \subset \mathbb{K}^R$ of dimension ℓ and a tensor $\mathcal{T} \in \mathbb{K}^{R \times R \times K}$, say $\mathfrak{Z}$ is a joint generalized eigenspace of $\mathcal{T}$ if there is a subspace $\mathfrak{Q} \subset \mathbb{K}^R$ of dimension less than or equal to ℓ such that

$$\mathbf{T}_k \mathfrak{Z} \subseteq \mathfrak{Q} \qquad \text{for all } k = 1, \ldots, K.$$

Here $\mathbf{T}_k \mathfrak{Z}$ denotes the subspace $\{\mathbf{T}_k \mathbf{z} : \mathbf{z} \in \mathfrak{Z}\} \subset \mathbb{K}^R$. In the case that $K = 2$ we simply call $\mathfrak{Z}$ a generalized eigenspace.[2] Given a JGE value $\mathfrak{L} \subset \mathbb{K}^K$ of a tensor $\mathcal{T} \in \mathbb{K}^{R \times R \times K}$, let $\mathfrak{Z}_{\mathfrak{L}} \subset \mathbb{K}^R$ be the subspace of joint generalized eigenvectors corresponding to $\mathfrak{L}$. One may easily show that $\mathfrak{Z}_{\mathfrak{L}}$ is a joint generalized eigenspace of $\mathcal{T}$.

Generalized eigenspaces of subtensors

The main goal of this chapter is to improve the accuracy of algebraic (pencil based) methods for computing a CPD. Key observations which allow us to accomplish this are that generalized subspaces may often be more stably computed than individual generalized eigenvectors and that the joint generalized eigenvectors of a tensor may be recovered by computing generalized eigenspaces for sufficiently many subtensors. The following proposition helps formalize this idea.

Proposition 10.1.1:
Let $\mathcal{T} \in \mathbb{K}^{R \times R \times K}$ be a tensor of $\mathbb{K}$-rank R and let $\mathbf{V} \in \mathbb{K}^{K \times L}$ be a matrix with full row rank. Set $\mathcal{S} = \mathcal{T} \cdot_3 \mathbf{V}^{\mathrm{T}}$, assume $\mathcal{S}$ is slice mix invertible, and let $\mathfrak{L}_{\mathcal{S}}$ be a JGE value of $\mathcal{S}$. There exist JGE values $\mathfrak{L}_1, \ldots, \mathfrak{L}_J$ of $\mathcal{T}$ such that

$$\mathfrak{Z}_{\mathfrak{L}_{\mathcal{S}}} = \operatorname{span}(\{\mathfrak{Z}_{\mathfrak{L}_j}\}_{j=1}^{J}).$$

[1] If $K = 2$, then joint generalized eigenvectors and joint generalized eigenvalues are exactly equal to classical generalized eigenvalues and generalized eigenvectors (in the sense of [198]).

[2] Generalized eigenspaces are sometimes called deflating subspaces by other authors.

Proof. First note that assuming $\mathcal{S}$ is slice mix invertible implies $\mathcal{T}$ is slice mix invertible. Now, following the proof of [67, Proposition 5.1], one finds that each joint generalized eigenvalue of $\mathcal{S}$ has the form $\mathbf{V}^{\mathrm{T}}\mathfrak{L}$ for some JGE value $\mathfrak{L}$ of $\mathcal{T}$. Note that one must have $\mathbf{V}^{\mathrm{T}}\mathfrak{L} \neq \mathbf{0}$ as otherwise Theorem 10.1.1 would imply that $\mathcal{S}$ has rank less than R, contradicting the assumption that $\mathcal{S}$ is slice mix invertible. Given some JGE value $\mathfrak{L}_{\mathcal{S}}$ of $\mathcal{S}$, let $\mathfrak{L}_1, \ldots, \mathfrak{L}_J$ be those joint generalized eigenvalues of $\mathcal{T}$ such that $\mathbf{V}^{\mathrm{T}}\mathfrak{L}_j = \mathfrak{L}_{\mathcal{S}}$. It is straightforward to check that

$$\operatorname{span}(\{\mathfrak{Z}_{\mathfrak{L}_j}\}_{j=1}^{J}) \subset \mathfrak{Z}_{\mathfrak{L}_{\mathcal{S}}}. \tag{10.2}$$

To prove the reverse containment, use the assumption that $\mathcal{S}$ is slice mix invertible and the fact that $\mathcal{T}$ has rank R to conclude $\mathcal{S}$ has rank R. Letting $\{\mathfrak{L}_{\mathcal{S},1}, \ldots, \mathfrak{L}_{\mathcal{S},\ell}\}$ be the set of JGE values of $\mathcal{S}$, it follows from Theorem 10.1.1 together with [67, Lemma 2.2] that the collection of spaces $\{\mathfrak{Z}_{\mathfrak{L}_{\mathcal{S},i}}\}_{i=1}^{\ell}$ spans $\mathbb{K}^R$ and also that these spaces pairwise have trivial intersection. Again using 10.1.1 shows that the JGE vectors of $\mathcal{T}$ span $\mathbb{K}^R$. Thus, since each JGE vector of $\mathcal{T}$ lies in some $\mathfrak{Z}_{\mathfrak{L}_{\mathcal{S},i}}$, the containment in Equation (10.2) must in fact be an equality. □

Proposition 10.1.1 has the implication that one may, for example, compute joint generalized eigenvectors of $\mathcal{T}$ by computing intersections of generalized eigenspaces of various different subpencils of $\mathcal{T}$. In this approach one must compute enough generalized eigenspaces so that all individual generalized eigenvectors may be recovered by taking intersections of generalized eigenspaces.

The generalized eigenvalue decomposition for CPD computation

We now briefly recall the main idea behind the GEVD algorithm for CPD computation. Suppose $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \in \mathbb{R}^{R \times R \times K}$ has rank R tensor and is slice mix invertible, and let $\mathbf{V}^{\mathrm{T}} \in K \times 2$ be a generic matrix. Additionally set $\mathcal{S} = \mathcal{T} \cdot_3 \mathbf{V} = [\![\mathbf{A}, \mathbf{B}, \mathbf{V}^{\mathrm{T}}\mathbf{C}]\!]$. Since $\mathbf{V}$ is generic, the tensor $\mathcal{S}$ has rank R and is slice mix invertible. Additionally, the column rank of $\mathbf{C}$ is the same as that of $\mathbf{V}^{\mathrm{T}}\mathbf{C}$. It follows, e.g. from Theorem 10.1.1, that the factor matrix $\mathbf{B}$ of $\mathcal{T}$ can be computed by computing a generalized eigenvalue decomposition of $\mathcal{S}$. Having solved for $\mathbf{B}$, one may recover $\mathbf{A}$ and $\mathbf{C}$ by solving for $\mathbf{C} \odot \mathbf{A}$ in

$$\mathbf{T}_{[2;3,1]} = \mathbf{B}(\mathbf{C} \odot \mathbf{A})^{\mathrm{T}},$$

then computing rank-1 decompositions of the columns of $\mathbf{C} \odot \mathbf{A}$.

The key difficulty in this algorithm that we aim to address is that the compression step which maps $\mathcal{T}$ to $\mathcal{S}=\mathcal{T}\cdot\mathbf{V}^{\mathrm{T}}$ is not stable [13]. Intuitively, this is related to the fact that the separation between the columns of $\mathbf{V}^{\mathrm{T}}\mathbf{C}$ is expected to be much smaller than the separation between the columns of $\mathbf{C}$. Indeed when one projects a collection of vectors in $\mathbb{K}^K$ to $\mathbb{K}^2$, the angle between those vectors cannot increase and is expected to decrease[3].

The original GEVD algorithm was communicated in 1970 by Harshman who credits the algorithm to Jennrich [95]. We note that aside from GEVD, there are several popular semi-algebraic methods for computing the CPD of a tensor, e.g., see [43, 160, 163].

10.1.3 Guide to the reader

Section 10.2 explains the generalized eigenspace decomposition (GESD) algorithm for computation of a canonical polyadic decomposition. In Section 10.3, we give a bound that is predictive for the maximal noise level that does not inhibit accurate decomposition. In particular, we give a deterministic bound using perturbation theory for JGE values. Sections 10.4 and 10.5 present numerical experiments for the GESD algorithm. In Section 10.4 we compare the performance (in terms of timing and accuracy) of GESD to the classical GEVD algorithm. In Section 10.5 we compare the accuracy of the GESD algorithm to the bound computed in Section and 10.3.

10.2 The GESD algorithm

We now give a formal description of the GESD algorithm for the computation of a canonical polyadic decomposition. The main step of the algorithm is to use generalized eigenspaces to recursively split a given tensor into subtensors of reduced rank until the subtensors all have rank 1 (up to noise). The CPD of each (noisy) rank-1 tensor may be computed by a best rank-1 approximation, and the CPD for the full tensor is obtained by combining the decompositions of these rank-1 tensors.

Let $\mathcal{T}$ be an $R\times R\times K$ slice mix invertible tensor of rank R with CPD $\mathcal{T}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$. Let $\mathbf{V}\in\mathbb{K}^{K\times 2}$ be a column-wise orthonormal matrix[4] of size $K\times 2$

[3] In fact, if one projects a collection of vectors from $\mathbb{R}^K$ to $\mathbb{R}^2$, then the angle between some pair of vectors in the collection must decrease unless the collection is contained in a two dimensional subspace of $\mathbb{R}^K$.

[4] Column-wise orthonormal matrices are used for numerical reasons.

and set $\mathcal{S} = \mathcal{T} \cdot_3 \mathbf{V}^{\mathrm{T}}$. The GESD algorithm begins by computing a collection of generalized eigenspaces $\mathfrak{Z}_1, \ldots, \mathfrak{Z}_L$ for the generalized eigenvalue problem

$$\mathbf{S}_i \mathbf{z} = \boldsymbol{\lambda}_i \mathbf{q} \text{ for } i = 1, 2$$

such that $\operatorname{span}(\{\mathfrak{Z}_\ell\}_{\ell=1}^{L}) = \mathbb{K}^R$. In each step the GESD algorithm computes as many well separated eigenspaces as possible; however, to simplify exposition we will take $L = 2$.

Say the generalized eigenvalues of $\mathcal{S}$ are split into two distinct groups of size j_1 and j_2 where $j_1 + j_2 = R$. The corresponding eigenspaces may then for instance be computed with a **QZ** decomposition[5] of the matrix pencil $(\mathbf{S}_1, \mathbf{S}_2)$ [26]. In particular, the **QZ** decomposition computes unitary matrices $\mathbf{Q}, \mathbf{Z} \in \mathbb{K}^R$ such that

$$\mathbf{S}_1 = \mathbf{Q}\mathbf{X}_1\mathbf{Z}^{\mathrm{H}} \qquad \text{and} \qquad \mathbf{S}_2 = \mathbf{Q}\mathbf{X}_2\mathbf{Z}^{\mathrm{H}}$$

where $\mathbf{X}_1$ and $\mathbf{X}_2$ are (quasi[6]) upper triangular matrices with the classical generalized eigenvalues of the pencil on the diagonal.

If the decomposition is ordered so that the j_1 generalized eigenvalues in the first cluster appear first on the diagonal of $\mathbf{X}_1$ and $\mathbf{X}_2$, then the first j_1 columns of $\mathbf{Z}$ give us a matrix $\mathbf{Z}_1$ whose columns form a basis for the corresponding generalized eigenspace of $\mathcal{S}$. Similarly, we obtain a matrix $\mathbf{Z}_2$ whose columns form a basis for the eigenspace corresponding to the next j_2 generalized eigenvalues by reordering the terms in the **QZ** decomposition so that these generalized eigenvalues appear first on the diagonal of $\mathbf{X}_1$ and $\mathbf{X}_2$.

Using Proposition 10.1.1 together with Theorem 10.1.1 shows that the matrices $\mathbf{Z}_1$ and $\mathbf{Z}_2$ that we have obtained are of the form $\mathbf{Z}_i = \mathbf{B}^{-\mathrm{T}}\mathbf{E}_i$ where the $\mathbf{E}_i$ have the property that the jth row of $\mathbf{E}_1$ is nonzero if and only if the jth row of $\mathbf{E}_2$ is a zero row, and vice versa. Define the tensors $\mathcal{T}^1$ and $\mathcal{T}^2$ by

$$\mathcal{T}^i = \mathcal{T} \cdot_2 \mathbf{Z}_i^{\mathrm{T}} = [\![\mathbf{A}, \mathbf{Z}_i^{\mathrm{T}}\mathbf{B}, \mathbf{C}]\!] = [\![\mathbf{A}, \mathbf{E}_i^{\mathrm{T}}, \mathbf{C}]\!].$$

and for $i = 1, 2$, set $R_i = \operatorname{rank}(\mathcal{T}^i)$. Since the $\mathbf{E}_i^{\mathrm{T}}$ have zero columns corresponding to the zero rows of $\mathbf{E}_i$, we have that $R_1 + R_2 = \operatorname{rank}(\mathcal{T})$. Moreover, suppose that $\mathcal{T}^i$ has CPD $[\![\mathbf{A}_i, \check{\mathbf{E}}_i, \mathbf{C}_i]\!]$ for $i = 1, 2$. Here the matrix $\check{\mathbf{E}}_i \in \mathbb{K}^{R \times R_i}$ is the matrix obtained by removing the zero columns of $\mathbf{E}_i^{\mathrm{T}}$. Then up to permutation and scaling, one has $\mathbf{A} = (\mathbf{A}_1 \;\; \mathbf{A}_2)$. Similarly, the factor matrix $\mathbf{C}$ of $\mathcal{T}$ has the form $\mathbf{C} = (\mathbf{C}_1 \;\; \mathbf{C}_2)$ up to scaling and the same permutation as $\mathbf{A}$.

[5] The **QZ** decomposition is the generalization of the Schur decomposition to matrix pencils, e.g., see [198, Chapter VI, Theorem 1.9].

[6] When working over $\mathbb{R}$, if $\mathbf{S}$ has complex generalized eigenvectors, then since $\mathbf{Q}$ and $\mathbf{Z}$ have real entries, there will be 2×2 block matrices on the diagonal of $\mathbf{X}_1$ and $\mathbf{X}_2$, e.g., see [121, Theorem 1.2].

This procedure partitions the rank-1 terms of $\mathcal{T}$ into two distinct collections so that one collection gives a decomposition for the tensor $\mathcal{T}_1$ while the other collection gives a decomposition for $\mathcal{T}_2$. The number of rank-1 terms in each collection, i.e., the rank of $\mathcal{T}_1$ and of $\mathcal{T}_2$, is determined by the number of generalized eigenvalues in each cluster.

The above methodology may be recursively applied to each $\mathcal{T}^i$ to further reduce their ranks until one obtains a collection of tensors all having rank 1 (up to noise). The columns of the factor matrices $\mathbf{A}$ and $\mathbf{C}$ may then be approximated by computing rank-1 approximations for each of these (approximately) rank-1 tensors, see Section 10.2.1 below for details.

Once we have computed the factor matrices $\mathbf{A}$ and $\mathbf{C}$, we may compute $\mathbf{B}$ by solving

$$\mathbf{T}_{[2;3,1]} = \mathbf{B}(\mathbf{C} \odot \mathbf{A})^{\mathrm{T}}.$$

10.2.1 Rank-1 terms

Consider the case where $\mathbf{Z}_1 \in \mathbb{K}^R$ where $\mathbf{Z}_1$ spans an eigenspace of dimension one. In this case $\mathbf{Z}_1$ must have the form

$$\mathbf{Z}_1 = \beta \mathbf{B}^{-\mathrm{T}} \mathbf{e}_\ell^{\mathrm{T}}$$

where $\mathbf{e}_\ell$ is a standard basis vector and $\beta \in \mathbb{K}$ is some constant. Thus we have

$$\mathcal{T} \cdot_2 \mathbf{Z}_1^{\mathrm{T}} = [\![\mathbf{A}, \beta \mathbf{e}_\ell, \mathbf{C}]\!] = [\![\mathbf{a}_\ell, \beta, \mathbf{c}_\ell]\!].$$

In this case we may compute $\mathbf{a}_\ell$ and $\mathbf{c}_\ell$ by computing a rank-1 approximation of the matrix $\mathcal{T} \cdot_2 \mathbf{Z}_1^{\mathrm{T}}$. In practice one may verify that such a tensor has rank 1 (up to noise) by computing the multilinear singular values of $\mathcal{T} \cdot_2 \mathbf{Z}_1^T$.

10.2.2 Choice of slices

We observe that the average performance of the GESD algorithm is improved by first considering a pencil formed by taking the first two frontal slices of the core of the MLSVD of the current subtensor on each splitting step. Intuitively, this is related to the fact that this matrix pencil explains more of the overall sum of squared entries than any other matrix pencil which can be formed from the frontal slices of the tensor.

We emphasize that this strategy is not always successful. Indeed, there are many settings where the generalized eigenvalues obtained by considering the

first two slices of the core of an MLSVD are poorly separated. When this occurs it is necessary to consider other matrix pencils; see Section 10.4.4 for an example. In the case that a successful split is not obtained by considering slices of the MLSVD, we move on to considering random linear combinations of frontal slices.

Note that an MLSVD is always computed for each subtensor to address other potential issues (e.g., repeated factor matrix columns). Furthermore computing MLSVDs of the subtensors is cheap relative to other steps of the algorithm. Thus first considering slices of the MLSVDs has little impact on overall computational cost and on average significantly improves accuracy.

Pseudo code for the GESD algorithm is found in Algorithm 10.1 and Algorithm 10.2.

10.2.3 Handling of complex eigenvectors when working over $\mathbb{R}$

Our explanation of the GESD algorithm focuses on the noiseless case. An important aspect of this case is that the joint generalized eigenvectors of a slice mix invertible real rank-R tensor $\mathcal{T} \in \mathbb{R}^{R \times R \times K}$ are necessarily real valued [67], and, thus, the subpencils of such a tensor have real valued generalized eigenvectors. In the noisy setting, however, it is possible that a subpencil of a perturbation $\mathcal{Y} = \mathcal{T} + \mathcal{N}$ has complex valued generalized eigenvectors.

For example, suppose $\mathcal{T} \in \mathbb{R}^{R \times R \times K}$ is a slice mix invertible tensor of real rank R, and suppose $\mathfrak{Z}$ is a JGE space of dimension 2. Let $\mathbf{Z} \in \mathbb{K}^{R \times 2}$ be a matrix whose columns form a basis for $\mathfrak{Z}$ and let $\mathcal{S}$ be an MLSVD of the tensor $\mathcal{Y} \cdot_2 \mathbf{Z}^{\mathrm{T}}$. Then $\mathcal{S}$ is a $2 \times 2 \times 2$ tensor. In the noiseless setting (i.e., if $\mathcal{Y} = \mathcal{T}$), if $\mathcal{S}$ is slice mix invertible, then $\mathcal{S}$ necessarily has a real rank 2 and both generalized eigenvectors of $\mathcal{S}$ are real valued. However, with sufficient noise, it is possible that $\mathcal{S}$ has real rank 3. This corresponds to the case where the generalized eigenvectors of $\mathcal{S}$ are complex valued.

If $\mathcal{S}$ has rank 3, then $\mathcal{S}$ does not have a best real rank-2 approximation [51], so $\mathcal{S}$ cannot be further decomposed and the true real rank-R decomposition of the original tensor cannot be computed by the GESD algorithm at this point. However, the factor matrix $\mathbf{A}$ (resp. $\mathbf{C}$) in the CPD for the original tensor will have two columns which (approximately) lie in the mode 1 (resp. mode 3) subspace of the tensor $\mathcal{Y} \cdot_2 \mathbf{Z}^{\mathrm{T}}$.

[7] When working over $\mathbb{C}$ there is no natural way to sort the generalized eigenvalues. As a consequence one must compute the chordal distance between all pairs of generalized eigenvalues in order to cluster the generalized eigenvalues into well separated collections.

Algorithm 10.1: GESD recursive step

function `cpd_gesd` $(\mathcal{M}, R, \texttt{Random})$
Input : Tensor $\mathcal{M}$, rank R, boolean `Random`
Output : $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$
$K = \min(\text{size}(\mathcal{T}, 3), R)$
$[\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathcal{Y}] = \texttt{mlsvd}(\mathcal{M}, [R, R, K])$;
% Compute MLSVD of $\mathcal{T}$ / Compress
if $R = 1$ **then**
  $\mathbf{A}_{\text{core}} = y_{111}$; $\mathbf{B}_{\text{core}} = 1$; $\mathbf{C}_{\text{core}} = 1$; % Trivial case
else
  $[\mathbf{Z}_1, \ldots, \mathbf{Z}_N] = \texttt{qz_split}(\mathcal{Y}, \texttt{Random})$;
  % Split JGE spaces into N separated subgroups (Algorithm 10.2)
  for $i = 1 : N$ **do**
    $\mathcal{Y}^i = \mathcal{Y} \cdot_2 \mathbf{Z}_i^{\mathrm{T}}$; % Reduce tensor to Nth subgroup
    $R_i = \texttt{size}(\mathbf{Z}_i, 2)$ % Specify rank for Nth subgroup
    $[\![\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]\!] = \texttt{cpd_gesd}(\mathcal{Y}^i, R_i)$;
    % Recursion on smaller tensor and smaller rank
  end
  $\mathbf{A}_{\text{core}} = [\mathbf{A}_1, \ldots, \mathbf{A}_N]$; $\mathbf{C}_{\text{core}} = [\mathbf{C}_1, \ldots, \mathbf{C}_N]$;
  % Build $\mathbf{A}$ and $\mathbf{C}$ from subblocks
  $\mathbf{B}_{\text{core}} = \mathbf{Y}_{[2;3,1]} / (\mathbf{C} \odot \mathbf{A})^{\mathrm{T}}$; % Find $\mathbf{B}$ by solving linear system
end
$\mathbf{A} = \mathbf{V}_1 \mathbf{A}_{\text{core}}$; $\mathbf{B} = \mathbf{V}_2 \mathbf{B}_{\text{core}}$; $\mathbf{C} = \mathbf{V}_3 \mathbf{C}_{\text{core}}$;
% Undo the MLSVD / Expand

Algorithm 10.2: Splitting eigenspaces (over $\mathbb{R}$)[7]

function `qz_split` ($\mathcal{Y}$, `Random`)
Input : MLSVD core tensor $\mathcal{Y}$, boolean `Random`
Output : N matrices $\mathbf{Z}_n$ spanning JGE spaces
$R = \texttt{size}(\mathcal{Y}, 1)$
$N = 0$
while $N < 2$ **do**
  if *Random* **then**
    $\mathcal{S} = \mathcal{Y} \cdot_3 \texttt{qr}(\texttt{randn}(R, 2))^{\mathrm{T}}$; % Construct random pencil
  else
    Set indices k_1 and k_2 of next set of MLSVD slices
    $\mathcal{S} = \mathcal{Y}_{:,:,[k_1,k_2]}$; % Take MLSVD slices as pencil
  end
  $[\mathbf{X}_1, \mathbf{X}_2, \mathbf{Q}, \mathbf{Z}] = \texttt{qz}(\mathbf{S}_{::1}, \mathbf{S}_{::2})$;
    % Compute QZ decomposition of pencil
  Order eigenvalues $x_{2,ii}/x_{1,ii}$
  Compute chordal gaps between adjacent pairs $(x_{1,ii}, x_{2,ii})$
  Set N to be the number of chordal gaps above threshold
end
Split eigenvalues into N clusters according to gaps above threshold
for $n = 1 : N$ **do**
  Define binary vector $\mathbf{i}_n$ indicating the nth cluster.
  $[\mathbf{X}_{1,n}, \mathbf{X}_{2,n}, \mathbf{Q}_n, \mathbf{Z}_n] = \texttt{ordqz}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{Q}, \mathbf{Z}, \mathbf{i}_n)$;
    % Modify QZ decomposition such that nth cluster is at top
  $\mathbf{Z}_n = (\mathbf{Z}_n)_{:,1:\texttt{sum}(\mathbf{i}_n)}$ % Extract JGE space of nth cluster
end

At this point one may accept that these particular columns of $\mathbf{A}$ and $\mathbf{C}$ cannot be computed and replace them with bases for the two dimensional subspaces to which they belong. Alternatively, one could use the results from the GESD algorithm (with the bases for these two dimensional subspaces used as those two columns for $\mathbf{A}$ and $\mathbf{C}$) as an initialization in an optimization routine for computing the full CPD. While it is unlikely that these basis vectors are the true columns of $\mathbf{A}$ or $\mathbf{C}$, the optimization routine will at least be initialized with these columns in the correct two dimensional subspace which is a significant improvement over a random initialization.

