KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Compressed sensing approaches to large-scale tensor decompositions

Nico Vervliet

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

May 2018

Compressed sensing approaches to large-scale tensor decompositions

Nico VERVLIET

Examination committee:
Prof. dr. C. Vandecasteele, chair
Prof. dr. ir. L. De Lathauwer, supervisor
Prof. dr. ir. N. Moelans
Prof. dr. ir. M. Van Barel
Prof. dr. ir. S. Van Huffel
Prof. dr. ir. T. van Waterschoot
Prof. dr. N. Sidiropoulos
(University of Virginia, USA)
Prof. Dr. Dr. h.c. W. Hackbusch
(Max Planck Institute for Mathematics in the Sciences, Germany, and Christian-Albrechts-Universität zu Kiel, Germany)

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

May 2018

© 2018 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Nico Vervliet, Kasteelpark Arenberg 10 bus 2440, 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

Kids are sometimes asked, "What do you want to become when you grow up?" Apart from professional football player, carpenter and "this big", my answer often was "professor in everything". Obviously, this answer was inspired by professor Barabas and professor Gobelijn from the comics series 'Suske en Wiske' and 'Jommeke'. A lot started when my father gave me a book on programming in Visual Basic 6.0, which has led to many programs, e.g., for doing calculus, memorizing lists of words and a Tamagotchi. It was clear early on that programming would be important in my life. Instead of doing mathematics and science in high school, I choose to study Greek and Latin combined with mathematics, even though this was the difficult track if I wanted to become an engineer. While I was initially disappointed that there was no such thing as 'engineering in mathematics' as a bachelor program, I chose the obvious next best thing: computer science. It was not until the master program fare that I learned about mathematical engineering, which turned out to be a perfect fit. It was during this master program that I met Otto, who suggested working on a thesis supervised by some professor he knew from Kortrijk on some strange topic with 'multidimensional matrices'. The novelty of this topic allowed great flexibility and this way this professor, Lieven De Lathauwer, designed a project specifically for us. After a few months I was hooked to tensors and now, five years later, I am happy to present this book. Many persons have influenced my research during all these years, and in the remainder of this preface, I would like to say thank you to everyone who made this possible.

The input and guidance of my supervisor, Lieven, were indispensable for the success of this PhD. It is a pleasure to work with such a world authority on tensor decompositions. I really liked the freedom that he has given me to figure out ways to tackle a new problem. Although this freedom was sometimes a necessity due to the distance between Leuven and Kortrijk, he was always there on the crucial moments, even if this meant a Skype call in the middle of the night in order to meet a paper deadline. Lieven, I enjoyed our talks after every exercise session in Kortrijk. I admire your drive to do research the proper way by focusing on numerically and theoretically well-founded techniques and to explain this in a didactic way. Thank you for providing so many opportunities. In Hong Kong, for example, we even did a duo presentation in a session with many leading figures in the tensor

community. (I had to present the boring slides with the formulas, though.) Lieven, thank you for sharing your passion for tensors during these past six years.

I would like to thank my examination committee for taking the time to go through this bulky thesis text. Thank you, prof. Sabine Van Huffel, prof. Marc Van Barel and prof. Toon van Waterschoot for making all these administrative supervisory committee meetings actually useful. Thank you, prof. Nele Moelans for the years of collaboration. I learned a lot about the development of new materials and our application has been an important motivation for combining linear constraints and incomplete tensors. Thank you, prof. Wolfgang Hackbusch for taking the long train ride to Leuven twice. Thank you, Prof. Nikos Sidiropoulos for attending the defense even if it starts at 6 AM due to the time difference, as well as for all the conversations at Tricap, TDA and the winter school. I would also like to thank prof. Carlo Vandecasteele for chairing this defense.

I am grateful to the Research Foundation Flanders — FWO for awarding me an aspirant grant as it allowed me to freely pursue my own ideas and research interests. This was vital for me in order to be able to engage in a large number of collaborations and enabled me to spend time on making all results available in a software package called Tensorlab. This toolbox has been an important motivation throughout my PhD as the overwhelmingly positive feedback and the ever-increasing number of users directly shows that my work is actually relevant for other researchers and companies. Conceived by Lieven, Tensorlab's first two versions have been implemented by Laurent. I am delighted to have been able to start from such a solid foundation. I want to thank Otto for his significant contributions to Tensorlab 3.0, Marc for the helpful discussions, and both Otto and Frederik for designing the new website. Martijn, Michiel, Rob and Matthieu, thank you for your efforts to make Tensorlab 4.0 hopefully even more successful.

I have had plenty of opportunities to travel to and speak at conferences in the US, Europe, and Hong Kong. During these conferences, I have met many of the great minds in the tensor world as well as promising young researchers. I have had discussions on tensor decompositions while walking in the snow in Pecol, Italy during Tricap. In Chicago, I met Ted and Katie during a treasure hunt at the magnificent mile. Later, we met again in Utah and they invited me to an unforgettable nightly trip to natural hot springs in the mountains. (My clothes stank for weeks, though.) Sometimes it is strange to have to go to Chicago to meet someone working at the same university (Roel), which later led to the founding of the SIAM student chapter of Leuven. There are also worse places to learn about randomized linear algebra than in the beautiful city of Delphi and at the beach. It is always great to see members of our group again at other SIAM conferences.