10.3 A deterministic bound for distinct eigenvalue clusters

Our main goal in this section is to provide a deterministic bound for the existence of a subpencil of the tensor of interest which has two distinct clusters of generalized eigenvalues which cannot have coalesced, a notion we now explain.

Suppose $\mathcal{T} \in \mathbb{K}^{R \times R \times 2}$ is a matrix pencil with distinct generalized eigenvalues. Then for sufficiently small perturbations $\mathcal{N}$, the generalized eigenvalues and generalized eigenvectors of $\mathcal{T}+\mathcal{N}$ are continuous in $\mathcal{N}$. However, if the perturbation $\mathcal{N}$ is large enough so that the matrix pencil $\mathcal{T}+\mathcal{N}$ has a repeated generalized eigenvalue, i.e., so that at least two generalized eigenvalues of $\mathcal{T}+\mathcal{N}$ have coalesced, then the path of corresponding generalized eigenvectors cannot be unambiguously continued, as after the generalized eigenvalues separate there is ambiguity in which generalized eigenvalue corresponds to which generalized eigenvector.

Although it is no longer possible to unambiguously continue the paths of generalized eigenvectors themselves after generalized eigenvalues have coalesced, the path of the corresponding generalized eigenspace is still continuous and unambiguous in $\mathcal{N}$ so long as the underlying generalized eigenvalues do not coalesce with other generalized eigenvalues. Thus guaranteeing the existence of two distinct clusters of generalized eigenvalues which cannot have coalesced guarantees that $\mathcal{T}+\mathcal{N}$ has at least two distinct generalized eigenspaces which are still continuous in the perturbation $\mathcal{N}$. These two generalized eigenspaces are still closely related to the original generalized eigenspaces of $\mathcal{T}$. This idea is formalized for matrix pencils in [55]. Also see [114] for the matrix case.

We note that the bound presented in this section can not only guarantee the existence of clusters of generalized eigenvalues which have not coalesced, but can also guarantee a minimum separation between the clusters of generalized eigenvalues. As previously mentioned, the separation between clusters of generalized eigenvalues heavily impacts the accuracy of computing corresponding generalized eigenspaces, so in this way our bounds have significant implications for the accuracy of a given step of the GESD algorithm.

When discussing our deterministic bound we restrict to the real setting. That is, in this section $\mathcal{T} \in \mathbb{R}^{R \times R \times K}$ is a slice mix invertible real tensor which has real rank R. The bound can be generalized to the complex setting as well, though.

We now introduce some notation. Given a matrix $\mathbf{M}$, let $\sigma_{\min}(\mathbf{M})$ denote the smallest singular value of $\mathbf{M}$. Define the Grassmannian $\mathrm{Gr}(1, \mathbb{K}^2)$ to be the collection of subspaces of $\mathbb{K}^2$ of dimension 1. Note that any element of $\mathrm{Gr}(1, \mathbb{R}^2)$

may be uniquely represented in the form $\operatorname{span}(\cos(\theta), \sin(\theta))$ for some $\theta \in [0, \pi)$. Thus given a subspace $\mathfrak{L} \in \operatorname{Gr}(1, \mathbb{R}^2)$ we define

$$\theta(\mathfrak{L}) \in [0, \pi) \qquad \text{by} \qquad \mathfrak{L} = \operatorname{span}\Big(\big(\cos(\theta(\mathfrak{L})), \sin(\theta(\mathfrak{L})) \big) \Big).$$

Additionally, for a nonzero vector $\boldsymbol{\lambda} \in \mathbb{R}^2$, define

$$\theta(\boldsymbol{\lambda}) = \theta(\operatorname{span}(\boldsymbol{\lambda})).$$

The metric we use for the Grassmannian is the chordal metric. Given two subspaces $\mathfrak{L}_1, \mathfrak{L}_2 \in \operatorname{Gr}(1, \mathbb{K}^2)$ define the chordal metric between $\mathfrak{L}_1$ and $\mathfrak{L}_2$, denoted $\chi(\mathfrak{L}_1, \mathfrak{L}_2)$ to be the sine of the (principal) angle between $\mathfrak{L}_1$ and $\mathfrak{L}_2$. It is straightforward to check that $\operatorname{Gr}(1, \mathbb{K}^2)$ is a path connected metric space under the chordal metric. We extend the definition of χ to $\mathbb{K}^2 \backslash \{0\}$ by identifying nonzero vectors in $\mathbb{K}^2$ with their spans. That is, given $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \mathbb{K}^2 \backslash \{0\}$ we have

$$\chi(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) := \chi(\operatorname{span}(\boldsymbol{\lambda}_1), \operatorname{span}(\boldsymbol{\lambda}_2)).$$

In [239], further discussion of the chordal metric and distance in Grassmanians can be found.

Lastly, for a tensor $\mathcal{T} \in K^{R \times R \times K}$, define the spectral norm, denoted $\|\mathcal{T}\|_{\text{sp}}$ by

$$\|\mathcal{T}\|_{\text{sp}} := \max_{\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = \|\mathbf{z}\|_2 = 1} |\mathcal{T} \cdot_1 \mathbf{x}^{\mathrm{T}} \cdot_2 \mathbf{y}^{\mathrm{T}} \cdot_3 \mathbf{z}^{\mathrm{T}}|$$

where the vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are of compatible dimension. The spectral norm of $\mathcal{T}$ is given by the Frobenius norm of a best rank-1 approximation of $\mathcal{T}$; see [75–77,131] for further discussion.

Our first step towards a bound for an $R \times R \times K$ tensor to have a subpencil in which generalized eigenvalues have not coalesced is to give a bound which guarantees that a matrix pencil, i.e., an $R \times R \times 2$ tensor, has generalized eigenvalues which have not coalesced.

Theorem 10.3.1:
Let $\mathcal{T}$ and $\mathcal{Y}$ be tensors of size $R \times R \times 2$. Assume that $\mathcal{T}$ is slice mix invertible and has real rank R with CPD $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, where the columns of $\mathbf{C}$ are normalized to have unit norm.

Let $\{\text{span}(\boldsymbol{\lambda}_1), \ldots, \text{span}(\boldsymbol{\lambda}_R)\}$ be the generalized eigenvalues of $\mathcal{T}$ where the indices are ordered so that

$$0 \leq \theta(\boldsymbol{\lambda}_r) \leq \theta(\boldsymbol{\lambda}_{r+1}) < \pi \quad \text{for} \quad r = 1, \ldots, R-1,$$

and assume at least J of the $\boldsymbol{\lambda}_r$ are distinct. Define $\boldsymbol{\lambda}_{R+1} = \boldsymbol{\lambda}_1$ and let δ be the Jth largest element of the set

$$\left\{ \frac{\chi(\boldsymbol{\lambda}_r, \boldsymbol{\lambda}_{r+1})}{2} \right\}_{r=1}^{R}.$$

Set $\epsilon_1 = \sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})\delta$ and let ϵ_2 be the distance in the spectral norm from $\mathcal{T}$ to the nearest pencil which is not slice mix invertible. If

$$\|\mathcal{T} - \mathcal{Y}\|_{\text{sp}} < \min\{\epsilon_1, \epsilon_2\}, \tag{10.3}$$

then the tensor $\mathcal{Y}$ has at least J distinct (possibly complex valued) generalized eigenvalues.

Furthermore, if Equation (10.3) is satisfied and one sets $\|\mathcal{T} - \mathcal{Y}\|_{\text{sp}} = \epsilon$, then the generalized eigenvalues of $\mathcal{Y}$ may be divided into J nonempty sets $\{\mathscr{L}^j\}_{j=1}^{J}$ such that

$$\min_{j \neq \ell} \chi(\mathscr{L}^j, \mathscr{L}^\ell) \geq 2\delta - \frac{2\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})} > 0. \tag{10.4}$$

In the upcoming Lemma 10.3.2, one can find a discussion of how to compute a lower bound for the quantity ϵ_2 appearing in Theorem 10.3.1. In the case $J = R$ one always has $\epsilon_1 \leq \epsilon_2$, see [67, Proposition 4.4].

The proof of Theorem 10.3.1 follows a simple idea. Using perturbation theoretic results for a matrix pencil, one may show that there are at least J distinct connected regions in $\text{Gr}(1, \mathbb{C}^2)$ such that each region contains a generalized eigenvalue of $\mathcal{Y}$ and such that all generalized eigenvalues of $\mathcal{Y}$ are contained in the union of these disjoint regions. Roughly speaking, the 'real part' of these regions lie between each of the J largest chordal gaps between the ordered generalized eigenvalues of $\mathcal{T}$. To formalize this idea we introduce the notion of a Bauer–Fike ball and a Bauer–Fike region around a given generalized eigenvalue of $\mathcal{T}$.

Let the tensor $\mathcal{T}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$ and the generalized eigenvalues $\{\text{span}(\boldsymbol{\lambda}_r)\}_{r=1}^{R}$ be as in the statement of Theorem 10.3.1. In particular, $\mathcal{T}$ is slice mix invertible with $\mathbb{R}$-rank R and the columns of $\mathbf{C}$ have unit norm. We define the (real or complex) Bauer–Fike ball, for the parameter ϵ around a given generalized eigenvalue $\text{span}(\boldsymbol{\lambda}_r)$ of $\mathcal{T}$, denoted $\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_r)\subset \text{Gr}(1,\mathbb{K}^2)$, by

$$\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_r)=\left\{\text{span}(\boldsymbol{\gamma})\ \in\ \text{Gr}(1,\mathbb{K}^2):\boldsymbol{\gamma}\in\mathbb{K}^2/\{0\}\text{ and }\chi(\boldsymbol{\gamma},\boldsymbol{\lambda}_r)\leq\frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}\right\}.$$

Using the notion of a Bauer–Fike ball[8], we define the Bauer–Fike region for the parameter ϵ around a generalized eigenvalue $\text{span}(\boldsymbol{\lambda}_r)$, denoted $\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r)$ in the following manner.

For each $N=0,1,2,\ldots$ define

$$\text{BF}_{\epsilon}^{\mathbb{K},N+1}(\boldsymbol{\lambda}_r)=\cup_{\text{BF}_{\epsilon}^{\mathbb{K},N}(\boldsymbol{\lambda}_r)\cap\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_\ell)\neq\emptyset}\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_\ell).$$

That is, $\text{BF}_{\epsilon}^{\mathbb{K},N+1}(\boldsymbol{\lambda}_r)$ is the union over ℓ of Bauer–Fike balls $\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_\ell)$ such that $\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_\ell)$ has nontrivial intersection with $\text{BF}_{\epsilon}^{\mathbb{K},N}(\boldsymbol{\lambda}_r)$. The Bauer–Fike region $\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r)$ is defined by

$$\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r)=\cup_{N=1}^{\infty}\text{BF}_{\epsilon}^{\mathbb{K},N}(\boldsymbol{\lambda}_r).$$

It is not difficult to show that there is an integer N such that

$$\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r)=\text{BF}_{\epsilon}^{\mathbb{K},N}(\boldsymbol{\lambda}_r).$$

In particular one can take $N=R$.

Bauer–Fike regions may be equivalently defined as follows. Set

$$\mathscr{D}^{\mathbb{K}}=\cup_{r=1}^{R}\text{BF}_{\epsilon}^{\mathbb{K},0}(\boldsymbol{\lambda}_r)\subset\text{Gr}(1,\mathbb{K}^2),$$

and let $\mathscr{D}^{\mathbb{K}}=\cup_{r=1}^{R'}\mathscr{C}_r^{\mathbb{K}}$ be a partitioning of $\mathscr{D}^{\mathbb{K}}$ into $R'\leq R$ disjoint connected components (in the topology induced by the chordal metric). The Bauer–Fike region $\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r)$ is the unique connected component $\mathscr{C}_j$ of $\mathscr{D}^{\mathbb{K}}$ such that $\boldsymbol{\lambda}_r\in\mathscr{C}_j$. While this definition may be more intuitive, our initial definition plays an important role in the proof of the upcoming Proposition 10.3.1.

The complex Bauer–Fike regions for a given tensor $\mathcal{T}$ are useful in that they give us control over the locations of the generalized eigenvalues of a perturbation

[8]For clarity we emphasize that a Bauer–Fike ball for the parameter ϵ around a generalized eigenvalue $\mathfrak{L}$ of $\mathcal{T}$ is simply the closed neighborhood in $\text{Gr}(1,\mathbb{K}^2)$ of radius $\frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}$ centered at $\mathfrak{L}$. We introduce this notation to streamline the upcoming definition of Bauer–Fike regions, and to clearly illustrate the connection between these sets.

of $\mathcal{T}$. Assume ϵ is small enough so that $\|\mathcal{T}-\mathcal{Y}\|_{\text{sp}} \leq \epsilon$ implies that $\mathcal{Y}$ is slice mix invertible. Then following the argument in the proof of [67, Theorem 3.5] shows that the number of generalized eigenvalues of $\mathcal{Y}$ contained by each complex Bauer–Fike region for the parameter ϵ is equal to the number of overlapping complex Bauer–Fike balls which make up that complex Bauer–Fike region. That is, for a fixed index r, if there are exactly m indices ℓ (counting the case $r=\ell$) such that

$$\text{BF}_{\epsilon}^{\mathbb{C}}(\boldsymbol{\lambda}_r) \cap \text{BF}_{\epsilon}^{\mathbb{C},0}(\boldsymbol{\lambda}_\ell) \neq \emptyset,$$

and if $\|\mathcal{T}-\mathcal{Y}\|_{\text{sp}} \leq \epsilon$, then the complex Bauer–Fike region $\text{BF}_{\epsilon}^{\mathbb{C}}(\boldsymbol{\lambda}_r)$ contains exactly m of the generalized eigenvalues of the tensor $\mathcal{Y}$ counting algebraic multiplicity.

Our goal in the proof of Theorem 10.3.1 will be to show that $\mathcal{T}$ has at least J distinct complex Bauer–Fike regions for the parameter ϵ. Following the above discussion shows that if $\mathcal{T}$ and $\mathcal{Y}$ satisfy the assumptions of Theorem 10.3.1, then $\mathcal{Y}$ has at least J distinct clusters of generalized eigenvalues which cannot have coalesced in a perturbation from $\mathcal{T}$ to $\mathcal{Y}$. To show that $\mathcal{T}$ has J distinct Bauer–Fike regions, we will make use of the fact that Bauer–Fike regions are connected subsets of $\text{Gr}(1,\mathbb{K}^2)$.

Lemma 10.3.1:
Let $\mathcal{T} \in \mathbb{R}^{R\times R\times 2}$ be a slice mix invertible $\mathbb{R}$-rank-R tensor with generalized eigenvalues $\mathfrak{L} = \{\boldsymbol{\lambda}_r\}_{r=1}^{R}$. Then for each r, the Bauer–Fike region $\text{BF}_{\epsilon}^{\mathbb{K}}(\boldsymbol{\lambda}_r) \subset \text{Gr}(1,\mathbb{K}^2)$ is path connected in the topology on $\text{Gr}(1,\mathbb{K}^2)$ given by the chordal metric.

Another technical part of our proof will be to restrict from considering complex Bauer–Fike regions to considering real Bauer–Fike regions. We emphasize that the generalized eigenvalues of $\mathcal{Y}$ can only be guaranteed to lie in the union of the complex Bauer–Fike regions of $\mathcal{T}$, thus it is necessary to use complex Bauer–Fike regions. However, complex Bauer–Fike regions are more difficult to work with due to a lack of any notion of ordering in $\text{Gr}(1,\mathbb{C}^2)$, hence our desire to reduce to working over $\mathbb{R}$. The following proposition makes this reduction possible.

Proposition 10.3.1:
Let $\mathcal{T} \in \mathbb{R}^{R\times R\times 2}$ be a slice mix invertible tensor having real rank R. Let $\{\text{span}(\boldsymbol{\lambda}_r)\}_{r=1}^{R}$ be the set of generalized eigenvalues of $\mathcal{T}$ with $\boldsymbol{\lambda}_r \in \mathbb{R}^2$ for each $r = 1, \ldots, R$. Then for any r_1, r_2 one has

$$\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset \quad \text{if and only if} \quad \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset. \tag{10.5}$$

Additionally,

$$\text{BF}_\epsilon^{\mathbb{C}}(\boldsymbol{\lambda}_{r_1}) = \text{BF}_\epsilon^{\mathbb{C}}(\boldsymbol{\lambda}_{r_2}) \quad \text{if and only if} \quad \text{BF}_\epsilon^{\mathbb{R}}(\boldsymbol{\lambda}_{r_1}) = \text{BF}_\epsilon^{\mathbb{R}}(\boldsymbol{\lambda}_{r_2})$$

It follows that the number of distinct complex Bauer–Fike regions for $\mathcal{T}$ is exactly equal to the number of distinct real Bauer–Fike regions for $\mathcal{T}$.

Proof. Let $\mathcal{T}$ and $\boldsymbol{\lambda}_{r_1}$ and $\boldsymbol{\lambda}_{r_2}$ be as in the statement of the lemma. It is clear that $\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset$ implies $\text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset$. On the other hand, using the fact that a vector $\boldsymbol{\zeta}$ which minimizes the chordal distance to a set of real vectors $\{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2\}$ may be obtained by taking $\boldsymbol{\zeta}$ to be a real vector that bisects the angle between $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$, it is straightforward to show that if

$$\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) \neq \emptyset,$$

then

$$\text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}) \neq \emptyset.$$

In particular, if $\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) \neq \emptyset$, then

$$\text{span}(\boldsymbol{\zeta}) \in \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}),$$

hence

$$\text{span}(\boldsymbol{\zeta}) \in \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}).$$

Thus we have $\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset$ if and only if $\text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_1}) \cap \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}) = \emptyset$.

Using Equation (10.5) together with the above, one may show by induction that for all $N \in \mathbb{N}$ and all $r_1, r_2 \in \{1, \ldots, R\}$ one has

$$\text{BF}_\epsilon^{\mathbb{C},0}(\boldsymbol{\lambda}_{r_2}) \subset \text{BF}_\epsilon^{\mathbb{C},N}(\boldsymbol{\lambda}_{r_1}) \quad \text{if and only if} \quad \text{BF}_\epsilon^{\mathbb{R},0}(\boldsymbol{\lambda}_{r_2}) \subset \text{BF}_\epsilon^{\mathbb{R},N}(\boldsymbol{\lambda}_{r_1}). \tag{10.6}$$

Furthermore, it is straightforward to check that

$$\text{BF}_\epsilon^{\mathbb{K}}(\boldsymbol{\lambda}_{r_1}) = \text{BF}_\epsilon^{\mathbb{K}}(\boldsymbol{\lambda}_{r_2})$$

if and only if there is an integer N such that

$$\text{BF}_\epsilon^{\mathbb{K},0}(\boldsymbol{\lambda}_{r_2}) \subset \text{BF}_\epsilon^{\mathbb{K},N}(\boldsymbol{\lambda}_{r_1}).$$

Combining this fact with Equation (10.6) completes the proof. □

10.3.1 Proof of Theorem 10.3.1

We have now laid the necessary ground work to give the proof of Theorem 10.3.1. The proof is technical, but, as previously mentioned, the main idea behind the proof is simple. Before giving technical details, we give an overview of the role played by Bauer–Fike regions.

As previously discussed, each distinct Bauer–Fike region for $\mathcal{T}$ contains a generalized eigenvalue of $\mathcal{Y}$ and all the generalized eigenvalues of $\mathcal{Y}$ are contained in some Bauer–Fike region of $\mathcal{T}$. This fact is proved in the tensor spectral norm by [67, Theorem 4.3]; however, in a different norm (which upper bounds the tensor spectral norm) this fact is classical, e.g., see [198, Chapter VI, Theorem 2.7]. By using this fact together with Proposition 10.3.1 to show that $\mathcal{Y}$ has at least J distinct generalized eigenvalues, it is sufficient to show that $\mathcal{T}$ has J distinct real Bauer–Fike regions.

To show that $\mathcal{T}$ has at least J distinct real Bauer–Fike regions, we divide $\mathrm{Gr}(1,\mathbb{R}^2)$ into J distinct regions such that each region fully contains a real Bauer–Fike region of $\mathcal{T}$. The desired division of $\mathrm{Gr}(1,\mathbb{R}^2)$ into distinct regions is accomplished by using subspaces of $\mathbb{R}^2$ to cut in half each of the J largest gaps between the ordered generalized eigenvalues of $\mathcal{T}$.

Our choice of parameter for our Bauer–Fike balls will guarantee that these cuts cannot be contained in any Bauer–Fike ball of $\mathcal{T}$, hence, these cuts cannot be contained in any Bauer–Fike region. The fact that a real Bauer–Fike region lying between two adjacent cuts is distinct from a real Bauer–Fike region lying between another pair of adjacent cuts follows from the path connectedness of the real Bauer–Fike regions. Details follow below.

Proof. Let $\mathcal{T}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$ and J and ϵ be as in the statement of Theorem 10.3.1 and assume $J\geq 2$. We first show that $\mathcal{T}$ has J distinct complex Bauer–Fike regions. Using Proposition 10.3.1, it is sufficient to show that $\mathcal{T}$ has J distinct real Bauer–Fike regions.

Now, from the ordering of our eigenvalues, there must exist J distinct indices $1\leq r_1<\cdots<r_J\leq R$ such that for each $r_j\neq R$ there is a unit vector $\boldsymbol{\eta}_j\in\mathbb{R}^2$ satisfying

$$\theta(\boldsymbol{\lambda}_{r_j})<\theta(\boldsymbol{\eta}_j)<\theta(\boldsymbol{\lambda}_{r_j+1})$$

and

$$\chi(\boldsymbol{\eta}_j,\boldsymbol{\lambda}_{r_j})\geq\delta \qquad \text{and} \qquad \chi(\boldsymbol{\eta}_j,\boldsymbol{\lambda}_{r_j+1})\geq\delta.$$

Using $\epsilon_1=\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})\delta$ and $\epsilon<\epsilon_1$ gives

$$\chi(\boldsymbol{\eta}_\ell,\boldsymbol{\lambda}_{r_\ell})>\frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})} \qquad \text{and} \qquad \chi(\boldsymbol{\eta}_\ell,\boldsymbol{\lambda}_{r_\ell+1})>\frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}.$$

If the case $r_J = R$ occurs, then we may instead choose a unit vector $\boldsymbol{\eta}_J$ such that

$$\chi(\boldsymbol{\eta}_J, \boldsymbol{\lambda}_R) > \frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})} \qquad \text{and} \qquad \chi(\boldsymbol{\eta}_J, \boldsymbol{\lambda}_1) > \frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}$$

and such that either

$$\theta(\boldsymbol{\lambda}_R) < \theta(\boldsymbol{\eta}_J) \qquad \text{or} \qquad \theta(\boldsymbol{\eta}_J) < \theta(\boldsymbol{\lambda}_1).$$

To simplify exposition, we assume that $\theta(\boldsymbol{\lambda}_R) < \theta(\boldsymbol{\eta}_J)$.

Note that $\cup_{j=1}^{J} \operatorname{span}(\boldsymbol{\eta}_j)$ divides $\operatorname{Gr}(1, \mathbb{R}^2)$ into J distinct open connected components. In particular, there exist J disjoint open subsets $\mathscr{G}_1, \ldots, \mathscr{G}_J$ of $\operatorname{Gr}(1, \mathbb{R}^2)$ such that

$$\cup_{j=1}^{J} \mathscr{G}_j = \operatorname{Gr}(1, \mathbb{R}^2) \setminus \left(\cup_{j=1}^{J} \operatorname{span}(\boldsymbol{\eta}_j) \right).$$

Using the ordering of our generalized eigenvalues together with the definition of the $\boldsymbol{\eta}_j$, a straightforward check shows that $\operatorname{span}(\boldsymbol{\eta}_j)$ cannot be contained in a real Bauer–Fike region $\mathrm{BF}_{\epsilon}^{\mathbb{R}}(\boldsymbol{\lambda}_r)$ for any $r = 1, \ldots, R$. In particular we have

$$\cup_{r=1}^{R} \mathrm{BF}_{\epsilon}^{\mathbb{R}}(\boldsymbol{\lambda}_r) \subset \left(\cup_{j=1}^{J} \mathscr{G}_j \right). \tag{10.7}$$

Recalling that each Bauer–Fike region is connected and that $\mathscr{G}_j$ are open and disjoint sets, Equation (10.7) then implies that if there are indices r and j such that

$$\mathrm{BF}_{\epsilon}^{\mathbb{R}}(\boldsymbol{\lambda}_r) \cap \mathscr{G}_j \neq \emptyset, \qquad \text{then} \qquad \mathrm{BF}_{\epsilon}^{\mathbb{R}}(\boldsymbol{\lambda}_r) \subseteq \mathscr{G}_j. \tag{10.8}$$

By construction we have

$$\mathscr{G}_j \cap \left(\cup_{r=1}^{R} \mathrm{BF}_{\epsilon}^{\mathbb{R}}(\boldsymbol{\lambda}_r) \right) \neq \emptyset \qquad \text{for each } j = 1, \ldots, J.$$

Combining this with Equation (10.8) shows that each $\mathscr{G}_j$ contains some real Bauer–Fike region of $\mathcal{T}$. In particular, $\mathcal{T}$ has at least J distinct real Bauer–Fike regions. An application of Proposition 10.3.1 then shows that $\mathcal{T}$ has at least J distinct complex Bauer–Fike regions.

Now let $\mathcal{Y} \in \mathbb{R}^{R \times R \times 2}$ be a tensor which satisfies

$$\|\mathcal{T} - \mathcal{Y}\|_{\mathrm{sp}} \leq \epsilon < \min\{\epsilon_1, \epsilon_2\}.$$

Following the argument used to prove [67, Theorem 2.8] shows that for each $j = 1, \ldots, J$, the region $\mathrm{BF}_{\epsilon}^{\mathbb{C}}(\boldsymbol{\lambda}_{r_j})$ must contain at least one generalized eigenvalue of $\mathcal{Y}$ and that all the generalized eigenvalues of $\mathcal{Y}$ are contained in some Bauer–Fike region. Therefore, since $\mathcal{T}$ has at least J distinct Bauer–Fike regions, the tensor $\mathcal{Y}$ has at least J distinct generalized eigenvalues.

It remains to construct the nonempty sets $\{\mathscr{L}^j\}_{j=1}^J$ of generalized eigenvalues of $\mathcal{Y}$ which satisfy Equation (10.4). To this end, let $\{\text{span}(\tilde{\boldsymbol{\lambda}}_r)\}_{r=1}^R$ be the set of generalized eigenvalues of $\mathcal{Y}$ and for each $j=1,\dots,J$ define $\mathscr{L}^j$ by

$$\mathscr{L}^j = \left\{\text{span}(\tilde{\boldsymbol{\lambda}}_\ell) \,:\, \tilde{\boldsymbol{\lambda}}_\ell \in \{\tilde{\boldsymbol{\lambda}}_r\}_{r=1}^R \cap \mathscr{G}_j\right\}.$$

In words, $\mathscr{L}^j$ is the set of generalized eigenvalues of $\mathcal{Y}$ which are contained in $\mathscr{G}_j$.

Now suppose $\tilde{\boldsymbol{\lambda}}_m$ and $\tilde{\boldsymbol{\lambda}}_n$ are generalized eigenvalues of $\mathcal{Y}$ such that $\tilde{\boldsymbol{\lambda}}_m \in \mathscr{G}_i$ and $\tilde{\boldsymbol{\lambda}}_n \in \mathscr{G}_\ell$ for some indices $i \neq \ell$. Then there must exist generalized eigenvalues $\boldsymbol{\lambda}_m$ and $\boldsymbol{\lambda}_n$ of $\mathcal{T}$ such that $\boldsymbol{\lambda}_m \in \mathscr{G}_i$ and $\boldsymbol{\lambda}_n \in \mathscr{G}_\ell$ and

$$\chi(\boldsymbol{\lambda}_m, \tilde{\boldsymbol{\lambda}}_m) \leq \frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})} \qquad \text{and} \qquad \chi(\boldsymbol{\lambda}_n, \tilde{\boldsymbol{\lambda}}_n) \leq \frac{\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}.$$

Furthermore, from the definition of the $\mathscr{G}_j$ we must have

$$\chi(\boldsymbol{\lambda}_m, \boldsymbol{\lambda}_n) \geq 2\delta.$$

Using the triangle inequality shows

$$\chi(\tilde{\boldsymbol{\lambda}}_m, \tilde{\boldsymbol{\lambda}}_n) \geq \chi(\boldsymbol{\lambda}_m, \boldsymbol{\lambda}_n) - \chi(\boldsymbol{\lambda}_m, \tilde{\boldsymbol{\lambda}}_m) - \chi(\boldsymbol{\lambda}_n, \tilde{\boldsymbol{\lambda}}_n) \geq 2\delta - \frac{2\epsilon}{\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})}$$

from which inequality (10.4) follows. □

10.3.2 Distinct eigenvalue clusters for subpencils of a tensor

We now use Theorem 10.3.1 to give the radius of a Frobenius norm ball around a given tensor in which every tensor is guaranteed to have a subpencil with at least J distinct generalized eigenvalues.

Corollary 10.3.1:
Let $\mathcal{T} \in \mathbb{R}^{R\times R\times K}$ be a slice mix invertible tensor of real rank R. Let $\mathbf{U} \in \mathbb{R}^{K\times K}$ be any orthogonal matrix. Set $\mathcal{S} = \mathcal{T} \cdot_3 \mathbf{U}$. For each $k=1,\dots,\lfloor K/2 \rfloor$, let $\epsilon_k \geq 0$ be the bound obtained from Theorem 10.3.1 which guarantees that matrix pencils in a ball of Frobenius radius ϵ_k around the subpencil

$$(\mathbf{S}_{2k-1}, \mathbf{S}_{2k})$$

have at least J distinct generalized eigenvalues. Here we take $\epsilon_k = 0$ if the corresponding subpencil is not slice mix invertible. Set $\epsilon = ||(\epsilon_1,\dots,\epsilon_{\lfloor K/2 \rfloor})||_2$. Then every tensor in an open ball of Frobenius radius ϵ around $\mathcal{T}$ has a subpencil which has at least J distinct generalized eigenvalues.

Proof. Let $\mathcal{Y} \in \mathbb{R}^{R\times R\times K}$ be a tensor such that $\|\mathcal{T}-\mathcal{Y}\|_{\mathrm{F}} < \epsilon$ and set $\mathcal{P} = \mathcal{Y} \cdot_3 \mathbf{U}$. Then we have $\|\mathcal{S}-\mathcal{P}\|_{\mathrm{F}} < \epsilon$. It follows that there must be some index $k \in \{1,\dots,\lfloor K/2 \rfloor\}$ such that

$$\|(\mathbf{S}_{2k-1},\mathbf{S}_{2k}) - (\mathbf{P}_{2k-1},\mathbf{P}_{2k})\|_{\mathrm{sp}} \leq \|(\mathbf{S}_{2k-1},\mathbf{S}_{2k}) - (\mathbf{P}_{2k-1},\mathbf{P}_{2k})\|_{\mathrm{F}} < \epsilon_k,$$

hence Theorem 10.3.1 may be applied to the subpencil $(\mathbf{P}_{2k-1},\mathbf{P}_{2k})$ of $\mathcal{P}$. In the above the norm $\|(\mathbf{M}_1,\mathbf{M}_2)\|_{\mathrm{sp}}$ is the tensor spectral norm of the tensor $\mathcal{M} \in \mathbb{R}^{R\times R\times 2}$ whose frontal slices are $\mathbf{M}_1$ and $\mathbf{M}_2$. □

Theorem 10.3.1 and Corollary 10.3.1 are similar to results appearing in [67]; however, the results appearing here are more general and their proofs require additional consideration.

10.3.3 Computation of the bounds appearing in Theorem 10.3.1 and Corollary 10.3.1

We now explain how to compute the bounds presented in this section. We begin by showing how to compute a lower bound for the quantity ϵ_2 in Theorem 10.3.1.

Lemma 10.3.2:
Let $\mathcal{T} \in \mathbb{R}^{R\times R\times K}$ and let $\mathcal{S}$ be a tensor which is not slice mix invertible. Then

$$\|\mathcal{T}-\mathcal{S}\|_{\mathrm{sp}} \geq \max_{\substack{\mathbf{v}\in\mathbb{R}^K \\ \|\mathbf{v}\|_2=1}} \sigma_{\min}(\mathcal{T}\cdot_3 \mathbf{v}^{\mathrm{T}}). \tag{10.9}$$

Proof. Let $\mathbf{v}_* \in \mathbb{R}^K$ be a unit vector which maximizes the right hand side of Equation (10.9). Since $\mathcal{S}$ is not slice mix invertible, the matrix $\mathcal{S}\cdot_3 \mathbf{v}_*^{\mathrm{T}} \in \mathbb{R}^{R\times R}$ has a nontrivial null space. It follows that there is a unit vector $\mathbf{y}_* \in \mathbb{R}^R$ such that $\mathcal{S}\cdot_2 \mathbf{y}_*^{\mathrm{T}} \cdot_3 \mathbf{v}_*^{\mathrm{T}} = \mathbf{0}$. We then have

$$\|\mathcal{T}-\mathcal{S}\|_{\mathrm{sp}} \geq \|(\mathcal{T}-\mathcal{S})\cdot_2 \mathbf{y}_*^{\mathrm{T}} \cdot_3 \mathbf{v}_*^{\mathrm{T}}\|_2 = \|(\mathcal{T}\cdot_3 \mathbf{v}_*^{\mathrm{T}})\mathbf{y}_*\|_2 \geq \sigma_{\min}(\mathcal{T}\cdot_3 \mathbf{v}_*^{\mathrm{T}})$$

as claimed. □

While the quantity on the right hand side of inequality (10.9) is itself nontrivial to compute, one can for example use the lower bound

$$\sigma_{\min}(\mathcal{T}\cdot_3 \mathbf{v}^{\mathrm{T}}) \geq \max\{\sigma_{\min}(\mathbf{T}_1),\dots,\sigma_{\min}(\mathbf{T}_k)\}.$$

We use this lower bound for ϵ_2 in the numerical experiment presented in the upcoming Figure 10.8 and for almost all pencils observe that the quantity ϵ_1 of Theorem 10.3.1 is less than or equal to this lower bound, hence the computed bound is rarely affected in our experiments by using this lower bound.

To compute the quantity ϵ_1, one need only compute a CPD $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathcal{T}$ where the columns of $\mathbf{C}$ are scaled to have unit norm. As shown in Theorem 10.3.1, the generalized eigenvalues of $\mathcal{T}$ are then equal to the spans of the columns of the factor matrix $\mathbf{C}$. A subtlety is that the quantity $\sigma_{\min}(\mathbf{A})\sigma_{\min}(\mathbf{B})$ depends on the scaling of the columns of $\mathbf{A}$ and $\mathbf{B}$. We observe good performance if we scale these factor matrices so that for all $r = 1, \ldots, R$, the norm of rth column of $\mathbf{A}$ is equal to that of the rth column of $\mathbf{B}$, and this is how we scale the factors in our experiments. Locally optimal scaling can be computed using an ALS approach for example; see the discussion in [67, Section 4.4.1] for more details.

For a given unitary $\mathbf{U}$, the bound presented in Corollary 10.3.1 may be computed by applying the preceding discussion to the appropriate subpencils of $\mathcal{T} \cdot_3 \mathbf{U}$. Computing a unitary which gives the optimal bound in Corollary 10.3.1 is a nontrivial problem which is related to the travelling salesman problem; see the discussion in [67, Section 5.1].

In practice we randomly generate unitaries $\mathbf{U}_1, \ldots, \mathbf{U}_L \in \mathbb{R}^{K \times K}$ for some chosen parameter L and keep the maximum bound obtained by applying Corollary 10.3.1 to the tensors $\mathcal{T} \cdot_3 \mathbf{U}_1, \ldots, \mathcal{T} \cdot_3 \mathbf{U}_L$. The random unitaries we consider have the form $\mathbf{Q} \cdot \mathrm{Diag}(\mathrm{sign}(\mathbf{R}))$ where $\mathbf{Q}$ and $\mathbf{R}$ are obtained from the $\mathbf{QR}$ decomposition applied to randomly generated matrices with i.i.d. Gaussian entries. Here $\mathrm{sign}(\mathbf{M})$ applies the sign operator to each entry of the matrix $\mathbf{M}$, while $\mathrm{Diag}(\mathbf{M})$ is the diagonal matrix whose diagonal entries are equal to those of $\mathbf{M}$. Note that generating unitaries in this method is equivalent to randomly sampling unitaries from the Haar (uniform) measure on the unitary group [140].

10.4 GESD vs. GEVD accuracy and timing

We now compare both the accuracy and timing of GESD to GEVD where the GEVD solution is computed using the first two slices of the MLSVD of the tensor[9]. To this end, we generate low-rank tensors and add noise with varying signal to noise ratios (SNRs). We then compute the CPD with both methods and compare the computation time and factor matrix error (`cpderr`) of the results. For a low-rank tensor generated from the factor

[9] Similar to GESD, the average performance of GEVD benefits by using the first two slices of the MLSVD. However, this strategy is not always successful, see Section 10.4.4.

matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, the factor matrix error of its estimated CPD $\left[\!\left[\hat{\mathbf{A}},\hat{\mathbf{B}},\hat{\mathbf{C}}\right]\!\right]$ is defined as $\max\left(\frac{\|\mathbf{A}-\hat{\mathbf{A}}\|_{\mathrm{F}}}{\|\mathbf{A}\|_{\mathrm{F}}}, \frac{\|\mathbf{B}-\hat{\mathbf{B}}\|_{\mathrm{F}}}{\|\mathbf{B}\|_{\mathrm{F}}}, \frac{\|\mathbf{C}-\hat{\mathbf{C}}\|_{\mathrm{F}}}{\|\mathbf{C}\|_{\mathrm{F}}}\right)$, where columns of $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ and $\hat{\mathbf{C}}$ have been optimally permuted and scaled to match the columns of $\mathbf{A}$,$\mathbf{B}$ and $\mathbf{C}$, respectively. The SNR of the tensor $\mathcal{T}$ and the noise $\mathcal{N}$ in dB is defined as $\mathbb{E}\left(20\log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)\right)$. For each SNR, the median error and computation time are computed over 50 trials unless differently specified. The computation time is the time required to obtain the factor matrices $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ and $\hat{\mathbf{C}}$ starting from the noisy tensor $\mathcal{T}+\mathcal{N}$. All experiments in this section and Section 10.5 are performed on a machine with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2020b and Tensorlab 3.0 [228].

We start with two easier problems where the tensor dimensions are considerably larger than the tensor rank. For a more challenging case, we then move on to tensors with factor matrices that have correlated columns. Next, we examine the accuracy and speed of the GESD method compared to a state-of-the art optimization method.

We note that as a tuning parameter in GESD one may adjust the chordal distance between generalized eigenvalues which is sufficiently large for splitting between the corresponding generalized eigenspaces. To a point, the accuracy of GESD may be improved by increasing the requested chordal distance; however, increasing this chordal distance can decrease the speed of GESD. One could also choose to restrict the number of generalized eigenvalue clusters which GESD is allowed to separate in at each step. The optimal splitting strategy in terms of both timing and accuracy is a topic of further investigation. Furthermore, one could additionally consider the separation between computed generalized eigenspaces as a parameter for deciding if a particular split is acceptable.[10] Unless otherwise stated, in the experiments in this section we use a chordal distance of 0.2 as splitting threshold.

10.4.1 Tensor dimension exceeds tensor rank

The following experiments resemble typical problems in signal processing, where the tensor dimensions are often orders of magnitude larger than the tensor rank. For the first experiment, the factor matrix entries of a $100\times100\times100$-tensor

[10] An in-depth discussion of the sensitivity of generalized eigenspace computations to the separation between the corresponding generalized eigenspaces is given in [55]. While the bounds discussed therein take into account both eigenvalue and eigenspace conditioning, they are expensive to compute and lead to major increases in runtime if used in GESD. This is of course undesirable if the GESD algorithm is used to quickly obtain a good initialization for an optimization algorithm. [68] further explores this topic in the context of joint generalized eigenspaces.

of rank 10 are independently sampled from the standard normal distribution. The results are shown in Figure 10.1. In this setting, the GESD algorithm is marginally more accurate than GEVD, but the computational cost of GESD is about three times that of GEVD. While GESD is more computationally expensive, the first step in this setting is to compress the starting tensor. The cost of this compression dominates the other parts of the computation, thus using GESD over GEVD leads to about a 25 percent increase in the total run time of the algorithm.

For the second experiment, the factor matrices are independently sampled from the uniform distribution on $[0,1]$ instead of the standard normal distribution. This setting is more challenging than the first since the columns of the factor matrices are less well separated. In Figure 10.2, we can see the accuracies and computation times of both methods. Compared to the case with normally distributed factor matrix entries, a higher SNR is needed to obtain a reasonable solution, but from an SNR of 0 dB upwards, the error of the GESD method is more than an order of magnitude smaller than the error for the GEVD method. That is, using GESD leads to a gain of over 20 dB in accuracy. As in the previous experiment, the total cost of compression together with GESD is only slightly higher than compression with GEVD since compression is the most computationally expensive step.

10.4.2 Factor matrices with correlated columns

We now consider the more challenging setting where the tensor of interest has rank equal to dimensions. That is, the tensors we consider in this section have rank R and size $R \times R \times R$. This case is significantly more difficult than the $I_1 \times I_2 \times I_3$ rank R case with $I_1, I_2, I_3 \gg R$ which generally occurs in practice and is discussed in Section 10.4.1. Recall that the first step in both GESD and GEVD is to compute a MLSVD of the tensor of interest. In the $I_1, I_2, I_3 \gg R$ case, computing this MLSVD and reducing to an $R \times R \times R$ tensor typically significantly increases the SNR of the problem thereby reducing the difficulty of the problem. In addition, the compression generally takes much more time than computing the CPD of the small core tensor using GEVD or GESD.

In Figure 10.3 we consider the particularly difficult case where the columns of the factor matrices are highly correlated. More concretely, we consider the case where the first column of $\mathbf{A}$ is sampled from the uniform distribution on $[0,1]$ as before and each successive column of $\mathbf{A}$ is randomly sampled so that it forms a 10° angle with $\mathbf{a}_1$ by computing, for $r = 2, 3, \ldots, R$, $\mathbf{a}_r = \cos(10^\circ)\mathbf{a}_1 + \sin(10^\circ)\mathbf{x}_r$ for a random vector $\mathbf{x}_r$ that is orthogonal to $\mathbf{a}_1$ and has the same norm as $\mathbf{a}_1$. The other factor matrices are sampled analogously. Results for this case

are shown in Figure 10.3. We note that, due to the difficulty of this setting, a very high SNR is required before either GESD or GEVD is able to compute a meaningful solution.

For the problems in Figure 10.3, we see that GESD is significantly more accurate than GEVD. In fact, GESD is able to compute meaningful solutions starting at about an order of magnitude lower SNR than GEVD. This demonstrates that the handling of pencil specific eigenvalue clusters is better in GESD than in GEVD due to the fact that GESD is able to compute an eigenspace for the cluster, but delays computation of eigenvectors to a pencil in which the previously clustered eigenvalues are better separated. As before, the GESD algorithm takes about three times as long as the GEVD algorithm.

In Figure 10.4, we investigate the speed and accuracy of both algorithms as a function of tensor rank. Here we run GESD and GEVD on noiseless tensors of rank R and size $R \times R \times R$ where R ranges from 50 to 350 and where the factor matrices are independently sampled from the uniform distribution on $[0, 1]$. For all values of R we see that GESD is roughly two orders of magnitude more accurate than GEVD, but that GESD takes about three times as long to run as GEVD.

10.4.3 Comparison with optimization

As the GEVD algorithm can quickly compute a reasonably good fit for the CPD, its solution is often used as an initialization for an optimization algorithm. By providing a good initialization, the number of iterations of the more accurate yet more computationally expensive optimization algorithm can ideally be lowered significantly. In the next experiment, we compare the error and computation time of a nonlinear least-squares (NLS) algorithm, outlined in Section 2.3.2 and in [223] for three different initialization strategies: starting from the GEVD solution, the GESD solution or a random initial point. The optimization algorithm is run until either the relative cost function change or the relative factor matrix change drops under 10^{-8}. The time to compute the GEVD or GESD solutions is included in the runtime of the optimization algorithm. For comparison, we also include results for the semi-algebraic SECSI algorithm [163], with the default parameters and the 'CON PS' option.

In Figure 10.5, median values are shown over 20 trials for a $10 \times 10 \times 10$-tensor of rank 10 with factor matrices independently sampled from the uniform distribution on $[0, 1]$. We can see that the NLS algorithm generally converges to a good solution for all three initialization strategies. The GEVD solution is notably less accurate than the other solutions, while GESD is almost as accurate as the semi-algebraic SECSI algorithm. In the right plot of Figure 10.5,

it can be seen that the GESD method is about an order of magnitude faster than the optimization algorithm and SECSI. Also, the computation time of the optimization algorithm is significantly reduced by starting from the GEVD or GESD initialization, even when their computation time is included. For all SNRs, computation of a GESD or GEVD initialization followed by optimization is equally fast.

In Figure 10.6, median values are shown over 20 trials for a $10 \times 10 \times 10$-tensor of rank 10 where the columns of the factor matrices are sampled with a fixed 10° angle between them. For this more challenging problem, we can see from the left plot of Figure 10.6 that decent algebraic initializations are only achieved at higher SNR values compared to the uncorrelated case. Additionally we see that the SNR required for GESD to compute reasonably accurate initializations is significantly lower than that for GEVD. In this experiment, random initialization of the optimization algorithm does not lead to a meaningful solution at all (even when the best result over several initializations is considered), which again highlights the importance of algebraic initialization. As before, optimization with a GESD initialization and optimization with a GEVD initialization typically take the same amount of time and the accuracy of the GESD solution lies close to the accuracy of the SECSI solution in an order of magnitude lower computation time.

In Figure 10.7 we consider an even more difficult setting. Here median values are shown over 100 trials for a $20 \times 20 \times 20$-tensor of rank 20 where the columns of the factor matrices are sampled with a fixed 8° angle between them. Due to the difficulty of this setting, meaningful solutions can only be computed with very high SNR. In the left plot of Figure 10.7 we see both that GESD computes a more accurate algebraic initialization than GEVD and, moreover, that using the algebraic initialization from GESD for optimization leads to a more accurate CPD than using the algebraic initialization from GEVD. In addition, we observe that GESD+optimization is faster than GEVD+optimization. As in the previous figure, optimization with random initialization does not lead to a meaningful solution (even when several random initializations are tried) and GESD is almost as accurate as SECSI in a fraction of the computation time.

These experiments illustrate that using GESD as an algebraic initialization compared to GEVD does not cause any loss in terms of time or accuracy. In fact, GESD+optimization can be both more accurate and faster than GVSD+optimization. Furthermore see we that GESD based initializations are significantly more accurate than GEVD based initializations. Thus using GESD over GEVD to initialize gives more reliable initializations without impacting the total computation time.

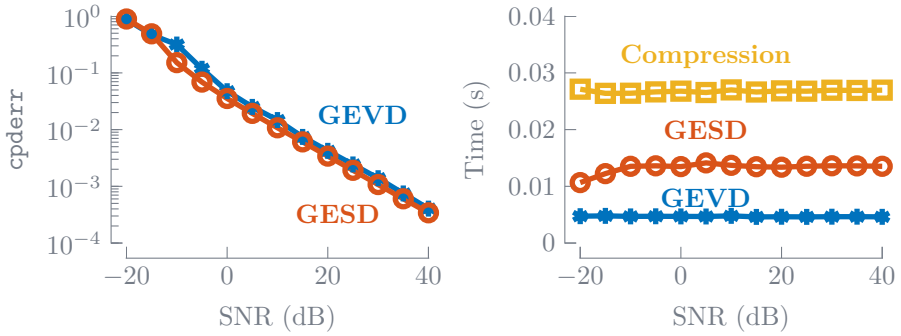

Figure 10.1: For normally distributed factor matrices, the GESD and GEVD methods have similar performance with respect to the accuracy of the computed factor matrices. Median (over 50 trials) `cpderr` (left) and computation time (right) for rank-10 tensors of dimensions $100 \times 100 \times 100$ with factor matrices with entries sampled from the standard normal distribution. The runtime of the GESD algorithm on the core tensor is about three times that of the GEVD algorithm on the core; however, overall runtime is dominated by the compression step.