I would like to especially thank a number of people for making the days at ESAT and in our tensor group so enjoyable. (My productive level may

not always agree, though.) First of all, thank you, Otto. It was always a pleasure to work, teach, guide students or write papers with you. After a while, simply standing at each other's desk in silence was enough to discuss and solve problems. I will always remember our $3\,\mathrm{m}^2$ room in Venice, the hard negotiations during 'Kolonisten van Catan' and intense games of squash. Tom, the hours that we have spent drinking coffee are uncountable. Otto, Tom and Steven, thanks for letting me be the 'wieltjeszuiger' on my normal bike when you went cycling on your race bikes. Martijn, the amount of red on reports of students to teach them how to write well is legendary. No one will forget how you showed us and everyone from TDA around in Brussels. Laurent, your enthusiasm when talking about new algorithms was an inspiration for continuing the work on Tensorlab. Frederik, you always had a selection of strange yet interesting questions ready. Paul, I will always remember your quote of 'something' that always runs downhill. Thank you, Rob, for all the recreational breaks with 'curve fever' and 'no brakes valet'. Griet, thank you for organizing all those great activities. Stijn, even though you joined our group and office only recently, we already had so many pleasant discussions and your knowledge of medical and other trivia seems infinite. Volleyball champion Michiel, it was a pleasure to work together on all these new methods, and thank you for being the moderating voice during thesis guidance sessions. Ignat, you always find the hardest problems for which Tensorlab fails. I want to thank all the colleagues in Kortrijk for the coffee breaks, lunches in Alma and trivia such as the fact that the word 'rugzak' is pronounced the same in Dutch and Russian. Thanks, Ignat, Mikael, Michiel, Chuan, Geunseop, Xiaofeng and Alwin! Yuning and Yunlong, thank you for introducing the Chinese specialty chicken in coca cola sauce.

Thank you, Otto, Frederik, Martijn and Stijn for finding all the forgotten words in my papers and thesis and for our discussions about the best line color to make a plot more understandable or about the positions of spaces in the word 'blinde signaalscheidingstechnieken'. I am grateful for all the efforts Anne, Elsy, John, Wim, Ida, Jacqueline and Maarten made to lighten the administrative burden and to solve many problems.

Sabine and the biomed group also deserve a special mention. When Otto and I started in 2013, we were the only (tensor) team members at ESAT. Thankfully, you 'adopted' us by inviting us to all the pizza evenings and Christmas parties. Thank you, Abhi, Adrian, Alexander B. Alexander C., Alexander S., Amalia, Amir, Bharath, Bori, Carolina, Claudio, Dorien, Dries, Griet, Jasper, Javier, John, Jonathan, Kaat, Laure, Lieven, Margot, Mario, Matthieu, Neetha, Nicolas, Ninah, Ofelie, Rob, Siba, Simon, Steven, Stijn, Thomas, Wout, Ying and Yissel. Thank you for all the lunches, parties and sports activities. Our (almost) weekly football game in the KBC hall was a highlight every week. While our email list may be long, it was always a challenge to get enough players, though. Therefore, I would like to especially thank Adrian and later Simon for taking the task upon them to gather enough

people every week and booking fields. The next goal will be to win the ESAT advanced football competition! Thank you, Nele, Inge, Bram, Yuan and Yuri for the nice collaboration and insights into materials science. I would also like to thank Mariya, Ivan, Philippe and Nick for all the discussions on tensor problems. I learned a lot from you!

My time as a project leader and coordinator at Academics for Companies was an important part of my nonacademic development especially regarding presentation skills, collaboration, and coaching. I have to thank my mentor Els for that and my fellow board members Stef, Isabelle, Nicolas, Quinten, Jelle, Diede, Klaas, Karl, Veronique, Diederik, Eline, Florian and Martijn: even though it was before my PhD, you had an enormous influence. Coaching the new board members every year and seeing what they accomplish each time has brought me a lot of joy, even if they complained that I answered every question with a new question. Thanks to my housemates in Schoonzicht for all the games of bumper pool. Bart, always remember to bring your towel and don't panic. Pieter and Nathalie, thank you for all the dinners, late night FIFA games and Tomorrowland parties.

Tot slot wil ik mijn mama, papa en zus bedanken voor alle steun die ze geboden hebben en alle kleine zaken waarvoor ik misschien niet vaak genoeg dankjewel zeg. Bedankt om begripvol te zijn op de momenten dat ik ongenietbaar was wanneer er weer iets niet werkte, en om een warme thuis te creëren. Mama, papa, bij deze wil ik deze thesis aan jullie opdragen.

Nico Vervliet
May 2018

Abstract

Today's society is characterized by an abundance of data that is generated at an unprecedented velocity. However, much of this data is immediately thrown away by compression or information extraction. In a compressed sensing (CS) setting the inherent sparsity in many datasets is exploited by avoiding the acquisition of superfluous data in the first place. We combine this technique with tensors, or multiway arrays of numerical values, which are higher-order generalizations of vectors and matrices. As the number of entries scales exponentially in the order, tensor problems are often large-scale. We show that the combination of simple, low-rank tensor decompositions with CS effectively alleviates or even breaks the so-called curse of dimensionality.

After discussing the larger data fusion optimization framework for coupled and constrained tensor decompositions, we investigate three categories of CS type algorithms to deal with large-scale problems. First, we look into sample-based algorithms that require only a subset of the tensor entries at a time and discuss (constrained) incomplete tensor techniques and randomized block sampling. Second, we exploit the inherent structure due to, e.g., sparsity or compression, by defining the structured tensor framework, which allows constraints and coupling to be incorporated trivially. Thanks to the new concept of implicit tensorization, deterministic blind source separation techniques become feasible in a large-scale setting. By formulating tensor updating in the framework, we derive new algorithms to track a time-varying decomposition or to update the decomposition when new data arrives at a possibly high rate. Finally, we present a technique to decompose tensors that are given implicitly as the solution of a linear system of equations. We present a single-step approach to compute the solution using algebraic and optimization-based algorithms and derive generic uniqueness conditions.