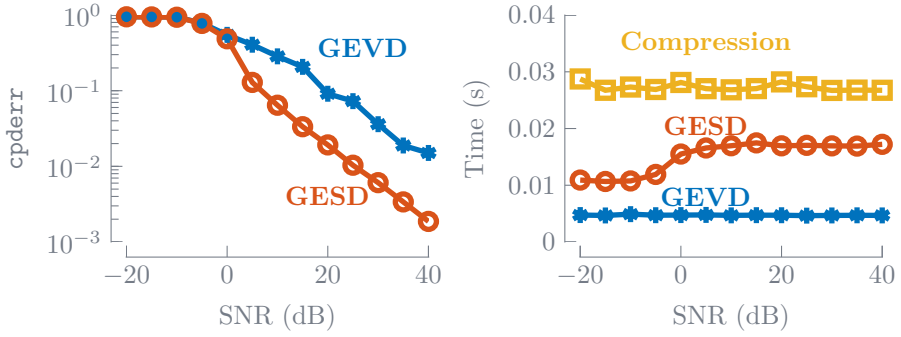

Figure 10.2: For factor matrices with entries sampled from the uniform distribution on $[0,1]$, the GESD method obtains a significantly higher accuracy for the estimated factor matrices than the GEVD method. Median (over 50 trials) `cpderr` (left) and computation time (right) for rank-10 tensors of dimensions $100 \times 100 \times 100$ with factor matrices sampled from the uniform distribution on $[0,1]$. As for the case with normally distributed factor matrix entries, the compression step is more expensive than both the GEVD and the GESD algorithms themselves.

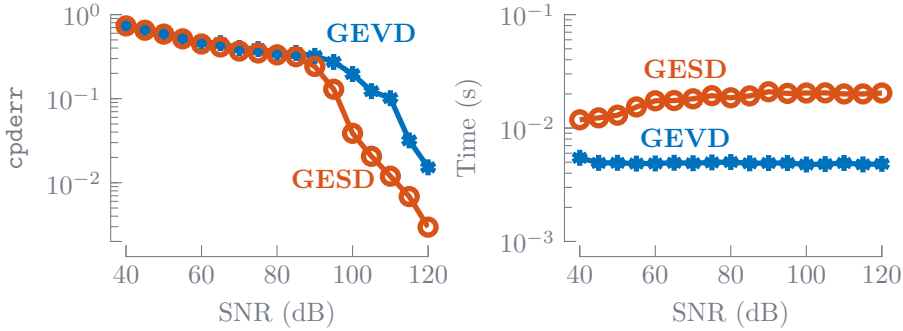

Figure 10.3: For factor matrices with correlated columns, the GESD method performs significantly better than the GEVD method with respect to the factor accuracy. Median (over 50 trials) `cpderr` (left) and computation time (right) for rank-10 tensors of dimensions $10 \times 10 \times 10$ with factor matrices with correlated columns (10° angles). Because of the difficulty of the problem only very high SNR values lead to a decent solution. The computation time of the GESD method is about three times that of the GEVD method.

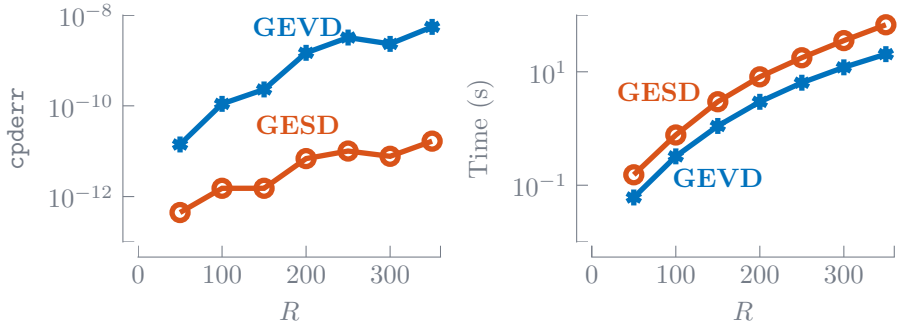

Figure 10.4: The relative time difference between the GESD and GEVD methods remains about the same as tensor rank increases. Median (over 20 trials) `cpderr` and computation time for exact rank-R tensors of dimensions $R \times R \times R$ with factor matrices sampled from the uniform distribution on $[0, 1]$. The GESD method obtains factors with a higher accuracy than the GEVD method for all tensor dimensions. In this experiment we use a splitting threshold of $5/R$.

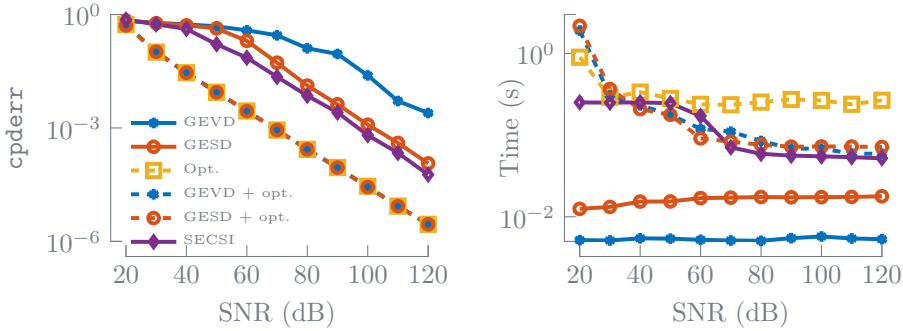

Figure 10.5: Initializing an optimization algorithm with the GESD or GEVD solution is faster than starting from a random initialization. Median (over 20 trials) `cpderr` (left) and computation time (right) for rank-10 tensors of dimensions $10 \times 10 \times 10$ with factor matrices sampled from the uniform distribution on $[0, 1]$. All initializations lead to the same solution. Optimization after initializing with the more accurate GESD solution is equally as fast as optimization after initializing with the GEVD solution.

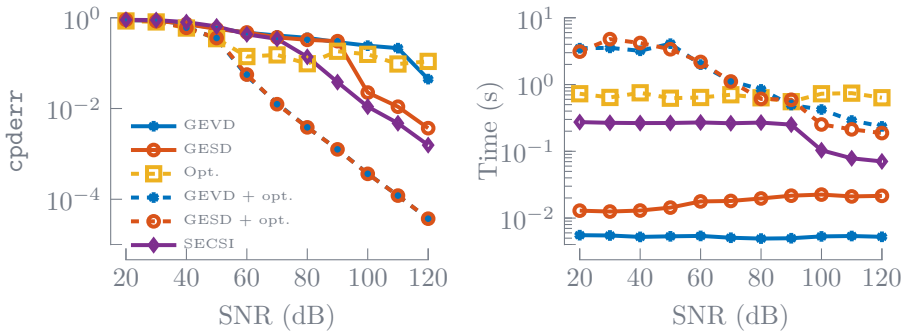

Figure 10.6: For factor matrices with correlated columns, the GESD solution is more accurate than the GEVD solution. An optimization algorithm initialized with the GESD solution is faster than when initialized with the GEVD solution. Median (over 20 trials) `cpderr` (left) and computation time (right) for rank-10 tensors of dimensions $10 \times 10 \times 10$ with correlated factor matrix columns (10° angles). Initializing the optimization algorithm randomly does not lead to a good solution.

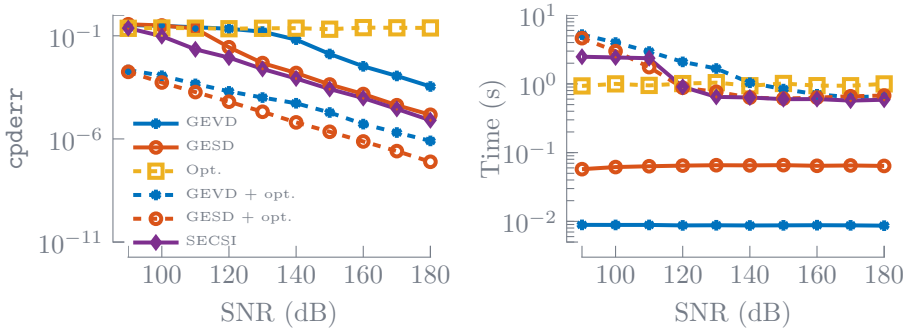

Figure 10.7: For sufficiently difficult problems, initialization with GESD leads to both a more accurate solution and lower computation time of the optimization algorithm compared to initialization with GEVD. Median (over 100 trials) `cpderr` (left) and computation time (right) for rank-20 tensors of dimensions $20 \times 20 \times 20$ with correlated factor matrix columns (8° angles). A splitting threshold of 0.02 was used in this experiment.

10.4.4 GESD and GEVD vs. noiseless problems

We now illustrate an additional advantage of GESD compared to GEVD. Namely there is no fixed strategy for pencil selection in GEVD which is able to solve all problems meeting the assumptions of the algorithm even in the noiseless setting. As an example, let $\mathcal{T}$ be the (real) rank-3 tensor with frontal slices

$$\mathbf{T}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{T}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{pmatrix} \quad \mathbf{T}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1/5 \\ 0 & 1/5 & 0 \end{pmatrix}.$$

As previously discussed, a popular strategy for pencil selection in GEVD is to use the first two slices of a core for the MLSVD of $\mathcal{T}$. The tensor $\mathcal{S}$ with frontal slices

$$\mathbf{S}_1 \approx -\begin{pmatrix} 1.39 & 0 & 0 \\ 0 & 1.11 & 0 \\ 0 & 0 & 1.11 \end{pmatrix} \quad \mathbf{S}_2 \approx \begin{pmatrix} -0.25 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 0.16 \end{pmatrix} \quad \mathbf{S}_3 \approx -\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1/5 \\ 0 & 1/5 & 0 \end{pmatrix}$$

is a core for the MLSVD of $\mathcal{T}$. If one uses the first two slices of $\mathcal{S}$ to compute the CPD of $\mathcal{T}$ via GEVD, then one must compute the eigenvectors of the matrix

$$\mathbf{S}_1^{-1}\mathbf{S}_2 \approx -\begin{pmatrix} 0.181 & 0 & 0 \\ 0 & 0.143 & 0 \\ 0 & 0 & 0.143 \end{pmatrix}.$$

Since $\text{span}\{\mathbf{e}_2, \mathbf{e}_3\}$ is an eigenspace of dimension 2 for this matrix corresponding to the eigenvalue -0.143, there is an ambiguity in which eigenvectors should be used. Tensorlab, for example, assumes $\mathbf{e}_2$ and $\mathbf{e}_3$ are the correct eigenvectors to take in this subspace and proceeds from there. Instead considering the matrix pencil $(\mathbf{S}_1, \mathbf{S}_3)$ shows that $\mathbf{e}_2 + \mathbf{e}_3$ and $\mathbf{e}_2 - \mathbf{e}_3$ are the correct eigenvectors to take from the subspace $\text{span}\{\mathbf{e}_2, \mathbf{e}_3\}$.

To make matters worse for GEVD, if one proceeds under the assumption that $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ are the correct eigenvectors to take, then the GEVD based solution one arrives at is a local optima for the optimization based approach for computing a CPD. Thus using this GEVD based initialization followed by optimization cannot yield the true rank-3 decomposition of $\mathcal{T}$.[11]

GESD, on the other hand, is easily able to compute the decomposition of this tensor. Although GESD still uses the pencil $(\mathbf{S}_1, \mathbf{S}_2)$ on its first iteration, GESD recognizes that the eigenspace $\text{span}\{\mathbf{e}_2, \mathbf{e}_3\}$ cannot be decomposed with this pencil and delays the decomposition to a later pencil.

Examined in detail, from the pencil $(\mathbf{S}_1, \mathbf{S}_2)$, GESD computes

$$\mathbf{Z}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \text{and} \qquad \mathbf{Z}_2 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and decomposes $\mathcal{S} = \mathcal{S} \cdot_2 \mathbf{Z}_1^{\mathrm{T}} + \mathcal{S} \cdot_2 \mathbf{Z}_2^{\mathrm{T}}$. The tensor $\mathcal{S} \cdot_2 \mathbf{Z}_1^{\mathrm{T}}$ has rank 1 and GESD extracts one rank-1 component of $\mathcal{T}$ from this tensor, while $\mathcal{S} \cdot_2 \mathbf{Z}_2^{\mathrm{T}}$ is a rank-2 tensor of size $3 \times 2 \times 3$. Letting $\mathcal{A}$ denote the core of a $2 \times 2 \times 2$ MLSVD of $\mathcal{S} \cdot_2 \mathbf{Z}_2^{\mathrm{T}}$, one has

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 1.12 \\ 1.12 & 0 \end{pmatrix} \qquad \mathbf{A}_2 = \begin{pmatrix} 1/5 & 0 \\ 0 & 1/5 \end{pmatrix}.$$

The generalized eigenvalues of the matrix pencil $(\mathbf{A}_1, \mathbf{A}_2)$ are $\text{span}((1.12, 1/5))$ and $\text{span}((-1.12, 1/5))$ with corresponding generalized eigenvectors $\mathbf{e}_1 + \mathbf{e}_2$ and $\mathbf{e}_1 - \mathbf{e}_2$, respectively. Since this matrix pencil has distinct generalized eigenvalues, GESD successfully extracts the remaining rank-1 components of $\mathcal{T}$ from the pencil.

[11] The GEVD based decomposition has approximately 0.516 factor matrix error when compared to the correct decomposition.

10.5 Accuracy of GESD versus deterministic bound

In this section we compare the deterministic bound computed in Section 10.3 for the existence of two distinct eigenvalue clusters against the accuracy of the first step of our GESD algorithm. We note that the bound computed from the original tensor only provides information about the first step of our algorithm. For this reason, we omit a direct comparison of this bound to the accuracy of the GESD algorithm.

Let $\mathcal{T} \in \mathbb{R}^{R\times R\times K}$ be a real rank-R tensor with CPD $[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$, and let $\mathcal{N}$ be a noise tensor. As previously discussed, given $\mathcal{T}+\mathcal{N}$ as an input, the goal of the first step of the GESD algorithm is to robustly decompose $\mathbb{R}^R$ into (at least) two disjoint JGE spaces of $\mathcal{T}$.

Suppose $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are matrices such that the range of each $\mathbf{Z}_i$ is a JGE space of $\mathcal{T}$ and such that the matrix $(\mathbf{Z}_1\ \mathbf{Z}_2)$ is invertible. Recall that the $\mathbf{Z}_i$ have the form $\mathbf{Z}_i = \mathbf{B}^{-\mathrm{T}}\mathbf{E}_i$ where $\mathbf{E}_1$ and $\mathbf{E}_2$ are matrices which for all $j=1,\dots,R$ satisfy that the jth row of $\mathbf{E}_1$ is nonzero if and only if the jth row of $\mathbf{E}_2$ is a zero row. Using this fact we may evaluate the robustness of the first step of our algorithm in two ways.

Let $\tilde{\mathbf{Z}}_1$ and $\tilde{\mathbf{Z}}_2$ be the matrices computed by the first step of our algorithm. Then one should have that the ranges of

$$\mathbf{B}^{\mathrm{T}}\tilde{\mathbf{Z}}_1 = \tilde{\mathbf{E}}_1 \qquad \text{and} \qquad \mathbf{B}^{\mathrm{T}}\tilde{\mathbf{Z}}_2 = \tilde{\mathbf{E}}_2.$$

are (approximately) orthogonal. Additionally, there should exist disjoint index sets $J_1, J_2 \subset \{1,\dots,R\}$ such that $J_1 \cup J_2 = \{1,\dots,R\}$ and such that

$$\operatorname{ran}\tilde{\mathbf{Z}}_1 \approx \operatorname{span}(\{\mathbf{b_j}\}_{\mathbf{j}\in\mathbf{J_1}}) \qquad \text{and} \qquad \operatorname{ran}\tilde{\mathbf{Z}}_{\mathbf{2}} \approx \operatorname{span}(\{\mathbf{b_j}\}_{\mathbf{j}\in\mathbf{J_2}})$$

where the $\mathbf{b_j}$ are columns of the matrix $(\mathbf{B}^{-\mathrm{T}})$. That is, the principal (largest) angle between the range of $\tilde{\mathbf{Z}}_i$ and $\operatorname{span}(\{\mathbf{b_j}\}_{\mathbf{j}\in\mathbf{J_i}})$ should be approximately zero.

In Figure 10.8, we plot the median (over 50 trials) of the aforementioned angles as a function of the SNR of $\mathcal{T}$ to $\mathcal{N}$ where $\mathcal{N}$ is randomly generated Gaussian noise. That is, for many choices of the SNR s, we randomly generate noise tensors $\mathcal{N}$ such that the SNR of $\mathcal{T}$ to $\mathcal{N}$ is equal to s. In each trial we compute subspaces $\tilde{\mathbf{Z}}_1$ and $\tilde{\mathbf{Z}}_2$ for the tensor $\mathcal{T}+\mathcal{N}$ using the GESD algorithm. We check accuracy by plotting the median over the trials of the angles described above. As before, $\mathcal{T}$ has been generated by independently sampling all entries of the factor matrices $\mathbf{A},\mathbf{B}$, and $\mathbf{C}$ uniformly on $[0,1]$ and forming the associated low-rank tensor.

The first image of Figure 10.8 shows the $R \times R \times R$ rank-R case with $R = 4$, while the case $R = 8$ is displayed in the second image. In addition to plotting the aforementioned angles, the bound from Corollary 10.3.1 for existence of two distinct eigenvalue clusters is plotted. The deterministic bound is computed by taking the largest bound found using either five or fifty different unitary matrices and the subpencil which gives the best bound is the subpencil used to compute generalized eigenspaces in the GESD algorithm.

We find that errors in GESD are well predicted by this bound. For SNRs above the deterministic bound, the first step of GESD is highly accurate, while small errors start to appear when the SNR decreases below the deterministic bound. For all SNR values, the inaccuracy in the orthogonality of the $\tilde{\mathbf{E}}_i$ and the inaccuracy in the angle between the $\tilde{\mathbf{Z}}_i$ and the columns of $\mathbf{B}^{-\mathrm{T}}$ are observed to have similar magnitude.

10.6 Conclusion

We have presented an algebraic generalized eigenspace based algorithm for CPD computation which uses multiple subpencils of a given tensor to compute a decomposition. A key idea in the GESD algorithm is that one may combine information from many subpencils of a tensor to compute its CPD, thus allowing one to only perform trustworthy computations in each pencil. The performance of GESD has been compared to that of the popular GEVD algorithm. We find that in all situations, GESD is at least as accurate as GEVD, and that GESD is typically significantly more accurate than GEVD. We see that GESD particularly shines in difficult problems where the factor matrices of a tensor are highly correlated.

While the GESD algorithm typically takes about three times as long to run on a core tensor as GEVD, in practical settings which involve compression steps and optimization steps, a CPD computation using GESD for algebraic initialization of optimization is at least as fast as a CPD computation using GEVD for algebraic initialization. We observed that GESD based initializations are often significantly more accurate than GEVD based initializations and that they can lead to more accurate and reliable optimization. We also illustrated that no single fixed pencil selection strategy for GEVD can solve all given noiseless CPD problems even when the initial tensor meets the assumptions of the GEVD algorithm. Since GESD intelligently determines which computations can be reliably performed with a given subpencil, GESD does not suffer from this issue.

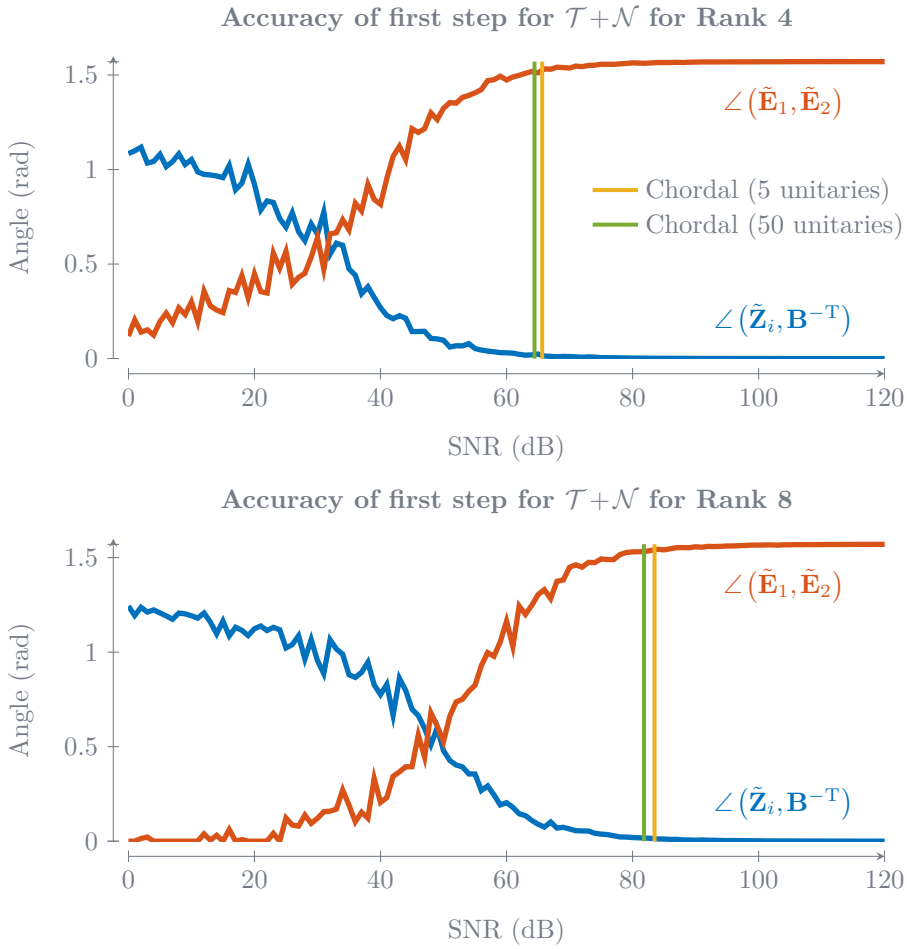

Figure 10.8: The angle between $\tilde{\mathbf{E}}_1$ and $\tilde{\mathbf{E}}_2$ stays close to $\pi/2$ until the bound from Corollary 10.3.1 is crossed, while the angle between $\tilde{\mathbf{Z}}_i$ and columns of $\mathbf{B}^{-\mathrm{T}}$ stays close to zero until this bound is crossed. Trying more unitary matrices shifts the bound sightly to the left. Mean (over 50 trials) angle between $\tilde{\mathbf{E}}_1$ and $\tilde{\mathbf{E}}_2$ and between $\tilde{\mathbf{Z}}_i$ and columns of $\mathbf{B}^{-\mathrm{T}}$ for a rank-4 tensor of dimensions $4\times4\times4$ (top) and for a rank-8 tensor of dimensions $8\times8\times8$ (bottom), picking the best unitary matrix out of either 5 or 50 trials. The SNR is varied from 0 dB to 120 dB.

In addition to experimental analysis, we provided a theoretical analysis of the GESD algorithm. Here we computed a deterministic bound for the existence of a subpencil which has two distinct clusters of generalized eigenvalues which have not coalesced. This bound may be interpreted as a theoretical guarantee that a next step in the GESD algorithm is able to compute generalized eigenspaces which are meaningfully related to the generalized eigenspaces of the original noiseless tensor.

Currently we assume the existence of reasonably sized eigenvalue gaps which may be used to compute at least two distinct generalized eigenspaces. While this assumption is reasonable for many tensors of low rank, this assumption may fail to hold for particularly ill-conditioned tensors or when the rank becomes large. Indeed, for generic tensors, the expected generalized eigenvalue gap becomes arbitrarily small as tensor rank and dimensions increase. Thus for tensors of sufficiently high rank, one should not expect to find a suitable gap. Resolving this is a direction for further research.

11 Conclusion

Tensors have gained a large interest in recent years, spurring the design of new tensor decompositions, the development of new algorithms to compute them and a whole array of new applications. In this thesis, we discussed a series of algorithms to make the use of tensor decompositions, as introduced in Chapter 2, applicable to a wider range of real-life applications. First, the updating methods designed in Chapters 3, 4 and 5 enable tensor data to be processed in real-time with low storage requirements and limited loss in accuracy. These methods make tensor analysis a viable approach for real-time applications such as patient monitoring, as discussed in Chapter A. Second, with the GGN methods from Chapters 6, 7 and 8 for the computation of CPDs with non-LS cost functions, one can efficientely compute more meaningful decompositions of tensors obtained from specific applications than those obtained with the LS cost function. Musical source separation and the analysis of fMRI signals are just two examples of applications that benefit from these methods. The latter was illustrated in Chapter 9. Finally, with the GESD CPD algorithm from Chapter 10, more accurate algebraic initializations can be obtained for optimization methods, leading to faster computations and/or more accurate CPDs. All algorithms from this thesis will be included in the upcoming release of Tensorlab 4.0, the newest version of the MATLAB toolbox, to make them available to researchers and practitioners around the world. A (nonexhaustive) preview of the included methods is given in Chapter C.

In the remainder of this chapter, we give an overview of our contributions in Section 11.1, and outline perspectives for future work in Section 11.2.

11.1 Contributions

Below, we give an overview of our contributions. For chapters that rely on results obtained in collaboration with other team members, we clearly indicate the contribution of each collaborator.

Chapter 3

In this chapter, we report the results of a collaboration with N. Vervliet. The doctoral candidate generalized a proof-of-concept for third-order tensors to Nth-order tensors. He enabled arbitrary additions of $(N-1)$th-order tensors to an Nth-order tensor, in any number of modes simultaneously. He also contributed the windowing strategy and performed the complexity analysis and the experiments. Finally, he generalized the method to a much wider class of tensor updating problems.

- *Memory-efficient CPD updating algorithm.* An efficient GN-based algorithm for updating the CPD of a nonstationary tensor was derived. Instead of keeping all entries of the tensor that have been processed in memory, only the factor matrices of its CPD are stored. When new $(N-1)$-th order tensors arrive that are to be added to the tensor, the factor matrices and the new additions are used to form an efficient representation of the new tensor. This representation can be seen as a structured tensor format which our algorithm exploits to achieve a low complexity in terms of memory and computations. The efficient implementation for the third-order case was contributed by N. Vervliet.
- *Tracking dynamic systems.* For nonstationary systems that output low-rank tensor data, the CPD can change over time. Using sliding and/or exponential windows, we can weight the past contributions according to their 'age'. A fast algebraic initialization strategy was derived as well, which is particularly effective when the factors change relatively slowly. By combining this algebraic initialization with the structure-exploiting algorithm and windowing strategies, new slices can be processed at significantly higher rates than when batch algorithms are used, as demonstrated for synthetic data.

Chapter 4

In this chapter, we report the results of a collaboration with M. Boussé and N. Vervliet. The adaptation of the weighted CPD algorithm to an updating context and its efficient implementation were performed by the doctoral candidate, as was the DOA experiment.

- *Memory-efficient WLS CPD updating algorithm.* If the weight tensor in the computation of a CPD in the WLS sense can be approximated well by a low-rank tensor, this can be exploited to obtain an efficient WLS CPD algorithm. By combining this approach with the structured updating

format from Chapter 3, a WLS updating method was derived that is efficient, both in time and in memory. M. Boussé contributed the batch weighted CPD algorithm. The adaptation of the weighted CPD algorithm to an updating context wherein any number of $(N-1)$th-order subtensors are added to an Nth-order tensor was performed by the doctoral candidate. N. Vervliet contributed to the efficient implementation of the method.

- *Tracking dynamic systems under variable conditions.* For nonstationary systems that output low-rank tensor data, the CPD can change over time, but the environmental variables can change as well. For instance, sensors can break down and suddenly give inaccurate readings, or the quality of certain data points can depend on the ambient temperature. In such cases, not every tensor entry should be weighted equally when a CPD of the tensor is computed. Using a WLS CPD updating scheme, both the changing tensor data and the changing environmental situation can be tracked and a representation of the underlying system is computed accurately and efficiently.

Chapter 5

- *(Truncated) MLUTVD.* The MLSVD of a tensor can be truncated to arbitrary multilinear rank so that it provides an LMLRA of the tensor which is close to optimal. However, one often merely wants to extract the principal subspace of rank R_n for each of the N tensor modes and truncate the MLSVD to a multilinear rank-$(R_1, R_2, \ldots, R_N)$-approximation of the tensor. In such cases, computing a full MLSVD is unnecessary. Instead, we proposed a new type of tensor decomposition called the rank-revealing MLUTVD. This decomposition generalizes the matrix UTV decomposition, wherein the middle matrix is upper- or lower-triangular. Like the truncated MLSVD, truncation of the MLUTVD can provide a multilinear rank-$(R_1, R_2, \ldots, R_N)$-approximation, but the latter can be computed more efficiently through the use of rank-revealing UTV decompositions instead of SVDs.

- *Bounded truncation error.* The truncation error of the MLUTVD of a tensor can be bounded in function of the supplied noise threshold. For each tensor mode, only the principal subspace in function of this threshold is kept, while all contributions in the orthogonal complement of the principal subspace are truncated. By combining these bounds, an overall bound for the truncated MLUTVD was obtained. Bounds were computed in both the Frobenius and spectral norms.

- *Efficient MLUTVD updating algorithm.* For nonstationary tensors, new entries are added at certain time intervals. Often, this new data arrives as an $(N-1)$th-order tensor that is to be added to the Nth-order tensor in a certain mode. An algebraic updating method was developed to efficiently update the MLUTVD of the tensor to take this new information into account. In every mode, the principal subspace is tracked using a matrix subspace tracking algorithm. If the mode is a non-evolving mode, the subspace tracking algorithm considers the addition of new columns via an SURV updating scheme. For the evolving mode, new rows arrive instead of columns and the principal subspace is tracked with the use of a rank-revealing URV updating method. To gradually forget the old tensor data, windowing strategies were provided. The effectiveness of the method was demonstrated on synthetic and real-life data.

Chapter 6

This chapter is the result of a collaboration with N. Vervliet. The doctoral candidate contributed to the derivation and efficient computation of the Hessian approximation for the GGN method to compute CPDs with non-LS cost functions. He also provided the proofs in the appendix and the strategies for handling indefiniteness of the Hessian approximation. In addition, he performed all experiments, including the hyperspectral imaging example.

- *GGN algorithm for non-LS CPD.* The GN method to compute the CPD of a tensor in LS sense was generalized to a GGN method that can handle arbitrary twice differentiable entry-wise cost functions. By exploiting the multilinear structure of the CPD, the Hessian approximation can be computed much more efficiently than when a naive approach is used. We showed that the GGN method outperforms state-of-the-art first-order methods, both in accuracy and computation time and requires much fewer iterations due to its faster convergence close to the solution. We also provided proofs that the efficient expressions for the blocks of the Hessian approximation lead to the same result as a naive computation. N. Vervliet contributed to the derivation of the expressions for the gradient and Hessian approximation.

- *CPD with β-divergence cost functions.* The class of β-divergence functions encompasses many of the most-used cost functions in practice, such as the LS distance, the KL divergence and the IS divergence. These cost functions can lead to better or worse results depending on the distribution of the tensor entries and the noise. We derived specialized routines to efficiently compute CPDs with β-divergence cost functions for arbitrary

values of β. We demonstrated, both on synthetic data and on real-life hyperspectral imaging data, that a better approximation error can be achieved by choosing a β-divergence cost function with the appropriate value of β over the LS cost function.

- *Strategies for handling indefiniteness of the Hessian approximation.* We discussed four strategies to avoid that the Hessian approximation becomes indefinite for certain β-divergence cost functions and fast convergence is impeded: modifying the tensor of second-order derivatives, tempering the value of β, using a weighted least-squares approximation and shifting the tensor. We showed that these methods are effective at reducing the computation time.

Chapter 7

This chapter is the result of a collaboration with N. Vervliet. The doctoral candidate contributed to the derivation and efficient computation of the Gramian-vector product that the inexact GGN method uses to compute CPDs with non-LS cost functions. He also derived the efficient preconditioners and the expressions for the Gramian-vector product in case that the tensor of second-order derivatives is approximated by an LMLRA. Finally, he applied the method to the specific case of the Rician-dedicated cost function and performed all experiments.

- *Inexact GGN algorithm for non-LS CPD.* An inexact GGN method to compute a CPD of a tensor with a non-LS cost function was derived, based on the work in Chapter 6. By using a CG approach, only Gramian-vector products need to be computed instead of the full Gramian, which leads to a method that scales much better to larger tensor dimensions, compared to both the exact GGN algorithm and existing first-order methods. N. Vervliet contributed to the derivation of the expressions for the gradient and Hessian approximation.

- *Strategies for speeding up the algorithm.* Several strategies were proposed to further decrease the computation time of the inexact GGN method. First, a Jacobi and a block-Jacobi preconditioner were designed to reduce the number of CG iterations that the inexact GGN method requires to converge to the solution. Due to careful exploitation of the multilinear structure of these preconditioners, their application is cheap in comparison to the Gramian-vector products themselves. We demonstrated that both preconditioners are effective in reducing the number of iterations and runtime of the GGN algorithm. Second, we showed that a significant

speedup can be obtained by compressing the tensor holding the second-order derivatives with the use of an LMLRA and exploiting the multilinear structure of this approximation. We demonstrated that the compression leads to faster iterations and, as long a the compression factor is not too large, does not negatively impact the convergence speed. Finally, we applied the method in a stochastic setting by embedding it in the RBS framework. By iteratively refining the CPD using small sampled subtensors of the full tensor, we demonstrated that the inexact GGN method can scale to dense tensors of even larger dimensions. The method was shown to outperform existing stochastic methods on a dense tensor with 125 million entries. N. Vervliet supplied the GN RBS CPD algorithm for LS cost functions, which was generalized to arbitrary cost functions by the doctoral candidate.

- *Computing a CPD for Rician distributed data.* For tensors with Rician distributed entries, using the LS cost function is not optimal. Such tensors are for example obtained from the magnitude of a complex signal where both the real and imaginary parts are collected independently with equal variance Gaussian noise. Instead of the LS cost function, a Rician-dedicated cost function should be applied. By using the inexact GGN CPD algorithm with this Rician-dedicated cost function, a better estimation of the underlying components can be obtained. The better performance of the Rician-dedicated cost function compared to the LS cost function was demonstrated in an experiment wherein synthetic fMRI sources are blindly separated. The generation of the complex fMRI data was performed by C. Chatzichristos, in the same manner as in Chapter 9.

Chapter 8

In this chapter, we report the results of a collaboration with N. Vervliet. The doctoral candidate derived the efficient expressions for the cost function, gradient, Hessian approximation and preconditioner for the GGN CPD algorithm for β-divergence cost function in the case where the rank of the CPD is one. He also performed the experiments.

- *Rank-1 inexact GGN CPD algorithm for β-divergence cost functions.* For the specific case of a rank-1 approximation of a tensor with a β-divergence cost function, the method of Chapter 6 was sped up significantly by deriving efficient expressions for the cost function, gradient, Hessian approximation and preconditioner. The memory requirements of the method are much lower than existing algorithms and, depending on the value of β, the method is also more accurate and/or faster.

Chapter 9

This chapter is the result of a collaboration with C. Chatzichristos. The doctoral candidate tailored the GGN CPD method of Chapter 6 to the specific case of a β-divergence cost function and applied the method to perform the blind source separation of synthetic fMRI sources.

- *Blind fMRI source separation.* In fMRI data analysis, it is typically the magnitude of the fMRI signal that is studied. However, both the real and complex parts of the data are collected separately and are independently perturbed by Gaussian noise. Consequently, using the LS cost function is not necessarily the best choice and other cost functions can potentially give better results. We showed that β-divergence cost functions can outperform the LS distance on simulated fMRI data. The generation of the simulated fMRI data and the computation of the source correlation were performed by C. Chatzichristos.

Chapter 10

The results in this chapter were obtained through a collaboration with E. Evert. The doctoral candidate contributed to the algorithm design, efficiently implemented the recursive algorithm and performed all experiments.

- *GESD-based algebraic CPD algorithm.* Despite their widespread use, algebraic pencil-based algorithms such as the GEVD algorithm are known to be unstable for the computation of the CPD of a tensor. The GEVD algorithm uses only one pencil to separate all generalized eigenvectors and obtain the resulting CPD, even if the associated generalized eigenvalues are not well-separated for that pencil. We showed that a much stabler algorithm can be obtained by using many different pencils instead of one and only splitting generalized eigenspaces corresponding to sufficiently well-separated sets of generalized eigenvalues. Consequently, the resulting GESD CPD algorithm is significantly more robust to noise than the GEVD algorithm. This manifests itself through a higher accuracy than the GEVD algorithm for the computation of the CPD, particularly on harder problems with correlated factor matrix columns, at the cost of only a limited increase in computation time. E. Evert contributed to the algorithm design.
- *Stability bounds for the GESD algorithm.* Probabilistic and deterministic bounds for the stability of the algorithm were derived. The probabilistic bound uses the Cramér-Rao induced bound for the mean-square angular

deviation of the vectors of a factor matrix to build confidence intervals for the generalized eigenvalues of a matrix pencil. These confidence intervals were used to ensure that the separation of the corresponding eigenspaces is well-justified in the GESD algorithm. The deterministic bound was obtained through the use of perturbation theory for joint generalized eigenvalues. We demonstrated empirically that both bounds are good predictors for the stability of the GESD algorithm and for the accuracy of the resulting CPD. The bounds and their proofs were contributed by E. Evert.

Chapter A

In this appendix, we report the results of a collaboration with A. Caicedo et al. The doctoral candidate provided the CPD updating algorithm and contributed to the experimental design.

- *CPD updating for brain hemodynamics monitoring in neonates.* CPD updating can be used for monitoring brain hemodynamics coupling in neonates. Similarity matrices for different segments of the signals were concatenated to form a tensor and the CPD updating algorithm was used to incrementally evaluate the impact of propofol in the coupling between the different signals. We showed that a drop in the interaction between signals can be distinguished due to the administration of propofol. A. Caicedo collaborated with the medical practitioners to obtain the data and was the main contributor to the experimental design.

Chapter B

The inexact GGN CPD algorithm for non-LS cost functions can be applied to tensors of arbitrary order. Efficient expressions were derived for the gradient, Jacobian, Hessian approximation, Gramian-vector product and the preconditioners.

Chapter C

In this appendix, some new features contributed to the upcoming Tensorlab 4.0 release by M. Boussé, N. Vervliet and the doctoral candidate are listed.

- *Preview of new features in Tensorlab 4.0.* After the success of Tensorlab 3.0 in 2016, a new release is in the works. Version 4.0 widens the applicability

of the toolbox to a larger range of real-world applications. Among other methods, the new features include algorithms for non-LS cost functions, such as low-rank WLS, contributed by M. Boussé, and β-divergence costs functions for the canonical polyadic decomposition (CPD). Updating algorithms for the CPD that allow both the tracking of streaming data and the incremental computation of the CPD of a large tensor are included as well. Additionally, an LS-CPD algorithm, contributed by M. Boussé and N. Vervliet, is provided to compute the CPD of a tensor that is only implicitly available as the solution of an underdetermined linear system.

11.2 Perspectives

The results in this thesis enable the use of tensor methods for a much wider range of applications. Still, there are several meaningful ways in which this work can be extended, both on the algorithmic and the application side. Below, we suggest some possible avenues that can be pursued in further research.

- *Other tensor decompositions and data fusion.* With the exception of the LMLRA updating algorithm in Chapter 5, all contributions in this thesis have been about the CPD. However, the LMLRA and other tensor decompositions such as the BTD, the TTD and the hierarchical Tucker decomposition have been proposed as well and are exceedingly used in practice. Data fusion, where multiple decompositions are combined by sharing (parts of) factors and structure is imposed, has also found numerous applications, for instance in biomedical signal processing. As for the CPD, extending both the updating and non-LS cost function methods to alternative decompositions and data fusion would open up a whole range of applications that currently cannot be adequately targeted.

- *Combining LMLRA and CPD updating.* While the CPD updating algorithms in this thesis are fast and accurate, previously truncated information cannot be taken into account when a rank increase is in order. By tracking an LMLRA such as the MLSVD or MLUTVD alongside the CPD, additional information from the previous updates can be recovered in the case of a rank increase.

- *Eigenspace-based bounds for algebraic CPD computation.* For the GESD CPD algorithm, bounds were derived that express the stability of the algorithm in function of the separation between sets of generalized eigenvalues. These bounds could be improved by also taking the separation into account between the associated generalized eigenspaces. By

computing these generalized eigenspace-based bounds before splitting off eigenspaces corresponding to sets of generalized eigenvalues, the stability of algebraic CPD algorithms could perhaps be increased even further.

- *Sparse data.* The work in this thesis mainly focuses on dense tensors. In many applications, however, tensor data is very sparse and applying methods for dense tensors is incredibly inefficient. For this reason, specialized algorithms have been designed for large-scale sparse tensors that use only the nonzero entries of the tensor and employ tools from high performance computing. Following these approaches, adapting the updating and non-LS cost function CPD methods from this thesis so that they only take the nonzero entries in the sparse tensor into account would allow them to be used in applications that generate huge, sparse tensors.

- *Updating in real-time monitoring applications.* The CPD updating algorithm has been shown to be successful for the incremental analysis of tensor data. However, it has not yet been tested in a true real-time setting, for instance in patient or process monitoring. Trying out the updating methods in real-time, for instance in epilepsy monitoring, is the next step on the road to inclusion of these methods in tools for practical use.

Monitoring of Brain Hemodynamics Coupling in Neonates using Updated Tensor Decompositions

A

ABSTRACT

In this chapter we explore the use of updated tensor decompositions for the monitoring of brain hemodynamics in neonates. For this study, we used concomitant measurements of heart rate, mean arterial blood pressure, arterial oxygen saturation, EEG, and brain oxygenation - measured using near-infrared spectroscopy. These measurements were obtained from 22 neonates undergoing an INSURE procedure (INtubation, SURfactant and Extubation) and sedation using propofol. To develop the monitoring framework using tensors, we used radial basis kernel function (RBF) to construct a similarity matrix for consecutive segments of the signals. These matrices were concatenated forming a tensor. Updating canonical polyadic decomposition was used to evaluate the impact of propofol in the coupling between the different signals. Results indicate, as previously reported, a drop in the interaction between signals due to propofol administration. This shows that tensor decompositions can be useful in order to monitor the coupling between different physiological signals.

This chapter is based on A. Caicedo, O. De Wel, M. Vandecappelle, L. Thewissen, A. Smits, K. Allegaert, L.. De Lathauwer, G. Naulaers, and S. Van Huffel, "Monitoring of Brain Hemodynamics Coupling in Neonates using Updated Tensor Decompositions," in 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2019), July 2019, pp. 660–663. The specific contributions of the doctoral candidate are listed in Section 11.1.

A.1 Introduction

Tensors are mathematical objects that generalize the concepts of vectors and matrices. Recently tensors have gained popularity due to their versatility, and because some problems are easily represented using these types of structure [35]. For instance, in clinical applications such as monitoring, tensors are attractive since the data can naturally be organized in a tensor. Moreover, tensor decomposition methods allow to store the information in a compact way, enabling the analysis of large datasets, such as the ones obtained from medical devices.

Normally, monitoring systems record different modalities simultaneously and visualize these on one screen. By analyzing the common dynamics of these signals, it is possible to infer the status of the patient. However, due to the nonstationary characteristics of human physiology, the interaction between these signals is likely to change in time. This type of changes can be tracked using tensors. Nevertheless, since monitoring devices are always acquiring new data, the storage of the information in the tensor structure can be challenging; therefore, updating tensor decomposition techniques should be considered. These techniques do not process all the data at once, but sequentially, to update the tensor decomposition. For instance, in [212] a canonical polyadic decomposition (CPD) updating algorithm is proposed. This algorithm sequentially updates the decomposition of a tensor using as input the CPD of the initial tensor. New slices corresponding to the new measurements are added sequentially to update the CPD, thereby exploiting the CPD structure of the data tensor as much as possible.

In this study, we use CPD updating for the monitoring of brain hemodynamics in neonates. More specifically we investigate the effect of the administration of propofol in neonates. Propofol is an anesthetic that is used in neonates. It has a rapid onset and a short duration. This model has been chosen because we have previously reported in several studies the effects of propofol on the neonates [181, 216]. We have observed that as side effects propofol induces a decrease in mean arterial blood pressure (MABP) [216], as well as a decoupling of the common dynamics between different signals [99], and a reduction in the aEEG background pattern during the first hour after propofol administration [203]. We use the results from these previous studies as a reference to compare the results provided by the CPD updating.

A.2 Methods

A.2.1 Data

The dataset consists of measurements from 22 neonates undergoing an INSURE (INtubation, SURfactant and Extubation) procedure. It contains concomitant measurements of heart rate (HR), MABP, arterial oxygen saturation (SaO2), EEG, and brain oxygenation (rScO2) – measured by means of Near-InfraRed Spectroscopy (NIRS). The data is a subsample from the data reported in [181], wherein 50 neonates were sedated using propofol as part of an endothracheal intubation procedure. The complete dataset was recorded at the Neonatal Intensive Care Unit of the University Hospitals Leuven, Gasthuisberg. Ethical approval was provided from the same institution. The trial was registered on ClinicalTrials.gov NCT01624373. From the 50 subjects, data from only 22 subjects could be used for this study, due to data quality and availability. As part of the protocol, all 22 included neonates underwent sedation using propofol, and the measurements were recorded up to 10 hours after administration. The postmenstrual age (PMA) of the neonates expressed in median (range) was 30 $(26-35)$ weeks, and the dose of propofol was 1.0 $(0.5-4.5)$ mg/kg. The data were split in three groups based on the PMA as follows: extremely preterm (5 neonates) with a PMA <28 weeks, very preterm (13 neonates) 28 weeks $<$ PMA $<31+6/7$ weeks, and moderate to late preterm (4 neonates) 32 weeks $<$PMA$<36+6/7$ weeks. The HR, MABP, and SaO2 were measured with IntelliVue MP70 (Philips, Eindhoven, The Netherlands), the NIRS signal was measured with INVOS 5100 (Covidien, Mansfield, Massachusetts), and the EEG was measured between the C3 and C4 electrodes, according to the $10-20$ system with a cerebral function monitor (Olympic CFM 6000, Natus); see [99, 181].

A.2.2 Preprocessing

The MABP, HR, SaO2, and rScO2 were recorded with a sampling frequency (F_s) of 1 Hz, while the EEG signal was recorded using a $F_s = 100$ Hz. The signals were preprocessed by computing their root mean square (RMS) value in a sliding window of one-minute length with overlapping of 59 seconds, providing a new RMS value every second. We used the same preprocessing scheme for the EEG. In this way the surrogate signal representing the EEG activity has the same sampling frequency as the other signals. From each neonate we analyzed 6 hours of data after the administration of propofol, since it is assumed that by that time the effect of propofol in the neonates is negligible [181, 216]. In Figure A.1 measurements, after preprocessing, from a representative subject are shown.

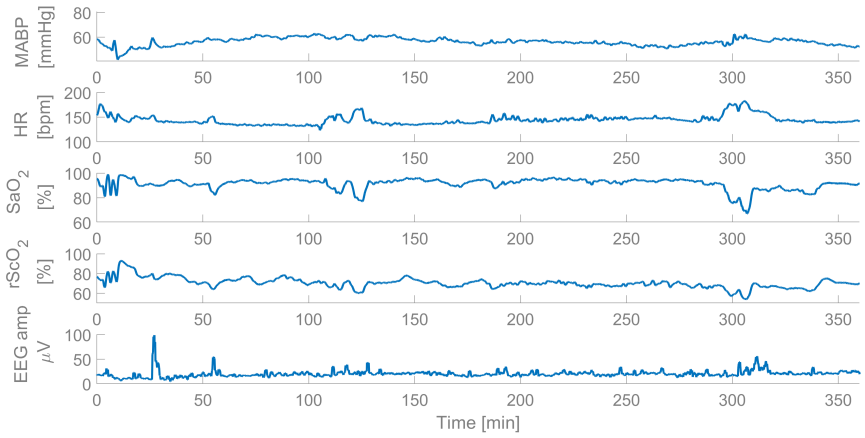

Figure A.1: Concomitant measurements from a characteristic neonate after propofol administration. The signals are shown after the preprocessing step.