Numerous applications such as blind separation of convolutive mixtures, classification of hazardous gasses and modeling the melting temperature of alloys involving gigabytes or terabytes of data, are handled throughout this thesis. A final part of this thesis is dedicated to two specific applications. First, we show how tensor models enable the simulation of microstructure evolution allowing the investigation of promising multicomponent alloys using a smooth model of a sparsely sampled tensor. Second, we formulate face recognition as an implicitly given tensor decomposition to improve the accuracy.

Beknopte samenvatting

Onze hedendaagse maatschappij wordt gekenmerkt door een overvloed aan data die aan een ongeziene snelheid gegenereerd wordt. Vaak wordt het merendeel van deze data echter onmiddellijk verwijderd door compressie of informatie-extractie. Gecomprimeerd bemonsteren (GB) is een techniek waarbij deze inherente ijlheid die vaak aanwezig is, uitgebuit wordt door overbodige data niet op te meten. We combineren GB met een hogere-ordeveralgemening van vectoren en matrices, namelijk meerwegse tabellen of *tensoren*. Aangezien het aantal waarden in een tensor exponentieel toeneemt met de orde, zijn tensorproblemen vaak grootschalig. We tonen aan dat de combinatie van eenvoudige, lage-rangbenaderingen met GB effectief is in het verlichten of zelfs breken van deze zogenaamde vloek van de dimensionaliteit.

Gebruik makend van een optimalisatiegebaseerd raamwerk voor datafusie, onderzoeken we drie categorieën van GB-technieken om grootschalige tensoren te ontbinden. Eerst bestuderen we methoden die slechts een beperkt aantal elementen tegelijk beschouwen. We bespreken de ontbinding van onvolledige tensoren met eventuele lineaire beperkingen en willekeurige blok bemonsteringstechnieken. Vervolgens buiten we de inherente structuur uit die bijv. aanwezig is in ijle tensoren of na compressie, m.b.v. het nieuwe raamwerk voor gestructureerde tensoren en tonen aan dat beperkingen en koppeling triviaal ingevoerd kunnen worden. Dankzij het nieuwe concept van impliciete tensorisatie kunnen deterministische blinde signaalscheidingstechnieken toegepast worden op grootschalige problemen. We formuleren tensorupdating in dit raamwerk en leiden zo efficiënte algoritmes af om een ontbinding te updaten wanneer nieuwe data beschikbaar wordt, of te volgen wanneer deze verandert in de tijd aan een mogelijk hoog tempo. Tot slot, behandelen we een techniek om tensoren te ontbinden die impliciet gegeven zijn als de oplossing van een lineair stelsel en presenteren generische uniciteitsvoorwaarden en éénstapsmethoden gebaseerd op algebraïsche technieken of optimalisatie.

We bespreken een groot aantal toepassingen waarin gigabytes tot terabytes aan data geanalyseerd worden, bijvoorbeeld het blind scheiden van convolutieve mengels, het classificeren van gevaarlijke gassen en het modelleren van de smelttemperatuur van legeringen. In het laatste deel werken we twee toepassingen in detail uit: we tonen hoe de simulatie van de microstructuur in multicomponentenlegeringen mogelijk wordt dankzij onvolledige tensoren en veeltermbeperkingen, en hoe gezichten nauwkeuriger herkend kunnen worden dankzij de ontbinding van een impliciet gegeven tensor.

Contents

List of figures xv

List of tables xxi

List of algorithms xxiii

List of acronyms xxv

List of symbols xxix

I Introduction 1

1 Introduction 3
1.1 Big data, information and compressed sensing . . . 3
1.2 Information through factorization . . . 4
1.2.1 The cave of shadows: matrices . . . 4
1.2.2 The great beyond: tensors . . . 6
1.3 Research aims and overview . . . 9
1.3.1 Detailed overview . . . 10

2 Numerical optimization-based algorithms for data fusion 19
2.1 Introduction . . . 20
2.1.1 Outline . . . 22
2.1.2 Notation and definitions . . . 22
2.2 Numerical optimization for tensor decompositions . . . 23
2.2.1 Line search and trust region . . . 24
2.2.2 Determining step direction $\mathbf{p}_k$ . . . 25
2.2.3 Solving $\mathbf{Hp} = -\mathbf{g}$ . . . 29
2.3 Canonical polyadic decomposition . . . 30
2.3.1 Intermezzo: multilinear algebra . . . 31
2.3.2 Gauss–Newton type algorithms . . . 32
2.3.3 Alternating least squares . . . 35
2.3.4 More general objective functions . . . 37
2.4 Constrained decompositions . . . 39
2.4.1 Parametric constraints . . . 40

2.4.2 Projection-based bound constraints . . . 45
2.4.3 Regularization and soft constraints . . . 47
2.4.4 Symmetry . . . 49
2.5 Coupled decompositions . . . 50
2.5.1 Exact coupling . . . 52
2.5.2 Approximate coupling . . . 54
2.6 Large-scale computations . . . 56
2.6.1 Compression . . . 56
2.6.2 Sampling: incompleteness, randomization and updating 57
2.6.3 Exploiting structure: sparsity and implicit tensorization 59
2.6.4 Parallelization . . . 59

3 Breaking the curse of dimensionality using decompositions of incomplete tensors 61
3.1 Introduction . . . 62
3.2 Notation and preliminaries . . . 62
3.3 Tensor decompositions . . . 63
3.3.1 Canonical polyadic decomposition . . . 63
3.3.2 Tucker decomposition and low multilinear rank approximation . . . 64
3.3.3 Tensor trains . . . 65
3.3.4 Tensor networks . . . 66
3.4 Computing decompositions of large, incomplete tensors . . . 67
3.4.1 Optimization-based algorithms . . . 68
3.4.2 Pseudo-skeleton approximation for matrices . . . 69
3.4.3 Cross approximation for TT . . . 70
3.4.4 Cross approximation for LMLRA . . . 71
3.5 Case studies . . . 72
3.5.1 Multidimensional harmonic retrieval . . . 73
3.5.2 Materials science example . . . 76
3.6 Conclusion . . . 78

II Algorithms 79

4 Canonical polyadic decomposition of incomplete tensors with linearly constrained factors 81
4.1 Introduction . . . 82
4.1.1 Notation . . . 84
4.1.2 Optimization framework . . . 85
4.2 CPD with linearly constrained factors for incomplete tensors 87
4.2.1 A data-dependent algorithm: CPDLI DD . . . 87
4.2.2 Removing data dependence: CPDLI DI . . . 90
4.2.3 Complexity . . . 93

4.3 Unconstrained CPD for incomplete tensors . . . 94
4.4 Preconditioner . . . 95
4.5 Experiments . . . 96
4.5.1 CPD of incomplete tensors . . . 96
4.5.2 Preconditioner . . . 97
4.6 Materials science application . . . 99
4.7 Conclusion . . . 102

5 A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors 105
5.1 Introduction . . . 106
5.2 Stochastic optimization . . . 107
5.3 CPD by randomized block sampling . . . 109
5.3.1 Sampling operator . . . 110
5.3.2 Computing the update . . . 111
5.3.3 Step size selection . . . 112
5.3.4 Stopping criterion . . . 113
5.4 Conceptual discussion . . . 116
5.4.1 Unrestricted phase . . . 116
5.4.2 Restricted phase . . . 118
5.5 Analysis and experiments . . . 120
5.5.1 Influence of the step size adaptation . . . 121
5.5.2 Step size selection for an 8 TB tensor . . . 123
5.5.3 Influence of the block size . . . 125
5.5.4 Classifying hazardous gasses . . . 126
5.6 Conclusion . . . 128

6 Exploiting efficient representations in large-scale tensor decompositions 131
6.1 Introduction . . . 132
6.1.1 Contributions . . . 133
6.1.2 Outline . . . 134
6.1.3 Notation . . . 134
6.2 Exploiting efficient representations . . . 135
6.2.1 Overview of tensor decompositions . . . 135
6.2.2 Optimization for least squares problems . . . 137
6.2.3 Exploiting efficient representations . . . 138
6.3 Operations on efficient representations . . . 140
6.3.1 Polyadic format . . . 142
6.3.2 Tucker format . . . 142
6.3.3 Tensor train format . . . 143
6.3.4 Implicit Hankelization . . . 145
6.3.5 Implicit Löwnerization . . . 146

6.4 Experiments . . . 148
6.4.1 Accuracy and conditioning . . . 149
6.4.2 Compression for nonnegative CPD . . . 150
6.4.3 Signal separation through Hankelization . . . 152
6.5 Conclusion . . . 155

7 Nonlinear least squares updating of the canonical polyadic decomposition 157
7.1 Introduction . . . 158
7.2 Notation and definitions . . . 159
7.3 NLS updating . . . 160
7.3.1 Objective function . . . 160
7.3.2 Gradient and Gramian . . . 162
7.3.3 Windowing and dynamic rank . . . 163
7.3.4 Complexity analysis . . . 163
7.4 Experiments . . . 165
7.5 Conclusion . . . 166
7.A Updating and accuracy . . . 168

8 Linear systems with a canonical polyadic decomposition constrained solution: algorithms and applications 171
8.1 Introduction . . . 172
8.1.1 Notation and definitions . . . 175
8.1.2 Multilinear algebraic prerequisites . . . 175
8.2 Linear systems with a CPD constrained solution . . . 176
8.2.1 Definition . . . 177
8.2.2 LS-CPD as CPD by exploiting structure of **A** . . . 178
8.2.3 Generic uniqueness . . . 179
8.3 Algorithms . . . 179
8.3.1 Algebraic computation . . . 180
8.3.2 Optimization-based methods . . . 182
8.4 Numerical experiments . . . 186
8.4.1 Proof-of-concept . . . 187
8.4.2 Comparison of methods . . . 188
8.4.3 Initialization methods . . . 189
8.4.4 Preconditioner . . . 189
8.5 Applications . . . 190
8.5.1 Tensor-based face recognition using LS-CPDs . . . 191
8.5.2 Constructing a tensor that has particular multilinear singular values . . . 192
8.5.3 Blind deconvolution of constant modulus signals . . . 194
8.6 Conclusion and future research . . . 195

III Applications 197

9 Efficient use of CALPHAD based data in phase-field spinodal decomposition simulations for a quaternary system through decomposed thermodynamic tensor models 199
- 9.1 Introduction . . . 200
- 9.2 Phase-field model . . . 203
- 9.3 CALPHAD thermodynamic model . . . 204
- 9.4 Thermodynamic tensor model . . . 205
  - 9.4.1 Tensor model computation . . . 209
- 9.5 Phase-field simulation details . . . 211
  - 9.5.1 Full CALPHAD model expressions coupling . . . 213
  - 9.5.2 Interface with thermodynamic software . . . 213
  - 9.5.3 Use of the decomposed tensor model . . . 213
- 9.6 Results and discussion . . . 214
  - 9.6.1 Validation for the tensor model . . . 214
  - 9.6.2 Validation for one-dimensional simulations . . . 216
  - 9.6.3 Validation for two-dimensional simulations . . . 218
- 9.7 Conclusion . . . 219