A.2.3 Algorithm

To simulate a real monitoring scheme, we divided the data in consecutive segments of 5 minutes, having an overlap of 4 minutes and 50 seconds, i.e., every 10 seconds we process the last 5 minutes of data. The similarity between the different signals was computed using an RBF kernel as follows:

$$S(i,j) = \exp\left(\frac{-(\|x_i - x_j\|_2^2)}{\sigma^2}\right),$$

where $S(i,j)$ represents the similarity measurements between the signals x_i and x_j, and σ^2 is the kernel bandwidth. After computing the similarity measurement from each segment of the recorded signals, we obtain a 5×5 matrix. In order to find an adequate parameter for σ^2 in the RBF kernel function, we concatenated all kernel matrices $\mathbf{S}$, obtained from the different signal segments and all subjects. We computed the σ^2 value maximizing the Shannon entropy of the resulting enlarged matrix, and used it as a fixed parameter for further analysis. By computing the kernel bandwidth in this way, we guarantee maximum excursion in the values computed with the RBF kernel.

To initialize the algorithm we used the first 10 minutes of data, which contain 31 different segments, producing 31 5×5 similarity matrices. These matrices are stacked as frontal slices of a third order tensor, forming an initial tensor structure of dimensions $5 \times 5 \times 31$. Since the similarity matrices are symmetric, the first and second mode of the resulting tensor represent the same information,

the similarity between one signal modality and the others. The third mode in the tensor represents the changes in time of the similarity measurements among the signals, and it is used as a monitoring variable. We used a rank-one CPD in order to retrieve information about the changes in the similarity measurement among the different signals, represented in the third mode of the decomposed tensor. New data is added to the tensor as new slices in its third mode. To avoid the tensor growing in this way and taking into account the limited computational resources, we used tensor decomposition updating proposed in [212]. This algorithm only requires the matrices obtained from the CPD of the initial tensor. These matrices initialize the decomposition as specified in [212]. After updating the decomposition of the new enlarged tensor, we obtain the updated third mode, which is used as a monitoring variable, and represents the changes in the similarity measurements between the different signals. In summary, the monitoring algorithm consists of the following steps:

1. Computing the RMS value of the signals using a sliding one-minute window.
2. Segmentation of the data using five-minute windows, with an overlap of 4 minutes and 50 seconds.
3. Computation of the similarity matrices using the RBF kernel, and construction of the initial tensor structure, using the first 10 minutes.
4. CPD rank-one of the initial tensor.
5. Generation of the new slices of the tensor using the available data from the next 10 seconds.
6. Computation of the CPD updating, as specified in [212].
7. The algorithm repeats as long as new data is acquired

A.3 Results

In Figure A.2, the factor in the third mode of the tensor decomposition is shown for all subjects. The results are presented as the median valuse (gray solid line), and the 25 and 75 percentiles (shadow area). In order to detect the return to baseline level we performed a statistical analysis using the Kruskal–Wallis test on the time series obtained from the median value (gray solid line in Figure A.2). For the test the data was segmented in two parts, an initial part until time t, and the rest of the time series. We identify the return to baseline level as the time t when the two data segments were not statistically different ($p > 0.05$)

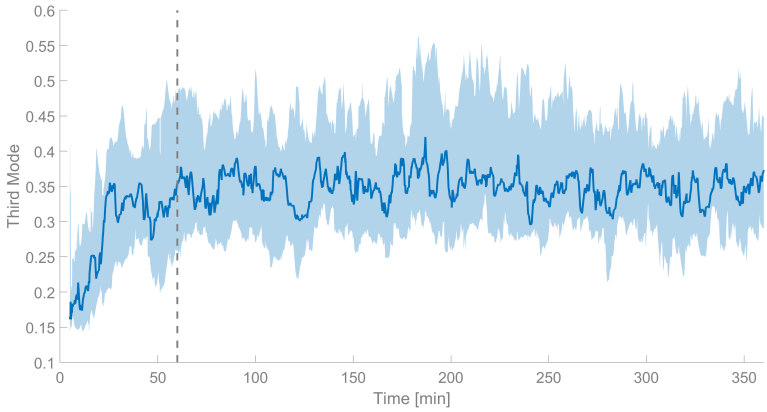

Figure A.2: Third mode of the decomposed tensor represented as the median value (black solid line), and the 25 and 75 percentile (shadow area). This mode indicates the changes in time of the interaction among the signals. The dashed line represents the time when the signal coupling returns to the baseline level.

anymore. For the complete dataset, we obtained a return to baseline level in 59 minutes, indicated in Figure A.2 as the dashed line.

In Figure A.3, the factor in the third mode of the tensor decomposition per patient group is depicted. From left to right, the results show the third mode for the extremely premature, very premature and the late premature neonates. The dashed line indicates the stabilization time, for each group, as presented in Figure A.2. We performed the same statistical analysis to identify the stabilization times in the three different groups and we found a stabilization time of 114, 26 and 28 minutes for the extremely premature, very premature and late premature population. The figure shows that the results from the extremely premature children present a larger value in magnitude, as well as a larger variability. It also suggests that the stabilization time for the extremely premature neonates is larger than the one computed for the complete patient group. The results for the very premature and late premature neonates are more similar in magnitude and present a lower variability between the subjects. But, their stabilization time is smaller than the stabilization time computed from the complete population.

We also studied the relationship between the propofol dose and the drop depth in the coupling between the variables, indicated by the third mode of the tensor decomposition. However, no relation was found between these variables. This is not unexpected, since no pharmacokinetics data was included in this study.

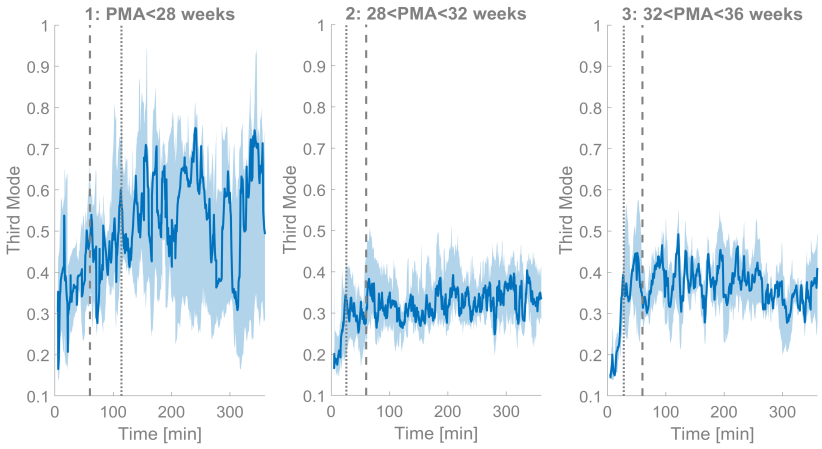

Figure A.3: Third mode of the decomposed tensor for the three different groups. The dashed line represents the stabilization time for the complete dataset, while the dotted line represents the stabilization time for each group.

A.4 Discussion

We have shown that CPD updating can be used in order to track the changes in the common dynamics between different signal modalities. By using a similarity matrix, obtained by means of an RBF function, it is possible to estimate the coupling between different time series. Changes in the coupling between these variables can be tracked by stacking the similarity matrix from different time segments, and using an appropriate tensor decomposition. Since the aim is to use this methodology for real-time monitoring, the tensor decomposition needs to be updated.

The results obtained using updating of the tensor decomposition indicate that the common dynamics of the signals are disrupted during the first hour after propofol administration. Previous results reported by our group have shown a stabilization time of 90 minutes [99]. Even though these results suggest a smaller stabilization time they are in agreement with clinical results presented in [83, 181, 212], where a drop in MABP was observed within the first hour after propofol administration. Since the drop in MABP is one of the main contributions to the decoupling between the different signal modalities, as shown in [99], the results provided by CPD updating might reflect this.

Additionally, different dynamics in the third mode of the tensor decomposition were observed among different patient groups. The extremely premature group

PMA $<$ 28 weeks presents not only a higher magnitude and variability in the monitoring variable, but also seems to present a slower return to baseline level (114 minutes). Moreover, the very preterm and late preterm groups exhibits a faster recovery time, 26 and 28 minutes respectively. These results are similar to the ones presented in [99]. However, in [99] the extremely premature subjects exhibit a behavior closer to the one reported for very premature subjects, whilst in this study their behavior was closer to the oldest group. Whether this is a maturational effect or pharmacokinetics (PK) driven effect needs further analysis.

We also analyzed the changes produced in the first and second mode during the updating. However, they do not seem to vary in a specific pattern. We hypothesize that these changes reflect the relation between one variable and the others; however, further research in the interpretation of these modes is needed. Due to space limitations we could not include a figure of these changes.

It is important to notice that in [99] a dynamic graph was used in order to monitor the coupling between different signal modalities, and assess the changes in their dynamics by means of graph metrics. These dynamic graphs can be related to tensor decompositions by studying these as a multi-way structure of a time-evolving network, as shown in [71].

One important drawback of this study is the lack of reference values prior to propofol administration. Since at least 5 minutes of data are required in order to produce a baseline value, it is difficult in this kind of procedures to obtain these data. However, it has been reported that the effects of propofol after 6 hours are negligible [99]. For this reason, we analyze only the first 6 hours after propofol administration.

A.5 Conclusion

We have shown that CPD tensor updating can be used for monitoring the coupling between different signal modalities. The results obtained in this study are similar to the results reported previously by our group, where it was found that propofol induces a disruption of the coupling between the common dynamics of the different signals. In addition, these results suggest also a link between tensor decompositions and dynamic graphs. This link has been previously reported in the literature for the analysis of social networks [71], but here we show their relation in the field of network physiology.

B Supplementary materials: Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions

The discussion in Chapter 7 is limited to third-order tensors for simplicity. Below, expressions are given for higher-order tensors.

Notation

For Nth-order tensors, the modes have dimensions I_n and corresponding factor matrices $\mathbf{U}^{(n)}$, for $1 \leq n \leq N$. The partial results have an index n instead of the letters $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. The CPD is written as $[\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$.

For convenience, we repeat that the LS cost function for the CPD can be generalized to

$$f(\mathbf{z}) = \sum_{n=1}^{I_1 I_2 \ldots I_N} f_n(m_n(\mathbf{z})),$$

with $m_n(\mathbf{z})$ the CPD model mapping the variables in $\mathbf{z}$ to the tensor elements in $\mathcal{M}$.

Gradient, Jacobian and Hessian

In Section 7.2.1, the form of the gradient, Jacobian and Hessian approximation matrices are derived. These expressions can be generalized to higher-order tensors as follows.

Gradient The nth part of the gradient $\mathbf{g} = \left[\mathbf{g}^{(1)}; \mathbf{g}^{(2)}; \ldots; \mathbf{g}^{(N)}\right]$ can be computed as

$$\mathbf{g}^{(n)} = \operatorname{vec}\left(\mathbf{R}_{(n)}\left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \mathbf{U}^{(n-2)} \odot \cdots \odot \mathbf{U}^{(1)}\right)\right),$$

with $\mathcal{R} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ the tensor holding all entry-wise first-order derivatives $\frac{\partial f_k}{\partial m_k}$, with $1 \leq k \leq I_1 I_2 \ldots I_N$.

Jacobian The nth part of the Jacobian $\mathbf{J} = \left[\mathbf{J}^{(1)}; \mathbf{J}^{(2)}; \ldots; \mathbf{J}^{(N)}\right]$ can be obtained as follows:

$$\mathbf{J}^{(n)} = \mathbf{\Pi}^{(n)^{\mathrm{T}}} \left(\left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \mathbf{U}^{(n-2)} \odot \cdots \odot \mathbf{U}^{(1)} \right) \otimes \mathbf{I}_{I_n} \right).$$

Hessian approximation As for the third-order case, the Hessian approximation is of the form $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, with $\mathbf{Z} = \mathrm{Diag}\left(\mathrm{vec}\left(\mathcal{D}\right)\right)$. The tensor $\mathcal{D} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ holds all entry-wise second-order derivatives $\frac{\partial^2 f_k}{\partial m_k^2}$, with $1 \leq k \leq I_1 I_2 \ldots I_N$.

Computing $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ efficiently

In Section 7.2.2, an efficient approach is given to evaluate $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$. The full vector $\mathbf{y} = \left[\mathbf{y}^{(1)}; \mathbf{y}^{(2)}; \ldots; \mathbf{y}^{(N)}\right]$ can be obtained by combining the partial results $\mathbf{y}^{(n)}$:

$$\mathbf{y}^{(n)} = \mathbf{J}^{(n)^{\mathrm{T}}}\mathbf{Z}\mathbf{J}^{(1)}\mathbf{x}^{(1)} + \mathbf{J}^{(n)^{\mathrm{T}}}\mathbf{Z}\mathbf{J}^{(2)}\mathbf{x}^{(2)} + \ldots + \mathbf{J}^{(n)^{\mathrm{T}}}\mathbf{Z}\mathbf{J}^{(N)}\mathbf{x}^{(N)}.$$

Following the methodology in the article, exploiting the structure of $\mathbf{J}$ and $\mathbf{Z}$ allows one to evaluate $\mathbf{y}^{(n)}$ as follows:

$$\mathbf{y}^{(n)} = \mathrm{vec}\left(\mathbf{Q}_{(n)} \left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(n+1)} \odot \mathbf{U}^{(n-1)} \odot \mathbf{U}^{(n-2)} \odot \cdots \odot \mathbf{U}^{(1)} \right)\right),$$

wherein $\mathcal{Q} = \mathcal{D} * \mathcal{E}$. The tensor $\mathcal{E}$ is a polyadic tensor with NR terms that can be written as follows:

$$\mathcal{E} = \left[\!\left[\mathbf{E}^{(1)}, \mathbf{E}^{(2)}, \ldots, \mathbf{E}^{(N)} \right]\!\right],$$

with

$$\mathbf{E}^{(n)} = \Big[\underbrace{\mathbf{U}^{(n)} \quad \cdots \quad \mathbf{U}^{(n)}}_{n-1 \text{ times}} \quad \mathbf{X}^{(n)} \quad \underbrace{\mathbf{U}^{(n)} \quad \cdots \quad \mathbf{U}^{(n)}}_{N-n \text{ times}} \Big].$$

Approximating $\mathcal{D}$

In Section 7.2.3, it is shown that $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ can be evaluated more efficiently when an LMLRA $\left[\!\left[\mathcal{S}; \mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(N)} \right]\!\right]$ of $\mathcal{D}$ is available. This is done through computation of the matrices $\mathbf{Y}^{(n)} = \mathrm{unvec}\left(\mathbf{y}^{(n)}\right)$. These can be obtained as follows:

$$\mathbf{Y}^{(n)} = \sum_{s=1}^{NR} (\mathbf{E}_s^{(n)} \mathbf{V}^{(n)}) \mathbf{S}_{(n)} \Bigg(\Bigg[\left(\mathbf{E}_s^{(N)} \mathbf{V}^{(N)} \right)^{\mathrm{T}} \mathbf{U}^{(N)} \Bigg] \odot \Bigg[\left(\mathbf{E}_s^{(N-1)} \mathbf{V}^{(N-1)} \right)^{\mathrm{T}} \mathbf{U}^{(N-1)} \Bigg]$$

$$\odot \cdots \odot \Bigg[\left(\mathbf{E}_s^{(n+1)} \mathbf{V}^{(n+1)} \right)^{\mathrm{T}} \mathbf{U}^{(n+1)} \Bigg] \odot \Bigg[\left(\mathbf{E}_s^{(n-1)} \mathbf{V}^{(n-1)} \right)^{\mathrm{T}} \mathbf{U}^{(n-1)} \Bigg]$$

$$\odot \Bigg[\left(\mathbf{E}_s^{(n-2)} \mathbf{V}^{(n-2)} \right)^{\mathrm{T}} \mathbf{U}^{(n-2)} \Bigg] \odot \cdots \odot \Bigg[\left(\mathbf{E}_s^{(1)} \mathbf{V}^{(1)} \right)^{\mathrm{T}} \mathbf{U}^{(1)} \Bigg] \Bigg),$$

in which $\mathbf{E}_s^{(n)} = \mathrm{Diag}\left(\mathbf{e}_{:s}^{(n)} \right)$.

Preconditioners

Dedicated and efficient preconditioners are given in Section 7.2.4 for solving the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ through the conjugated gradients or MINRES algorithm.

Block-Jacobi preconditioner The N blocks on the diagonal of the block-Jacobi preconditioner can be computed as follows:

$$\mathbf{G}_{\mathrm{nz}}^{(n,n)} = \mathbf{D}_{(n)} \Big(\left(\mathbf{U}^{(N)} \odot^{\mathrm{T}} \mathbf{U}^{(N)} \right) \odot \left(\mathbf{U}^{(N-1)} \odot^{\mathrm{T}} \mathbf{U}^{(N-1)} \right) \odot \cdots \odot \left(\mathbf{U}^{(n+1)} \odot^{\mathrm{T}} \mathbf{U}^{(n+1)} \right)$$

$$\odot \left(\mathbf{U}^{(n-1)} \odot^{\mathrm{T}} \mathbf{U}^{(n-1)} \right) \odot \left(\mathbf{U}^{(n-2)} \odot^{\mathrm{T}} \mathbf{U}^{(n-2)} \right) \odot \cdots \odot \left(\mathbf{U}^{(1)} \odot^{\mathrm{T}} \mathbf{U}^{(1)} \right) \Big).$$

Jacobi preconditioner The Jacobi preconditioner is a diagonal matrix with the vector $\mathbf{m}_{\mathrm{diag}}$ as diagonal. This vector can be computed as:

$$\mathbf{m}_{\mathrm{diag}} = \begin{bmatrix} \mathrm{vec}\left(\mathbf{D}_{(1)} \left(\left(\mathbf{U}^{(N)^{\bullet 2}} \right) \odot \left(\mathbf{U}^{(N-1)^{\bullet 2}} \right) \odot \cdots \odot \left(\mathbf{U}^{(2)^{\bullet 2}} \right) \right) \right) \\ \vdots \\ \mathrm{vec}\left(\mathbf{D}_{(n)} \left(\left(\mathbf{U}^{(N)^{\bullet 2}} \right) \odot \left(\mathbf{U}^{(N-1)^{\bullet 2}} \right) \odot \cdots \odot \left(\mathbf{U}^{(n+1)^{\bullet 2}} \right) \right. \right. \right. \\ \left. \left. \left. \odot \left(\mathbf{U}^{(n-1)^{\bullet 2}} \right) \odot \left(\mathbf{U}^{(n-2)^{\bullet 2}} \right) \odot \cdots \odot \left(\mathbf{U}^{(1)^{\bullet 2}} \right) \right) \right) \\ \vdots \\ \mathrm{vec}\left(\mathbf{D}_{(N)} \left(\left(\mathbf{U}^{(N-1)^{\bullet 2}} \right) \odot \left(\mathbf{U}^{(N-2)^{\bullet 2}} \right) \odot \cdots \odot \left(\mathbf{U}^{(1)^{\bullet 2}} \right) \right) \right) \end{bmatrix}.$$

Tensorlab 4.0 – A preview

C

ABSTRACT

Since its initial release in 2013, Tensorlab has evolved into a powerful MATLAB toolbox for the analysis of tensors and the computation of tensor decompositions. This upcoming release of Tensorlab, version 4.0, widens the applicability of the toolbox to a larger range of real-world applications. New β-divergence and low-rank weighted least squares (WLS) cost-functions are introduced for the canonical polyadic decomposition (CPD), offering higher flexibility to the user. Further, updating algorithms for the CPD allow both the tracking of streaming data and the incremental computation of the CPD of a large tensor. An LS-CPD algorithm is included to compute the CPD of a tensor that is only implicitly available as the solution of an underdetermined linear system.

C.1 Tensorlab 3.0

The current 3.0 version of Tensorlab [228] features a large range of methods for the manipulation, analysis and visualization of tensors, as well as algebraic and optimization-based algorithms for the computation of tensor decompositions [174]. These range from the CPD and multilinear singular value decomposition (MLSVD) to block term decompositions (BTD) such as the decomposition in multilinear rank-$(L_r, L_r, 1)$ and more general variants. Specialized routines handle sparse, incomplete and large-scale tensors. Structured tensors (defined implicitly in Hankel, Löwner, CPD, or other form) are also supported, and their structure is strongly exploited during the computation of their decompositions. Additionally, the structured data fusion (SDF) framework provides an intuitive way to compute structured and/or coupled decompositions of tensors. The

This chapter is based on M Vandecappelle, M Boussé, N Vervliet, M Vendeville, R Zink and Lieven De Lathauwer "Tensorlab 4.0 – A preview," in 2018 14th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA 2018), July. 2018. The specific contributions of the doctoral candidate are listed in Section 11.1.

numerical philosophy behind the algorithms in Tensorlab is explained in [223]. An extensive user guide and a series of demos are available online.

C.2 New features in Tensorlab 4.0

With the upcoming release of version 4.0 of Tensorlab, its functionality will again be increased significantly. One of the main new features is the increased flexibility for computing the CPD of a tensor. The CPD of a tensor decomposes a rank-R tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ as a linear combination of R rank-1 terms: $\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ with $\otimes$ the outer product of vectors, $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_R] \in \mathbb{R}^{I \times R}$, and similarly for $\mathbf{B}$ and $\mathbf{C}$. A wider range of cost functions will be supported by allowing β-divergence and WLS cost functions. Additionally, incremental computation of the decomposition will become possible with CPD updating which will allow both the tracking of a CPD in real-time as well as the piecewise computation of the CPD of a large tensor. Furthermore, one will be able to compute the CPD of a tensor that is only available as the solution of a linear system of equations with the new LS-CPD algorithm.

C.2.1 β-divergence cost functions for the CPD

Tensorlab 3.0 only uses the Euclidean distance as metric for the cost function when approximating a tensor. This leads to good results in a lot of applications, but there are numerous situations in which the choice of a different metric can be beneficial. The Euclidean distance can be generalized to the family of β-divergences, where β is a parameter. This more general family of cost functions allows tensors to also be used for the analysis of for example counts, concentrations and spectra, for which the Euclidean distance may yield inferior results [36]. It includes, aside from the Euclidean distance ($\beta = 2$), a whole range of divergences of which the well-known Kullback–Leibler (KL) and Itakura–Saito (IS) divergences, with $\beta = 1$ and $\beta = 0$ respectively, are also part. The first- and second-order optimization algorithms for the CPD in Tensorlab were generalized so that they also support these β-divergence cost functions.

C.2.2 Low-rank WLS cost functions for the CPD

When using standard least squares cost functions to compute the CPD of a tensor, one inherently assumes that the residuals have equal variances and are uncorrelated. These assumptions may not hold in practice, however, for example

when sensors of different quality are used in array processing applications. It therefore makes sense to assign different weights to the different residuals of the decomposition and take their correlation into account. Using a full weight tensor would be too computationally demanding, but a low-rank weight tensor can already offer much more flexibility compared to standard least squares (where the weight tensor is rank-1, with vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ of which all entries are equal to 1 in the third-order case) while its CPD structure can be exploited during the computations. This leads to the following low-rank WLS cost function for the CPD of a tensor $\mathcal{T}$ in the third-order case:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \|[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] * ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T})\|_F^2,$$

where $[\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$ is a low-rank approximation of the weight tensor $\mathcal{W}$. Note that $\mathcal{W}$ and $\mathcal{T}$ can have different rank. Tensorlab 4.0 offers first- and second-order optimization algorithms to efficiently compute a CPD of a tensor in the WLS sense with a low-rank weight tensor [16].