10 Face recognition as a Kronecker product equation 223
- 10.1 Introduction . . . 224
  - 10.1.1 Notations and basic definitions . . . 225
  - 10.1.2 Multilinear singular value decomposition . . . 225
  - 10.1.3 (Coupled) Kronecker product equations . . . 225
- 10.2 Face recognition using KPEs . . . 226
  - 10.2.1 Tensorization and MLSVD model . . . 226
  - 10.2.2 Kronecker product equation . . . 227
  - 10.2.3 Face recognition . . . 227
- 10.3 Numerical experiments . . . 228
  - 10.3.1 Proof-of-concept . . . 228
  - 10.3.2 Performance . . . 229
  - 10.3.3 Improving performance through coupling . . . 229
  - 10.3.4 Updating the database with a new person . . . 230
- 10.4 Conclusion . . . 232

11 Conclusion 233
- 11.1 Comparison of methods . . . 234
- 11.2 Overview of contributions . . . 238
- 11.3 Perspectives . . . 246

A Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization **249**
A.1 Introduction . . . 250
A.1.1 History and philosophy . . . 250
A.1.2 Notation . . . 252
A.2 Structured data fusion . . . 253
A.3 Tensorization . . . 254
A.4 Coupled matrix/tensor factorization . . . 256
A.5 Large-scale tensor decompositions . . . 257
A.5.1 Randomized compression . . . 258
A.5.2 Incomplete tensors . . . 258
A.5.3 Randomized block sampling . . . 260
A.5.4 Efficient representation of structured tensors . . . 260
A.6 Conclusion . . . 261

B Supplementary materials: canonical polyadic decomposition of incomplete tensors with linearly constrained factors **263**
B.1 Derivation of CPDLI . . . 264
B.1.1 Identities . . . 264
B.1.2 Derivation of the data-dependent algorithm . . . 264
B.1.3 Derivation of the data-independent algorithm . . . 266
B.2 Preconditioner . . . 269
B.3 Experiment parameters . . . 271
B.3.1 CPD of incomplete tensors . . . 271
B.3.2 Materials science application . . . 272

List of figures

1.1 A (canonical) polyadic decomposition writes a tensor as a (minimal) sum of R rank-1 terms. 7
1.2 A low multilinear rank approximation writes a tensor as a multilinear transformation of a core tensor $\mathcal{S}$ which is multiplied in each mode by a factor matrix, which forms a basis for the respective subspace. 8
1.3 Schematic overview of the relation between chapters. 11

2.1 Clustering the points from a 3D space is impossible when only one of the left two views of the data is given. However, if the views are analyzed together, it becomes clear that both datasets can be separated by a plane as shown in the combined view. 20
2.2 As the GN algorithm converges quadratically near a local minimum, only a few iterations are required to converge to the optimum up to machine precision. ALS, which converges linearly, requires many iterations to obtain a similar precision. . 27
2.3 While the improvement in objective function value levels off after a few iterations because of the perturbations by noise (SNR is 20 dB) for both GN and ALS, the error on the factor matrices can still be improved. 28
2.4 While the direct method finds the solution of $\mathbf{Hp} = -\mathbf{g}$ in a single step, it is outperformed by the iterative methods in terms of time. 30
2.5 Grafical representation of the system $\mathbf{Hp} = -\mathbf{g}$, which is solved for $\mathbf{p}$ every iteration of the GN algorithm. 34
2.6 Every nth ALS subiteration, the system $\left(\mathbf{W}^{(n)} \otimes \mathbf{I}\right) \operatorname{vec}\left(\mathbf{P}_n\right) = -\operatorname{vec}\left(\mathbf{G}_n\right)$, with $n = 1, 2, 3$, is solved. 37
2.7 By imposing polynomial constraints, which we assume as prior knowledge, the smooth factor vectors can be recovered using a CPD of a noisy tensor. 40
2.8 To impose parametric constraints on the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, which depend on the variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively, an additional block diagonal matrix $\mathbf{J}_z$ is introduced in the system used to find $\mathbf{p}$, i.e., the update vector for the variables. 42

2.9 When adding regularization or implementing soft constraints, a block is added to the Hessian approximation and to the gradient. . . . 48
2.10 For symmetric tensors, e.g., $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{C}]\!]$, the system (2.5) can be solved more cheaply by summing blocks corresponding to the same variable. . . . 50
2.11 Did or will a user attend a certain activity at a certain location? By augmenting the GPS data tensor with information such as features for each location, relations between users and whether a user has been at a certain location, the unknown entries in the tensor can be predicted more accurately. . . . 51
2.12 In the case of coupled tensor matrix factorization in which $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and $\mathbf{M} \approx \mathbf{C}\mathbf{D}^{\mathrm{T}}$, the system (2.5) is simply the sum of the two systems [306]. . . . 53
2.13 Over a certain range of ratios ω_1/ω_2, the validation error E is reduced when jointly factorizing both measurements $\mathbf{H}_1$ and $\mathbf{H}_2$ of $h(x, y)$, compared to using only one measurement. . . . 55
2.14 While the core tensor is decomposed in the CANDELINC model, the structured tensor framework replaces the tensor with its truncated MLSVD such that the original factor matrices are kept and constraints and coupling can be imposed easily. . . . 57
2.15 Thanks to the locality principle, only few variables in the rank-1 terms affect the sampled block or subtensor. . . . 58

3.1 A *polyadic decomposition* of a third-order tensor $\mathcal{T}$ takes the form of a sum of R rank-1 tensors. . . . 64
3.2 The Tucker decomposition of a third-order tensor $\mathcal{T}$ involves a multilinear transformation of a core tensor $\mathcal{G}$ by factor matrices $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. . . . 65
3.3 A fourth-order tensor $\mathcal{T}$ can be written as a *tensor train* by linking a matrix $\mathbf{A}^{(1)}$, two tensors $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$ (the carriages) and a matrix $\mathbf{A}^{(4)}$. . . . 66
3.4 Different types of tensor networks. . . . 67
3.5 Influence of the number of known elements (left) and SNR (right) on E_{RMS}. . . . 75
3.6 Using a rank-5 model is a good trade-off between accuracy and computation time and avoids overmodeling as can be seen for the validation error. . . . 77
3.7 The values in the $R = 5$ factor vectors follow a smooth function. . . . 78
3.8 Visualization of the continuous surface of melting points when all but two concentrations are fixed. . . . 78

4.1 The Gauss–Newton type algorithms CPD and CPDI outperform the first-order NCG type algorithms as the higher per-iteration cost is countered by a significantly lower number of required iterations. 98
4.2 When missing entries follow a structured pattern, CPDI needs only few iterations to find a solution. 99
4.3 A 'dense' sampling scheme for $\mathcal{G}$ results in pyramidal structure as $c_1 + c_2 + c_3 \leq 1$. 101
4.4 Comparison of the time needed to achieve a given accuracy or training error. 102
4.5 CPDLI finds a high accuracy solution quickly even though MVR ALS is faster in achieving a low accuracy solution. 104

5.1 Illustration of the block sampling operator for a second-order tensor of size 6×6 and block size 3×2. 110
5.2 Decreasing the step size exponentially, reduces the variance quickly but levels off, while using inverse decays continue reducing the variance. 119
5.3 The CPD error E_{CPD} is decreased further when the step restriction becomes active. 121
5.4 Thanks to step restriction, the error E_{CPD} is as small as if the full tensor is used, given a large enough SNR. 122
5.5 The NLS variant consistently performs as well as or better than the ALS variant, especially for more difficult problems involving entries drawn from a uniform distribution. 123
5.6 To determine good step restriction parameters, a smaller random sample of size $80 \times 80 \times 80 \times 80$ is decomposed first. . . 125
5.7 Decomposition of the full $1000 \times 1000 \times 1000 \times 1000$ tensor with a SNR of 20 dB and sample blocks of size $40 \times 40 \times 40 \times 40$. 126
5.8 Choosing the optimal block size in terms of computation time involves a trade-off between the number of iterations and the cost for decomposing a sampled block. 127
5.9 Detail of trade-off between cost per block and number of iterations. 127
5.10 When using smaller blocks, not all tensor entries are required to recover the CPD using RBS. 128
5.11 The accuracy can be improved greatly by using step restriction. 129
5.12 Recovered factor matrices for a rank $R = 5$ CPD of the chemical analytes dataset. 130

6.1 By reducing the angle α between the factor vectors, the condition of the CPD worsens, resulting in a higher error E_{CPD}. 150

6.2 When using a compressed tensor instead of the full tensor, the computational cost scales linearly in the tensor dimensions instead of cubically for a rank-10 nonnegative tensor of size $I \times I \times I$. 151

6.3 Using a TT approximation instead of the full $10 \times 10 \times \cdots \times 10$ tensor removes the exponential dependence on the order N when computing a rank-5 nonnegative CPD. 152

6.4 For low and medium SNR, the errors on the mixing matrix for the explicit and implicit Hankelization approaches are equal. While the error for the implicit tensorization stagnates for SNR larger than 180 dB, the error continues to decrease when using the explicit tensorization. The errors are medians over 100 experiments, each using a best-out-of-five initializations strategy. 154

6.5 For both the unconstrained LL1 decomposition and the constrained BTD, the computation time using implicit tensorization increases more slowly, enabling large-scale applications. . 155

6.6 The relative error E of the estimated mixing matrix decreases when the number of samples M increases. 156

7.1 In the updating procedure, the decomposition of the old tensor is updated and a new row is added, both based on the new slice. 161

7.2 The proposed updating methods perform almost as well as the batch methods. 167

7.3 In the noiseless case, the error increases due to numerical error accumulation, while the error decreases in the noisy case thanks to statistical averaging. 169

8.1 Tensor decompositions are a higher-order generalization of matrix decompositions and are well-known tools in many applications within various domains. Although multilinear systems are a generalization of linear systems in a similar way, this domain is relatively unexplored. LS-CPDs can be interpreted as multilinear systems of equations, providing a broad framework for the analysis of these types of problems. 174

8.2 Our algebraic method and optimization-based method (with random initialization) can perfectly reconstruct an exponential solution vector in the noiseless case. 187

8.3 Our optimization-based method (with random initialization) can reconstruct the rational solution vector in the noiseless case. 188

8.4 The naive method fails for an underdetermined LS-CPD while the NLS and algebraic method both perform well. The computational complexity of the algebraic method is much higher than the other two methods, especially for the highly underdetermined case (i.e., M close to the number of free variables). 189
8.5 By initializing the NLS algorithm with the algebraic solution instead of using a random initialization, fewer iterations are needed to achieve convergence. 190
8.6 Correct classification of an image of a person under a *new* illumination condition. 192
8.7 The LS-CPD approach obtains more accurate results than the naive method and achieves similar accuracy as the dedicated OSACM method. The run-time of LS-CPD is slightly higher than OSACM for this example. 196