C.2.3 CPD updating

Batch tensor methods can easily become too slow to continually recompute a CPD of a tensor when new data is added to or removed from the tensor. Also, for very large tensors, storing the full tensor to be decomposed can already be problematic. CPD updating methods solve these problems by applying efficient updates to a previously computed CPD instead of recomputing the entire decomposition when new data is added to the tensor [149, 212]. In Tensorlab, the structure of the decomposition is exploited to achieve fast updates for the CPD when new slices are added to the tensor in certain modes [212, 223]. In addition, only the previous CPD and the new tensor slices have to be stored in every updating step. This enables both the tracking of the CPD of a fast-changing tensor and the piece-wise computation of the CPD of a large tensor.

C.2.4 LS-CPD

Algebraic and first- and second-order optimization-based algorithms have been added to Tensorlab to compute the CPD of tensors that are only available as the solution of a system of equations [17]. These problems can for instance appear in tensor-based classification, or signal processing applications such as blind deconvolution of constant modulus signals. In the third-order case, one

tries to find $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, such that

$$\mathbf{M}\mathbf{x} = \mathbf{p}, \text{ with } \mathbf{x} = \text{vec}(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket),$$

where vec(.) denotes the vectorization of a tensor. The algorithms exploit the structure of $\mathbf{x}$ to directly obtain the factor matrices of the CPD, while being computationally efficient. This is especially beneficial when the system to be solved is (highly) underdetermined.

C.3 Conclusion

Tensorlab 4.0 will introduce a number of new algorithms that widen its applicability significantly. The focus of this release lies on the generalization of the CPD algorithms so that they offer a wider range of cost functions and can be used when the tensor to decompose is not static or is only given as the solution of a linear system.

Bibliography

[1] E. A. Abbott, *Flatland: A Romance of Many Dimensions*. London, UK: Seeley & Co., 1884.

[2] E. Acar, D. Dunlavy, and T. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *J. Chemometrics*, vol. 25, no. 2, pp. 67–86, 2011.

[3] E. Acar, R. Bro, and A. K. Smilde, "Data fusion in metabolomics using coupled matrix and tensor factorizations," *Proc. IEEE*, vol. 103, no. 9, pp. 1602–1620, 2015.

[4] E. Acar, E. E. Papalexakis, G. Gürdeniz, M. A. Rasmussen, A. J. Lawaetz, M. Nilsson, and R. Bro, "Structure-revealing data fusion," *BMC Bioinformatics*, vol. 15, no. 1, p. 239, 2014.

[5] S. Aja-Fernández, G. Vegas-Sánchez-Ferrero, and A. Tristán-Vega, "Noise estimation in parallel MRI: GRAPPA and SENSE," *Magnetic Resonance in Medecine*, vol. 32, no. 3, pp. 281–290, 04 2014.

[6] A. H. Andersen and W. S. Rayens, "Structure-seeking multilinear methods for the analysis of fMRI data," *NeuroImage*, vol. 22, no. 2, pp. 728–739, 06 2004.

[7] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. Papalexakis, and D. Koutra, "Com2: Fast automatic discovery of temporal ('comet') communities," in *In Advances in Knowledge Discovery and Data Mining*, V. Tseng, T. Ho, Z.-H. Zhou, A. Chen, and H.-Y. Kao, Eds. Springer, 2014, pp. 271–283.

[8] R. Badeau, B. David, and G. Richard, "Fast approximated power iteration subspace tracking," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2931–2941, 8 2005.

[9] B. W. Bader, T. G. Kolda *et al.*, "MATLAB Tensor Toolbox Version 3.1," Available online, 6 2019. [Online]. Available: http://www.tensortoolbox.org

[10] R. W. Basedow, D. C. Carmer, and M. E. Anderson, "HYDICE system: implementation and performance," in *Imaging Spectrometry*, M. R.

Descour, J. M. Mooney, D. L. Perry, and L. R. Illing, Eds., vol. 2480. International Society for Optics and Photonics, 1995, pp. 258 – 267.

[11] C. F. Beckmann and S. M. Smith, "Tensorial extensions of independent component analysis for multisubject fMRI analysis," *NeuroImage*, vol. 25, no. 1, pp. 294–311, 03 2005.

[12] A. Belouchrani, K. Abed-Meraim, J.-F. Cardoso, and E. Moulines, "A blind source separation technique using second-order statistics," *IEEE Trans. Signal Process.*, vol. 45, no. 2, pp. 434–444, 2 1997.

[13] C. Beltrán, P. Breiding, and N. Vannieuwenhoven, "Pencil-based algorithms for tensor rank decomposition are not stable," *SIAM J. Matrix Anal. Appl.*, vol. 40, no. 2, pp. 739—773, 01 2019.

[14] M. Benzi, "Preconditioning techniques for large linear systems: A survey," *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, 11 2002.

[15] N. Bertin, C. Févotte, and R. Badeau, "A tempering approach for Itakura–Saito non-negative matrix factorization. with application to music transcription," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, Taipei, Taiwan, 2009, pp. 1545–1548.

[16] M. Boussé and L. De Lathauwer, "Nonlinear least squares algorithm for canonical polyadic decomposition using low-rank weights," in *IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP 2017, Curaçao, Dutch Antilles)*, 12 2017, pp. 39–43.

[17] M. Boussé, N. Vervliet, I. Domanov, O. Debals, and L. De Lathauwer, "Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications," *Numer. Linear Algebra Appl.*, vol. 25, no. 6, p. e2190, 8 2018.

[18] E. Bratsolis and M. Sigelle, "Fast SAR image restoration, segmentation, and detection of high-reflectance regions," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 12, pp. 2890–2899, 2003.

[19] P. Breiding and N. Vannieuwenhoven, "The condition number of join decompositions," *SIAM J. Matrix Anal. Appl.*, vol. 39, 02 2018.

[20] R. Bro, "PARAFAC. Tutorial and applications," *Chemometr. Intell. Lab.*, vol. 38, no. 2, pp. 324–340, 1997.

[21] R. Bro and C. Andersson, "Improving the speed of multiway algorithms: Part II: Compression," *Chemometr. Intell. Lab.*, vol. 42, no. 12, pp. 105–113, 1998.

[22] R. Bro and H. Kiers, "A new efficient method for determining the number of components in PARAFAC models," *J. Chemometrics*, vol. 17, pp. 274–286, 2003.

[23] C. F. Caiafa and A. Cichocki, "Generalizing the column–row matrix decomposition to multi-way arrays," *Linear Algebra Applications*, vol. 433, no. 3, pp. 557–573, 2010.

[24] V. D. Calhoun and T. Adalı, "Unmixing fMRI with independent component analysis," *IEEE Eng. Med. Biol. Mag.*, vol. 25, no. 2, pp. 79–90, 03 2006.

[25] V. D. Calhoun, T. Adalı, G. D. Pearlson, P. C. M. Van Zijl, and J. Pekar, "Independent component analysis of fMRI data in the complex domain," *Magnetic Resonance in Medicine*, vol. 48, no. 1, pp. 180–192, 07 2002.

[26] D. Camps, "Pole swapping methods for the eigenvalue problem—Rational QR algorithms," Ph.D. dissertation, KU Leuven, 2019.

[27] J. D. Carrol, S. Pruzansky, and J. B. Kruskal, "CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters," *Psychometrika*, vol. 45, no. 1, pp. 3–24, 3 1980.

[28] J. L. Castellanos, S. Gómez, and V. Guerra, "The triangle method for finding the corner of the L-curve," *Applied Numerical Mathematics*, vol. 43, no. 4, pp. 359–373, 12 2002.

[29] C. Chatzichristos, E. Kofidis, Y. Kopsinis, M. Morante Moreno, and S. Theodoridis, "Higher-order block term decomposition for spatially folded fMRI data," in *13th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA 2017, Grenoble, France)*, 02 2017, pp. 3–15.

[30] C. Chatzichristos, E. Kofidis, M. Morante Moreno, and S. Theodoridis, "Blind fMRI source unmixing via higher-order tensor decompositions," *Journal of Neuroscience Methods*, vol. 315, pp. 17–47, 03 2019.

[31] C. Chatzichristos, M. Vandecappelle, E. Kofidis, S. Theodoridis, L. De Lathauwer, and S. Van Huffel, "Tensor-based blind fMRI source separation without the Gaussian noise assumption—A beta-divergence approach," in *Proceedings of the 7th IEEE Global Conference on Signal*

and Information Processing (GlobalSIP 2019, Ottawa, Canada), 11 2019, pp. 1–5.

[32] L. Cheng, X. Tong, S. Wang, Y.-C. Wu, and H. V. Poor, "Learning nonnegative factors from tensor data: Probabilistic modeling and inference algorithm," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1792–1806, 2020.

[33] Y. Cheng, F. Roemer, O. Khatib, and M. Haardt, "Tensor subspace tracking via Kronecker structured projections (TeTraKron) for time-varying multidimensional harmonic retrieval," *EURASIP Journal on Advances in Signal Processing*, vol. 2014:123, pp. 1–14, 11 2014.

[34] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations," *SIAM J. Matrix Anal. Appl.*, vol. 33, no. 4, pp. 1272–1299, 2012.

[35] A. Cichocki, D. P. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. F. Caiafa, and A.-H. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, 3 2015.

[36] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S. Amari, "Non-negative tensor factorization using alpha and beta divergences," in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007 (ICASSP 2007, Honolulu, HI)*, vol. 3, 4 2007, pp. III–1393–III–1396.

[37] A. Cichocki, R. Zdunek, A. Phan, and S. Amari, *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation*. Wiley, 2009.

[38] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 3, pp. 708–721, 2009.

[39] P. Comon and C. Jutten, *Handbook of blind source separation: Independent component analysis and applications*. Academic press, 2009.

[40] S. Coxe, S. G. West, and L. S. Aiken, "The analysis of count data: A gentle introduction to Poisson regression and its alternatives," *Journal of Personality Assessment*, vol. 91, no. 2, pp. 121–136, 2009.

[41] T. Davis, S. Rajamanickam, and W. Sid-Lakhdar, "A survey of direct methods for sparse linear systems," *Acta Numerica*, vol. 25, pp. 383–566, 2016.

[42] A. L. F. de Almeida and A. Y. Kibangou, "Distributed large-scale tensor decomposition," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014, Florence, Italy)*, 5 2014.

[43] L. De Lathauwer, "A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 642–666, 9 2006.

[44] ——, "Decompositions of a higher-order tensor in block terms — Part II: Definitions and uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1033–1066, 9 2008.

[45] ——, "A short introduction to tensor-based methods for factor-analysis and blind source separation," in *Proceedings of the 7th International Symposium on Image and Signal Processing and Analysis (ISPA 2011, Dubrovnik, Croatia)*, 9 2011, pp. 558–563.

[46] L. De Lathauwer and A. De Baynast, "Blind deconvolution of DS-CDMA signals by means of decomposition in rank-$(1, L, L)$ terms," *IEEE Trans. Signal Process.*, vol. 56, no. 4, pp. 1562–1571, 4 2008.

[47] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 4 2000.

[48] ——, "On the best rank-1 and rank-$R_1, R_2, \cdots, R_N$ approximation of higher-order tensors," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1324–1342, 7 2000.

[49] L. De Lathauwer, B. D. Moor, and J. Vandewalle, "An introduction to independent component analysis," *J. Chemometrics*, vol. 14, pp. 123–149, 5 2000.

[50] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms — Part III: Alternating least squares algorithms," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1067–1083, 9 2008.

[51] V. de Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, 9 2008.

[52] O. Debals and L. De Lathauwer, "The concept of tensorization," *Technical Report 17–99, ESAT-STADIUS, KU Leuven, Leuven, Belgium*, 2017.

[53] O. Debals, M. Van Barel, and L. De Lathauwer, "Löwner-based blind signal separation of rational functions with applications," *IEEE Trans. Signal Process.*, vol. 64, no. 8, pp. 1909–1918, 4 2016.

[54] J. P. Delmas, *Subspace Tracking for Signal Processing.* John Wiley & Sons, Ltd, 2010, ch. 4, pp. 211–270.

[55] J. W. Demmel and B. Kågström, "Computing stable eigendecompositions of matrix pencils," *Linear Algebra and its Applications*, vol. 88-89, pp. 139–186, 1987.

[56] A. den Dekker and J. Sijbers, "Implications of the Rician distribution for fMRI generalized likelihood ratio tests," *Magnetic Resonance Imaging*, vol. 23, no. 9, pp. 953 – 959, 2005.

[57] I. Domanov and L. De Lathauwer, "On the uniqueness of the canonical polyadic decomposition of third-order tensors — Part I: Basic results and uniqueness of one factor matrix," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 855–875, 7 2013.

[58] ——, "On the uniqueness of the canonical polyadic decomposition of third-order tensors — Part II: Uniqueness of the overall decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 876–903, 7 2013.

[59] ——, "Generic uniqueness conditions for the canonical polyadic decomposition and INDSCAL," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 4, pp. 1567–1589, 11 2015.

[60] ——, "Canonical polyadic decomposition of third-order tensors: Relaxed uniqueness conditions and algebraic algorithm," *Linear Algebra Appl.*, vol. 513, pp. 342–375, 1 2017.

[61] ——, "On uniqueness and computation of the decomposition of a tensor into multilinear rank-$(1, L_r, L_r)$ terms," *SIAM J. Matrix Anal. Appl.*, vol. 41, no. 2, pp. 747–803, 6 2020.

[62] ——, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 35, no. 2, pp. 636–660, 5 2014.

[63] D. L. Donoho, "Aide-Memoire. High-dimensional data analysis: The curses and blessings of dimensionality," in *AMS Conference on Math Challenges of the 21st Century (Los Angeles, USA)*, 8 2000.

[64] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[65] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 9 1936.

[66] M. Espig and W. Hackbusch, "A regularized Newton method for the efficient approximation of tensors represented in the canonical tensor format," *Numer. Math.*, vol. 122, no. 3, pp. 489–525, 5 2012.

[67] E. Evert and L. De Lathauwer, "Guarantees for the existence of best low rank tensor approximations," ESAT-STADIUS, KU Leuven, Leuven, Belgium, Tech. Rep. 19-71, 2019.

[68] ——, "Perturbation theory for joint generalized eigenvalues," ESAT-STADIUS, KU Leuven, Leuven, Belgium, Tech. Rep. 20-82, 2021.

[69] E. Evert, M. Vandecappelle, and L. De Lathauwer, "A recursive eigenspace computation for the canonical polyadic decomposition," *Technical Report 20-80, ESAT-STADIUS, KU Leuven, Leuven, Belgium*, 2020.

[70] S. Ferdowsi, V. Abolghasemi, and S. Sanei, "A comparative study of α-divergence based NMF techniques for fMRI analysis," in *Proceedings of the 19th European Signal Processing Conference(EUSIPCO 2011, Barcelona, Spain)*, 09 2011, pp. 71–75.

[71] S. Fernandes, H. Fanaee-T, and J. Gama, "Dynamic graph summarization: A tensor decomposition approach," *Data Mining and Knowledge Discovery*, vol. 32, pp. 1397–1420, 07 2018.

[72] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the β-divergence," *Neural Comput.*, vol. 23, no. 9, pp. 2421–2456, 9 2011.

[73] R. D. Fierro and P. C. Hansen, "Low-rank revealing utv decompositions," *Numerical Algorithms*, vol. 15, no. 1, pp. 37–55, 1997.

[74] D. FitzGerald, M. Cranitch, and E. Coyle, "Extended nonnegative tensor factorisation models for musical sound source separation," *Computational Intelligence and Neuroscience*, vol. 2008, pp. 50–64, 2008.

[75] S. Friedland, V. Mehrmann, R. Pajarola, and S. Suter, "On best rank one approximation of tensors," *Numerical Linear Algebra with Applications*, vol. 20, no. 6, pp. 942–955, 2013.

[76] S. Friedland and L.-H. Lim, "Nuclear norm of higher-order tensors," *Mathematics of Computation*, vol. 87, pp. 1255–1281, 11 2016.

[77] S. Friedland and G. Ottaviani, "The number of singular vector tuples and uniqueness of best rank-one approximation of tensors," *Foundations of Computational Mathematics*, vol. 14, pp. 1209–1242, 10 2014.

[78] X. Fu, C. Gao, H.-T. Wai, and K. Huang, "Block-randomized stochastic proximal gradient for constrained low-rank tensor factorization," in *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2019, Brighton, UK)*. IEEE, 2019, pp. 7485–7489.

[79] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the Itakura–Saito divergence: With application to music analysis," *Neural Computation*, vol. 21, no. 3, pp. 793–830, 2009.

[80] C. Févotte and A. Ozerov, "Notes on nonnegative tensor factorization of the spectrogram for audio source separation: statistical insights and towards self-clustering of the spatial cues," in *7th International Symposium on Computer Music Modeling and Retrieval (CMMR 2010, Málaga, Spain)*, 2010, pp. 102–115.

[81] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent, "Fast approximate natural gradient descent in a Kronecker factored eigenbasis," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 9550–9560.

[82] D. Gerard and P. Hoff, "A higher-order LQ decomposition for separable covariance models," *Linear Algebra and its Applications*, vol. 505, pp. 57–84, 2016.

[83] S. Ghanta, M. Abdel-Latif, K. Lui, H. Ravindranathan, A. John, and J. Oei, "Propofol compared with the morphine, atropine, and suxamethonium regimen as induction agents for neonatal endotracheal intubation: A randomized, controlled trial - In reply," *Pediatrics*, vol. 119, no. 6, pp. e1248–55, 06 2007.

[84] N. Gillis, *Nonnegative Matrix Factorization*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2020.

[85] N. Gillis and F. Glineur, "Using underapproximations for sparse nonnegative matrix factorization," *Pattern recognition*, vol. 43, no. 4, pp. 1676–1687, 2010.

[86] G. Golub and C. V. Loan, *Matrix Computations*. The Johns Hopkins University Press, 1996.

[87] ——, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2012.

[88] L. Grasedyck, "Hierarchical singular value decomposition of tensors," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 2029–2054, 2010.

[89] ——, "Polynomial approximation in hierarchical Tucker format by vector-tensorization," 4 2010, preprint 43, DFG/SPP1324, RWTH Aachen.

[90] H. Gudbjartsson and S. Patz, "The Rician distribution of noisy MRI data," *Magnetic Resonance in Medicine*, vol. 34, no. 6, pp. 910–914, 1995.

[91] M. Haardt, F. Roemer, and G. Del Galdo, "Higher-order SVD-based subspace estimation to improve the parameter estimation accuracy in multidimensional harmonic retrieval problems," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3198–3213, 7 2008.

[92] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation," *Journal of Fourier Analysis and Applications*, vol. 15, pp. 706–722, 10 2009.

[93] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.

[94] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations," *Optimization Methods and Software*, 4 2015.

[95] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[96] K. Hayashi, G. Ballard, Y. Jiang, and M. J. Tobia, "Shared-memory parallelization of MTTKRP for dense tensors," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Los Angeles, CA, USA, 2018, pp. 393–394.

[97] S. He, Z. Li, and S. Zhang, "Approximation algorithms for homogeneous polynomial optimization with quadratic constraints," *Mathematical Programming*, vol. 125, no. 2, pp. 353–383, 2010.

[98] N. E. Helwig and S. Hong, "A critique of tensor probabilistic independent component analysis: Implications and recommendations for multi-subject fMRI data analysis," *Journal of Neuroscience Methods*, vol. 213, no. 2, pp. 263–273, 03 2013.

[99] D. Hendrikx, L. Thewissen, A. Smits, G. Naulaers, K. Allegaert, S. Van Huffel, and A. Caicedo, "Using graph theory to assess the interaction between cerebral function, brain hemodynamics, and systemic variables in premature infants," *Complexity*, vol. 2018, pp. 1–15, 08 2018.

[100] R. Hennequin, B. David, and R. Badeau, "Beta-divergence as a subclass of Bregman divergence," *IEEE Signal Process. Lett.*, vol. 18, no. 2, pp. 83–86, 2 2011.

[101] L. T. K. Hien and N. Gillis, "Algorithms for nonnegative matrix factorization with the Kullback–Leibler divergence," *Journal of Scientific Computing*, vol. 87, no. 3, pp. 1–32, 2021.

[102] F. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *J. Math. Phys.*, vol. 6, pp. 164–189, 1927.

[103] ——, "Multiple invariants and generalized rank of a p-way matrix or tensor," *J. Math. Phys.*, vol. 7, pp. 39–79, 1927.

[104] G. Hollander, P. Dreesen, M. Ishteva, and J. Schoukens, "Approximate decoupling of multivariate polynomials using weighted tensor decomposition," *Numer. Linear Algebra Appl.*, p. e2135, 1 2018.

[105] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Rev.*, vol. 62, no. 1, pp. 133–163, 2020.

[106] F. G. C. Hoogenraad, J. R. Reichenbach, E. M. Haacke, S. Lai, K. Kuppusamy, and M. Sprenger, "In vivo measurement of changes in venous blood-oxygenation with high resolution functional MRI at 0.95 Tesla by measuring changes in susceptibility and velocity," *Magnetic Resonance in Medicine*, vol. 39, no. 1, pp. 97–107, 1998.

[107] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," *International Journal of Computer Vision*, vol. 91, no. 3, pp. 303–327, 2 2011.

[108] K. Huang and N. D. Sidiropoulos, "Kullback-Leibler principal component for tensors is not NP-hard," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 693–697.

[109] B. Hunyadi, W. Van Paesschen, M. De Vos, and S. Van Huffel, "Tensor decompositions and data fusion in epileptic EEG and fMRI data," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, pp. 1–15, 11 2016.

[110] M. Ishteva, P.-A. Absil, S. Van Huffel, and L. De Lathauwer, "On the best low multilinear rank approximation of higher-order tensors," in *Recent Advances in Optimization and its Applications in Engineering*, M. Diehl, F. Glineur, E. Jarlebring, and W. Michiels, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 145–164.

[111] L. Kangjoo, S. Tak, and J. C. Ye, "A data-driven sparse GLM for fMRI analysis using sparse dictionary learning with MDL criterion," *IEEE Trans. Med. Imag.*, vol. 30, no. 5, pp. 1076–1089, 12 2011.

[112] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Comput.*, vol. 57, pp. 222–234, 9 2016.

[113] H. Kasai, "Online low-rank tensor subspace tracking from incomplete data by cp decomposition using recursive least squares," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2016)*, Shanghai, China, 2016, pp. 2519–2523.

[114] T. Kato, *Perturbation theory for linear operators*. Springer Science & Business Media, 2013, vol. 132.

[115] H. A. L. Kiers, "Weighted least squares fitting using ordinary least squares algorithms," *Psychometrika*, vol. 62, no. 2, pp. 251–266, 6 1997.

[116] M. Kim and P. Smaragdis, "Efficient model selection for speech enhancement using a deflation method for nonnegative matrix factorization," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP 2014), Atlanta, GA, USA*. IEEE, 2014, pp. 537–541.

[117] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, 2015, pp. 9550–9560.

[118] T. G. Kolda, "Numerical optimization for symmetric tensor decomposition," *Mathematical Programming*, vol. 151, no. 1, pp. 225–248, 4 2015.

[119] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 8 2009.

[120] T. G. Kolda and D. Hong, "Stochastic gradients for large-scale tensor decomposition," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 4, 2020.

[121] D. Kressner, *Numerical Methods for General and Structured Eigenvalue Problems*. Springer-Verlag, 2005.

[122] H. Krim and M. Viberg, "Two decades of array signal processing: The parametric approach," *IEEE Signal Process. Mag.*, vol. 13, no. 4, pp. 67–94, 1996.

[123] J. B. Kruskal, "Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra Appl.*, vol. 18, no. 2, pp. 95–138, 1977.

[124] J. Landsberg, *Tensors: Geometry and applications.* American Mathematical Society, 2012.

[125] L. Lauwers and K. Barbé, "General linear models under Rician noise for fMRI data," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015, Brisbane, QLD, Australia)*, 04 2015.

[126] L. Lauwers, K. Barbé, W. Moer, and R. Pintelon, "Estimating the parameters of a Rice distribution: A Bayesian approach," in *Instrumentation and Measurement Technology Conference(I2MTC 2009)*, Singapore, 2009, pp. 114–117.