9.1 Representation of the Gibbs free energy tensor with different resolutions used in the decomposition model. 206
9.2 Illustration of a canonical polyadic decomposition (CPD). . . 207
9.3 The different schemes to use a CALPHAD model in a phase-field model are compared in this chapter. 212
9.4 The exponential dependence of the tensor on its order is broken by using a TTM, for which the number of coefficients grows only linearly with N. In the plot, it can be seen that the number of entries in the tensor increases exponentially in the order. In contrast, for the TTMs with ranks $R = 3, 6, 9$, for example, the number of coefficients necessary to represent the data with good accuracy depends only linearly on the order of the tensor. 215
9.5 The largest improvements in accuracy are seen for TTMs with $R = 5, 6, 7$; For higher ranks, i.e., $R > 7$, no noticeable improvements are observed. 216
9.6 No improvements to the accuracy of the simulations are observed for $R > 6$. 217
9.7 Analysis of the 2D simulation confirm the results obtained from 1D, with no improvements to the accuracy being observed by using TTMs with $R > 6$. 220
9.8 The measurements of volume fraction shown that depending on the application a rank $R = 5$ or even $R = 4$ can lead to accurate results. 221

10.1 Classification of a person that is included in the dataset. . . . 229
10.2 The MLSVD model captures the new person reasonably well. 231

10.3 Although we update the database with a new person using only one illumination condition, the KPE-based method recognizes that person in a new image under a different illumination condition. 231

A.1 Block term decomposition of a tensor $\mathcal{T}$ in terms with multilinear ranks (L_r, M_r, N_r). 253
A.2 A schematic of structured data fusion. 255
A.3 For a joint decomposition of $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{B}]\!]$ and $\mathbf{M} = \mathbf{B}\mathbf{B}^{\mathrm{T}}$, the CCPD algorithm directly computes the contracted Gramian and gradient, while `sdf_nls` computes all blocks separately. . 258

List of tables

2.1 The (element-wise) objective function f_i and its derivatives for some divergences. 37

3.1 Number of parameters and touched elements for the three decompositions of incomplete tensors. 74

4.1 The per-iteration complexity of the CPDLI DD algorithm is dominated by Gramian operations. 93
4.2 The per-iteration complexity of the CPDLI DI algorithm is dominated by Gramian operations involving $\mathbf{D}$. 94
4.3 The proposed PC and the incompleteness agnostic PC from [262] reduce the number of CG iterations significantly compared to the nonpreconditioned case. 99
4.4 The results computed by CPDLI algorithms are almost an order of magnitude more accurate than those computed by MVR ALS and SDF in terms of median error and maximal error. 103
4.5 CPDLI is more than an order of magnitude more accurate than MVR ALS in terms of relative error on the validation data, when 200 samples are used for training (experiment 2). 103

5.1 When ALS converges, it is usually fast, but needs a lot of samples; NLS always converges and often does not access the full tensor. 124
5.2 Performance on the chemical analytes dataset without and with step restriction. 129

6.1 Computational per-iteration complexity when computing a rank-R CPD of an Nth-order $I \times \cdots \times I$ tensor given in its efficient representation. 141

7.1 Median of the CPU time (in ms) for a single update using the new updating method. 166
7.2 Weighted mean errors for the new updating method. 167

8.1 The per-iteration computational complexity of the NLS algorithm for LS-CPD is dominated by the computation of the Jacobian. 185
8.2 Both PCs reduce the number of CG iterations in the underdetermined and square case. In the highly underdetermined case only the block-Jacobi PC can reduce the number of CG iterations. 190
8.3 The LS-CPD method for constructing a tensor with particular multilinear singular values is faster than APM. 194

9.1 Alloys compositions selected for spinodal decomposition simulations. 211
9.2 Parameters used in (9.6) for conducting the phase-field simulations. 213
9.3 Certain features present in the 2D microstructures resulting from simulation with the full CALPHAD expression can also be seen in microstructures from simulations with TTMs $R \geq 6$. 219

10.1 By reformulating face recognition as a Kronecker product equation, higher performance (%) can be obtained in comparison to conventional techniques such as Eigenfaces as well as the tensor-based approach in [296]. 229
10.2 Higher performance (%) can be achieved by using multiple images under different illuminations. 230
10.3 When updating the database with a new person, our method can achieve higher accuracy (%) by fusing multiple images under different illumination conditions instead of using only one image of the new person. 231

A.1 Compared to SDF, CCPD requires less time and fewer iterations to converge when computing (A.3). 257

List of algorithms

3.1 Cross approximation for matrices [214]. 70
3.2 Tensor train decomposition using SVD [217]. 70
3.3 Fiber sampling tensor decomposition algorithm [51]. 72

4.1 CPDLI DD using Gauss–Newton with dogleg trust region. . . . 87
4.2 CPDLI DI using Gauss–Newton with dogleg trust region. 90

5.1 Randomized block sampling CPD. 110

7.1 NLS updating for CPD. 164

8.1 Algebraic algorithm to solve $\mathbf{Ax} = \mathbf{b}$ in which $\mathbf{x}$ has a rank-1 CPD structure. 182
8.2 LS-CPD using Gauss–Newton with dogleg trust region. 184

A.1 Computation of MLSVD using randomization and subspace iteration. 259

List of acronyms

ACA	adaptive cross approximation
ACMA	analytical constant modulus algorithm
ACMTF	advanced coupled matrix tensor factorization
ADMoM	alternating direction method of multipliers
ALS	alternating least squares
APM	alternating projection method
AR	autoregressive
BCD	block coordinate descent
BFGS	Broyden–Fletcher–Goldfarb–Shanno
BPSK	binary phase shift keying
BTD	block term decomposition
CA	cross approximation
CALPHAD	computer coupling of phase diagrams and thermochemistry
CCPD	coupled canonical polyadic decomposition
CE	CALPHAD expression
CERN	Conseil Européen pour la Recherche Nucléaire
CG	conjugate gradients
cKPE	coupled Kronecker product equation
CM	constant modulus
CMA	constant modulus algorithm
CMTF	coupled matrix tensor factorization
COT	complex optimization toolbox
CPD	canonical polyadic decomposition
CPDI	CPD for incomplete tensors
CPDLI	CPD with linear constraints for incomplete tensors
CPF	cumulative probability function
CPU	central processing unit
CRB	Cramér–Rao bound
CS	compressed sensing
CSF	compressed sparse fibers
DD	data dependent