[127] S. E. Leurgans, R. T. Ross, and R. B. Abel, "A decomposition for three-way arrays," *SIAM J. Matrix Anal. Appl.*, vol. 14, no. 4, pp. 1064–1083, 1993.

[128] M. Li, Z. Li, and A. V. Vasilakos, "A survey on topology control in wireless sensor networks: Taxonomy, comparative study, and open issues," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2538–2557, 2013.

[129] L.-H. Lim and P. Comon, "Nonnegative approximations of nonnegative tensors," *J. Chemometrics*, vol. 23, no. 7–8, pp. 432–441, 2009.

[130] M. A. Lindquist *et al.*, "The statistical analysis of fMRI data," *Statistical science*, vol. 23, no. 4, pp. 439–464, 2008.

[131] Y. X. Liqun Qi, Shenglong Hu, "Spectral norm and nuclear norm of a third order tensor," *Journal of Industrial and Management Optimization*, 2021.

[132] S. Liu and G. Trenkler, "Hadamard, Khatri–Rao, Kronecker and other matrix products," *International Journal of Information and Systems Sciences*, vol. 4, no. 1, pp. 160–177, 2008.

[133] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: Properties and applications in non-Gaussian signal processing," *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5286–5298, 2007.

[134] X. Liu, S. Bourennane, and C. Fossati, "Denoising of hyperspectral images using the PARAFAC model and statistical performance analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 10, pp. 3717–3724, 2012.

[135] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognit.*, vol. 44, no. 7, pp. 1540–1551, 2011.

[136] C. Ma, X. Yang, and H. Wang, "Randomized online CP decomposition," in *2018 10th International Conference on Advanced Computational Intelligence (ICACI 2018)*, Xiamen, China, 2018, pp. 414–419.

[137] X. Ma, D. Schonfeld, and A. Khokhar, "Dynamic updating and downdating matrix SVD and tensor HOSVD for adaptive indexing and retrieval of motion trajectories," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, Taipei, Taiwan, April 2009, pp. 1129–1132.

[138] K. Madsen, N. W. Churchill, and M. Mørup, "Quantifying functional connectivity in multi-subject fMRI data using component models," *Human Brain Mapping*, vol. 38, no. 2, pp. 882–899, 10 2016.

[139] M. Mardani, G. Mateos, and G. B. Giannakis, "Imputation of streaming low-rank tensor data," *2014 IEEE 8th Sensor Array and Multichannel Signal Processing Workshop (SAM 2014, A Coruña, Spain)*, 6 2014.

[140] F. Mezzadri, "How to generate random matrices from the classical compact groups," *Notices of the American Mathematical Society*, vol. 54, no. 5, pp. 592–604, 10 2006.

[141] A. Milzarek, X. Xiao, S. Cen, Z. Wen, and M. Ulbrich, "A stochastic semismooth Newton method for nonsmooth nonconvex optimization," *SIAM Journal on Optimization*, vol. 29, no. 4, pp. 2916–2948, 2018.

[142] S. Miron, Y. Song, D. Brie, and K. Wong, "Multilinear direction finding for sensor-array with multiple scales of invariance," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 51, no. 3, pp. 2057–2070, 7 2015.

[143] M. Moonen and B. De Moor, *SVD and signal processing, III: Algorithms, architectures and applications*. Elsevier, 1995.

[144] M. Mørup and L. K. Hansen, "Automatic relevance determination for multi-way models," *Journal of Chemometrics*, vol. 23, no. 7-8, pp. 352–363, 2009.

[145] M. Mørup, L. K. Hansen, and S. M. Arnfred, "Algorithms for sparse nonnegative Tucker decompositions," *Neural Comput.*, vol. 20, no. 8, pp. 2112–2131, 8 2008.

[146] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

[147] G. Naik and W. Wang, *Blind Source Separation Advances in Theory: Algorithms and Applications*. Springer, 2014.

[148] Y. Nakatsukasa, "Algorithms and perturbation theory for matrix eigenvalue problems and the singular value decomposition," Ph.D. dissertation, University of California Davis, 2011.

[149] D. Nion and N. Sidiropoulos, "Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor," *IEEE Trans. Signal Process.*, vol. 57, no. 6, pp. 2299–2310, 6 2009.

[150] J. Nocedal and S. Wright, *Numerical optimization*. Springer New York, 7 2006.

[151] I. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 9 2011.

[152] P. Paatero, "A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis," *Chemometr. Intell. Lab.*, vol. 38, pp. 223–242, 10 1997.

[153] C. Paige and M. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, 1975.

[154] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "ParCube: Sparse parallelizable tensor decompositions," in *Machine Learning and Knowledge Discovery in Databases*, P. A. Flach, T. De Bie, and N. Cristianini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 521–536.

[155] E. E. Papalexakis, T. M. Mitchell, N. D. Sidiropoulos, C. Faloutsos, P. P. Talukdar, and B. Murphy, "Turbo-SMT parallel coupled sparse matrix-tensor factorizations and applications," *Statistical Analysis and Data Mining The ASA Data Science Journal*, vol. 9, no. 4, pp. 269–290, 6 2016.

[156] A.-H. Phan and A. Cichocki, "Tensor decompositions for feature extraction and classification of high dimensional datasets," *Nonlinear Theory and Its Applications (NOLTA IEICE)*, vol. 1, pp. 1324–1342, 2010.

[157] ——, "PARAFAC algorithms for large-scale problems," *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, 2011.

[158] A.-H. Phan, P. Tichavský, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4834–4846, 2013.

[159] A.-H. Phan, P. Tichavský, and A. Cichocki, "Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 1, pp. 126–147, 1 2013.

[160] A.-H. Phan, P. Tichavskỳ, and A. Cichocki, "Tensor deflation for CANDECOMP/PARAFAC—Part I: Alternating subspace update algorithm," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 5924–5938, 2015.

[161] Y. Qian, Y. Shen, M. Ye, and Q. Wang, "3-D nonlocal means filter with noise estimation for hyperspectral imagery denoising," in *Proceedings of the 2012 IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2012)*, Munich, Germany, 2012, pp. 1345–1348.

[162] M. Rajih, P. Comon, and R. Harshman, "Enhanced line search: A novel method to accelerate parafac," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1128–1147, 01 2008.

[163] F. Roemer and M. Haardt, "A semi-algebraic framework for approximate CP decompositions via simultaneous matrix diagonalizations (SECSI)," *Signal Processing*, vol. 93, no. 9, pp. 2722–2738, 2013.

[164] D. B. Rowe and B. R. Logan, "An fMRI activation method using complex data," in *IEEE International Symposium on Biomedical Imagaging (ISBI 2004, Arlington, VA, USA)*, 04 2004, pp. 876–879 Vol. 1.

[165] R. Roy and T. Kailath, "ESPRIT-estimation of signal parameters via rotational invariance techniques," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 7, pp. 984–995, 1989.

[166] J. P. Royer, N. Thirion-Moreau, and P. Comon, "Nonnegative 3-way tensor factorization taking in to account possible missing data," in *Proceedings of the 20th European Signal Processing Conference (EUSIPCO 2012, Bucharest, Romania)*, 8 2012, pp. 71–75.

[167] M. Rudelson and R. Vershynin, "Non-asymptotic theory of random matrices: extreme singular values," in *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*. Hyderabad, India: World Scientific, August 2010, pp. 1576–1602.

[168] E. Sanchez and B. Kowalski, "Tensorial resolution: A direct trilinear decomposition," *J. Chemometrics*, vol. 4, no. 1, pp. 29–45, 1990.

[169] N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Computation*, vol. 14, pp. 1723–1738, 2002.

[170] A. Shashua and T. Hazan, "Non-negative tensor factorization with applications to statistics and computer vision," in *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), Bonn, Germany*, 08 2005, pp. 792–799.

[171] X. Shi, H. Ling, J. Xing, and W. Hu, "Multi-target tracking by rank-1 tensor approximation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2013, Portland, OR, USA)*, 06 2013, pp. 2387–2394.

[172] N. Sidiropoulos, "Generalizing Carathéodory's uniqueness of harmonic parametrization to N dimensions," *IEEE Trans. Inf. Theory*, vol. 47, no. 4, pp. 1687–1690, 5 2001.

[173] N. Sidiropoulos, R. Bro, and G. Giannakis, "Parallel factor analysis in sensor array processing," *IEEE Trans. Signal Process.*, vol. 48, no. 8, pp. 2377–2388, 8 2000.

[174] N. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, 7 2017.

[175] N. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 57–70, 9 2014.

[176] N. D. Sidiropoulos and R. Bro, "On the uniqueness of multilinear decomposition of N-way arrays," *J. Chemometrics*, vol. 14, no. 3, pp. 229–239, 5 2000.

[177] J. Sijbers, A. den Dekker, D. Dyck, and E. Raman, "Estimation of signal and noise from Rician distributed data," in *Proceedings of the 1998 International Conference on Signal Processing and Communications*, Gran Canaria, Canary Islands, Spain, 1998, pp. 140–142.

[178] J. Sijbers, A. den Dekker, P. Scheunders, and D. Van Dyck, "Maximum-likelihood estimation of Rician distribution parameters," *IEEE Transactions on Medical Imaging*, vol. 17, no. 3, pp. 357–361, 1998.

[179] S. Smith, K. Huang, N. D. Sidiropoulos, and G. Karypis, "Streaming tensor factorization for infinite data sources," in *Proceedings of the 2018 SIAM International Conference on Data Mining (SDM'18)*. San Diego, CA, USA: SIAM, 2018, pp. 81–89.

[180] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDSP 2015)*, Hyderabad, India, 2015, pp. 61–70.

[181] A. Smits, L. Thewissen, A. Caicedo, G. Naulaers, and K. Allegaert, "Propofol dose-finding to reach optimal effect for (semi-)elective intubation in neonates," *The Journal of Pediatrics*, vol. 179, pp. 54–60.e9, 2016.

[182] A. Sobral, S. Javed, S. Ki Jung, T. Bouwmans, and E.-h. Zahzah, "Online stochastic tensor decomposition for background subtraction in multispectral video sequences," in *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshops (ICCVW 2015)*, Santiago, Chile, 2015, pp. 106–113.

[183] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 695–720, 4 2013.

[184] ——, "Structured data fusion," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, 6 2015.

[185] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization," *Computational Optimization and Applications*, vol. 63, no. 1, pp. 121–142, 05 2015.

[186] M. Sørensen and L. De Lathauwer, "Blind signal separation via tensor decomposition with Vandermonde factor: Canonical polyadic decomposition," *IEEE Trans. Signal Process.*, vol. 61, no. 22, pp. 5507–5519, 11 2013.

[187] ——, "Coupled tensor decompositions for applications in array signal processing," in *IEEE 5th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP 2013, Saint Martin, French West Indies, France)*, 12 2013, pp. 228–231.

[188] ——, "Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-$(L_{r,n}, L_{r,n}, 1)$ terms — Part I: Uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 2, pp. 496–522, 4 2015.

[189] M. Sørensen, L. De Lathauwer, P. Comon, S. Icart, and L. Deneire, "Canonical polyadic decomposition with orthogonality constraints," *SIAM J. Matrix Anal. Appl.*, vol. 33, no. 4, pp. 1190–1213, 10 2012.

[190] M. Sørensen, I. Domanov, and L. De Lathauwer, "Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-$(L_{r,n}, L_{r,n}, 1)$ terms — Part II: Algorithms," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 3, pp. 1015–1045, 4 2015.

[191] M. Sørensen, F. Van Eeghem, and L. De Lathauwer, "Blind multichannel deconvolution and convolutive extensions of canonical polyadic and block term decompositions," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4132–4145, 8 2017.

[192] M. Sørensen and L. De Lathauwer, "A multilinear singular value decomposition method for a tensor that has missing fibers," 2020, tech. report 20–53, ESAT-STADIUS, KU Leuven (Leuven, Belgium).

[193] A. Stegeman, "Comparing independent component analysis and the PARAFAC model for artificial multi-subject fMRI data," Unpublished Technical Report, Univ. of Groeningen, 02 2007.

[194] G. Steidl and T. Teuber, "Removing multiplicative noise by Douglas–Rachford splitting methods," *Journal of Mathematical Imaging and Vision*, vol. 36, no. 2, pp. 168–184, 2010.

[195] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM Journal on Numerical Analysis*, vol. 20, no. 3, pp. 626–637, 1983.

[196] G. W. Stewart, "Error and perturbation bounds for subspaces associated with certain eigenvalue problems," *SIAM Review*, vol. 15, no. 4, pp. 727–764, 1973.

[197] ——, "An updating algorithm for subspace tracking," *IEEE Trans. Signal Processing*, vol. 40, pp. 1535–1541, 1992.

[198] G. Stewart, J. Stewart, and Sun, *Matrix Perturbation Theory*, ser. Computer Science and Scientific Computing. Cambridge, Massachusetts, USA: Academic Press, 1990.

[199] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM Trans. Knowl. Discov. Data*, vol. 2, no. 11, pp. 1–37, 10 2008.

[200] Y. Sun, Y. Guo, C. Luo, J. Tropp, and M. Udell, "Low-rank Tucker approximation of a tensor from streaming data," *SIAM J. Math. Data Science*, vol. 2, no. 4, pp. 1123–1150, 2020.

[201] K. K. Talukdar and W. D. Lawing, "Estimation of the parameters of the Rice distribution," *Journal of the Acoustical Society of America*, vol. 89, no. 3, pp. 1193–1197, 1991.

[202] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*. Academic Press, 2015.

[203] L. Thewissen, A. Caicedo, A. Dereymaeker, S. Van Huffel, G. Naulaers, K. Allegaert, and A. Smits, "Cerebral autoregulation and activity after propofol for endotracheal intubation in preterm neonates," *Pediatric Research*, vol. 84, no. 5, pp. 719–725, 09 2018.

[204] P. Tichavský, A. Phan, and A. Cichocki, "Weighted Krylov–Levenberg-Marquardt method for canonical polyadic tensor decomposition," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, Barcelona, Spain, 2020, pp. 3917–3921.

[205] T. Tieleman and G. Hinton, "Lecture 6.5-RMSProp, Coursera: Neural networks for machine learning," *University of Toronto, Technical Report*, 2012.

[206] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the PARAFAC model," *Comput. Stat. Data Anal.*, vol. 50, no. 7, pp. 1700–1734, 4 2006.

[207] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[208] L. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, pp. 279–311, 1966.

[209] F. Van Eeghem, M. Sørensen, and L. De Lathauwer, "Tensor decompositions with several block-Hankel factors and application in blind system identification," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4090–4101, 8 2017.

[210] M. Vandecappelle, M. Boussé, N. Vervliet, and L. De Lathauwer, "CPD updating using low-rank weights," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018, Calgary, Alberta, Canada)*, 4 2018, pp. 6368–6372.

[211] M. Vandecappelle and L. De Lathauwer, "Low multilinear rank updating," in *Proceedings of the 53rd Asilomar Conference on Signals, Systems and Computers (Pacific Grove, California, USA)*, 11 2019, pp. 437–441.

[212] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Nonlinear least squares updating of the canonical polyadic decomposition," in *Proceedings*

of the 25th European Signal Processing Conference (EUSIPCO 2017, Kos, Greece), 9 2017, pp. 693–697.

[213] ——, "Rank-one tensor approximation with beta-divergence cost functions," in *Proceedings of the 27th European Signal Processing Conference (EUSIPCO 2019, A Coruña, Spain)*, 9 2019, pp. 1–10.

[214] ——, "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost," *IEEE Trans. Signal Process.*, vol. 68, pp. 4454–4465, 8 2020.

[215] ——, "Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost function," *IEEE J. Sel. Topics Signal Process.*, vol. 15, no. 3, pp. 491–505, 4 2021.

[216] J. Vanderhaegen, G. Naulaers, S. Van Huffel, C. Vanhole, and K. Allegaert, "Cerebral and systemic hemodynamic effects of intravenous bolus administration of propofol in neonates," *Neonatology*, vol. 98, pp. 57–63, 06 2010.

[217] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM J. Sci. Comput.*, vol. 34, no. 2, pp. A1027–A1052, 1 2012.

[218] ——, "Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking," *SIAM J. Sci. Comput.*, vol. 37, no. 3, pp. C415–C438, 2015.

[219] D. Varadarajan and J. P. Haldar, "A majorize-minimize framework for Rician and non-central chi MR images," *IEEE Trans. Med. Imag.*, vol. 34, no. 10, pp. 2191–2202, 10 2015.

[220] M. A. O. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," in *Proceedings of the European Conference on Computer Vision (ECCV 2002, Copenhagen, Denmark)*, 5 2002, pp. 447–460.

[221] R. Verschueren, N. van Duijkeren, R. Quirynen, and M. Diehl, "Exploiting convexity in direct optimal control: a sequential convex quadratic programming method," in *Proceedings of the 2016 IEEE 55th Conference on Decision and Control (CDC 2016)*, Las Vegas, NV, USA, 2016, pp. 1099–1104.

[222] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE J. Sel. Topics Signal Process.*, vol. 10, pp. 284–295, 2016.

[223] ——, "Numerical optimization based algorithms for data fusion," in *Data Fusion Methodology and Applications*, M. Cocchi, Ed. Elsevier, 2019, vol. 31, ch. 4, pp. 81–128.

[224] N. Vervliet, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization," in *Proceedings of the 50th Asilomar Conference on Signals, Systems and Computers (Pacific Grove, California, USA)*, 11 2016, pp. 1733–1738.

[225] ——, "Canonical polyadic decomposition of incomplete tensors with linearly constrained factors," *Technical Report 16—172, ESAT-STADIUS, KU Leuven, Leuven, Belgium*, 2017.

[226] ——, "Exploiting efficient representations in large-scale tensor decompositions," *SIAM J. Sci. Comput.*, vol. 41, no. 2, pp. A789–A815, 3 2019.

[227] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 71–79, 9 2014.

[228] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab 3.0," 3 2016. [Online]. Available: http://www.tensorlab.net/

[229] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers (ASILOMAR 2003)*, vol. 2, Pacific Grove, CA, USA, 2003, pp. 1398–1402.

[230] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, 2002.

[231] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[232] A. M. Wink and J. B. Roerdink, "BOLD noise assumptions in fMRI," *International Journal of Biomedical Imaging*, vol. 2006, 2006.

[233] P. Wright, "Magnetic Resonance Imaging," *IEEE Signal Process. Mag.*, vol. 14, no. 1, pp. 56–66, 01 1997.

[234] Y. Xu, "Fast algorithms for higher-order singular value decomposition from incomplete data," *J. Comput. Math.*, vol. 35, no. 4, pp. 397–422, 2017.

[235] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imaging Sci.*, vol. 6, no. 3, pp. 1758–1789, 9 2013.

[236] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, "An alternating direction algorithm for matrix completion with nonnegative factors," *Frontiers of Mathematics in China*, vol. 7, no. 2, pp. 365–384, 2012.

[237] B. Yang, "An extension of the PASTd algorithm to both rank and subspace tracking," *IEEE Signal Processing Letters*, vol. 2, no. 9, pp. 179–182, 9 1995.

[238] Y. Yang, Y. Feng, X. Huang, and J. Suykens, "Rank-1 tensor properties with applications to a class of tensor optimization problems," *SIAM J. Optim.*, vol. 26, no. 1, pp. 171–196, 2016.

[239] K. Ye and L.-H. Lim, "Schubert varieties and distances between subspaces of different dimensions," *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 3, pp. 1176–1197, 2016.

[240] A. Yeredor and M. Haardt, "Maximum likelihood estimation of a low-rank probability mass tensor from partial observations," *IEEE Signal Processing Letters*, vol. 26, no. 10, pp. 1551–1555, 2019.

[241] S. Zafeiriou, *Algorithms for Nonnegative Tensor Factorization.* London: Springer London, 2009, pp. 105–124.

[242] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian CP factorization of incomplete tensors with automatic rank determination," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1751–1763, 9 2015.

[243] M. Zhou and A. van der Veen, "Stable subspace tracking algorithm based on a Signed URV decomposition," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 3036–3051, 6 2012.

[244] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online CP decompositions for higher order tensors," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016, San Francisco, California, USA)*, 2016.

[245] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, 1997.

List of Publications

Journal papers

[1] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost," *IEEE Trans. Signal Process.*, vol. 68, pp. 4454–4465, 8 2020.

[2] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost function," *IEEE J. Sel. Topics Signal Process.*, vol. 15, pp. 491–505, 4 2021.

[3] E. Evert, M. Vandecappelle, and L. De Lathauwer, "A recursive eigenspace computation for the canonical polyadic decomposition," *Technical Report 20-80, ESAT-STADIUS, KU Leuven, Leuven, Belgium, submitted to SIMAX*, 2021.

[4] M. Vandecappelle and L. De Lathauwer, "A multilinear UTV decomposition," *Technical Report 21-122, ESAT-STADIUS, KU Leuven, Leuven, Belgium, to be submitted to IEEE Signal Processing Letters*, 2021.

[5] M. Vandecappelle and L. De Lathauwer, "Tracking the multilinear UTV decmposition," *Technical Report 21-123, ESAT-STADIUS, KU Leuven, Leuven, Belgium, to be submitted to IEEE Transactions on Signal Processing*, 2021.

[6] E. Evert, M. Vandecappelle, and L. De Lathauwer, "Computing the canonical polyadic decomposition via QZ," *Technical Report 21-124, ESAT-STADIUS, KU Leuven, Leuven, Belgium, to be submitted to IEEE Signal Processing Letters*, 2021.

Conference papers

[1] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Nonlinear least squares updating of the canonical polyadic decomposition," in *Proceedings of the 25th European Signal Processing Conference (EUSIPCO 2017, Kos, Greece)*, pp. 693–697, 9 2017.

[2] M. Vandecappelle, M. Boussé, N. Vervliet, M. Vendeville, R. Zink, and L. De Lathauwer, "Tensorlab 4.0 – a preview," in *14th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA 2018, Guildford, UK)*, pp. 1–2, 7 2018.

[3] M. Vandecappelle, M. Boussé, N. Vervliet, and L. De Lathauwer, "CPD updating using low-rank weights," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018, Calgary, Alberta, Canada)*, pp. 6368–6372, 4 2018.

[4] M. Vandecappelle and L. De Lathauwer, "Low multilinear rank updating," in *Proceedings of the 53rd Asilomar Conference on Signals, Systems and Computers (Pacific Grove, California, USA)*, pp. 437–441, 11 2019.

[5] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Rank-one tensor approximation with beta-divergence cost functions," in *Proceedings of the 27th European Signal Processing Conference (EUSIPCO 2019, A Coruña, Spain)*, pp. 1–10, 9 2019.

[6] N. Vervliet, M. Vandecappelle, M. Boussé, R. Zink, and L. De Lathauwer, "Recent numerical and conceptual advances for tensor decompositions — A preview of Tensorlab 4.0," in *Proceedings of the 2019 IEEE Data Science Workshop (DSW 2019, Minneapolis, USA)*, pp. 1733–1738, 6 2019.

[7] C. Chatzichristos, M. Vandecappelle, E. Kofidis, S. Theodoridis, L. De Lathauwer, and S. Van Huffel, "Tensor-based blind fMRI source separation without the Gaussian noise assumption—A beta-divergence approach," in *Proceedings of the 7th IEEE Global Conference on Signal and Information Processing (GlobalSIP 2019, Ottawa, Canada)*, pp. 1–5, 11 2019.

[8] A. Caicedo, O. D. Wel, M. Vandecappelle, L. Thewissen, A. Smits, K. Allegaert, L. De Lathauwer, G. Naulaers, and S. Van Huffel, "Monitoring of brain hemodynamics coupling in neonates using updated tensor decompositions," in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2019, Berlin, Germany)*, pp. 660–663, 7 2019.

Other publications

[1] M. Vandecappelle, "Tensors: big data, big future," *The Reviewer, Onderzoeksmagazine KU Leuven Campus Kulak en Campus Brugge*, p. 7, 3 2018.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
STADIUS CENTER FOR DYNAMICAL SYSTEMS, SIGNAL PROCESSING AND DATA ANALYTICS
Kasteelpark Arenberg 10, box 2446
B-3001 Leuven
michiel.vandecappelle@kuleuven.be
http://www.esat.kuleuven.be/stadius/