DI	data independent
DLTR	dogleg trust region
EEG	electroencephalography
EM	expectation maximization
FFT	fast Fourier transform
fMRI	functional magnetic resonance imaging
GEVD	generalized eigenvalue decomposition
GN	Gauss–Newton
GPS	global positioning system
HOSVD	higher order singular value decomposition
HT	hierarchical Tucker
KL	Kullback–Leibler
KPE	Kronecker product equation
LBFGS	limited memory BFGS
LHC	Large Hadron Collider
LL1	decomposition in multilinear rank-$(L_r, L_r, 1)$ terms
LM	Levenberg–Marquardt
LMLRA	low multilinear rank approximation
LMS	least mean squares
LS	least squares
LS-CPD	linear system with CPD constrained solution
MHR	multidimensional harmonic retrieval
MLSVD	multilinear singular value decomposition
MODIS	Moderate Resolution Imaging Spectroradiometer
MPS	matrix product states
MRSI	magnetic resonance spectroscopy imaging
MTKRONPROD	matricized tensor Kronecker product
MTKRPROD	matricized tensor Khatri–Rao product
MVR	multivariate regression
NASA	National Aeronautics and Space Administration
NCG	nonlinear conjugate gradients
NLS	nonlinear least squares
NMF	nonnegative matrix factorization
OSCMA	optimal step-size constant modulus algorithm
PC	preconditioner
PCA	principal component analysis

PCG	preconditioned conjugate gradients
PD	polyadic decomposition
PDE	partial differential equation
PEPS	projected entangled-pair states
QAM	quadrature amplitude modulation
qN	quasi-Newton
RAM	random access memory
RBS	randomized block sampling
RMSE	root mean square error
SDF	structured data fusion
SGD	stochastic gradient descent
SGTE	scientific group thermodata Europe
SISO	single input single output
SNR	signal-to-noise ratio
SVD	singular value decomposition
TD	Tucker decomposition
TPU	tensor processing unit
TT	tensor trains
TTM	thermodynamic tensor model

List of symbols

Unary

$a, b, \ldots$	scalar
$\mathbf{a}, \mathbf{b}, \ldots$	vector
$\mathbf{A}, \mathbf{B}, \ldots$	matrix
$\mathcal{A}, \mathcal{B}, \ldots$	tensor
$\mathcal{I}, \mathcal{J}, \mathcal{K}$	index set
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with mean μ and variance σ^2
$\mathcal{U}(a, b)$	uniform distribution in interval $[a, b]$
$.^{(n)}$	nth entry in (ordered) set
$\mathbf{T}_{(n)}$	mode-n unfolding of tensor $\mathcal{T}$
$\mathbf{T}_{(m,n)}$	mode-(m, n) unfolding of tensor $\mathcal{T}$
$\mathbb{R}$	real field
$\mathbb{C}$	complex field
$\mathbb{K}$	real or complex field
$\operatorname{Re}(\cdot)$	real part
$\operatorname{Im}(\cdot)$	imaginary part
$\lvert\cdot\rvert$	absolute value or modulus
$\lvert\lvert\cdot\rvert\rvert$ or $\lvert\lvert\cdot\rvert\rvert_{\mathrm{F}}$	Frobenius norm
$.^{\mathrm{T}}$	transpose
$.^{\mathrm{H}}$	Hermitian transpose
$.^{-1}$	inverse
$.^{\dagger}$	Moore–Penrose pseudoinverse
$\bar{.}$	conjugation
$\mathcal{O}(\cdot)$	big-O notation
$\operatorname{vec}(\cdot)$	column-wise vectorization
$\operatorname{unvec}(\cdot)$	reshape vector into tensor
$\mathbf{I}$ or $\mathbf{I}_n$	identity matrix (of size $n \times n$)
$\mathbf{0}_n$	vector of zeros with length n
$\mathbf{0}_{m \times n}$	matrix of zeros with dimensions $m \times n$
$\mathbf{1}_n$	vector of ones with length n
$\mathbf{1}_{m \times n}$	matrix of ones with dimensions $m \times n$
$\mathbb{E}[\cdot]$	expectation
$\mathfrak{F}$	fast Fourier transform
$\mathfrak{F}^{-1}$	inverse fast Fourier transform

$\underline{\mathfrak{F}}^{-1}$	last I rows of inverse fast Fourier transform
$\lfloor x \rfloor$	floor (nearest integer smaller than or equal to)
$\lceil x \rceil$	ceil (nearest integer larger than or equal to)
$\mathcal{C}_2$	second compound matrix
$\equiv$	equivalent
d	derivative
∂	partial derivative
∇	gradient
∇^2	Hessian

Binary

$\odot$	column-wise Khatri–Rao product
$\odot^{\mathrm{T}}$	row-wise Khatri–Rao product
$\otimes$	Kronecker product
$\otimes$	outer product
$\langle \cdot, \cdot \rangle$	inner product
$*$	Hadamard or element-wise product
$\cdot_n$	mode-n tensor matrix product

Other

$[\![\mathbf{A}, \mathbf{B}, \ldots]\!]$	polyadic decomposition with factors $\mathbf{A}, \mathbf{B}, \ldots$
$[\![\mathcal{G}; \mathbf{A}, \mathbf{B}, \ldots]\!]$	low multilinear rank approximation with core tensor $\mathcal{G}$ and factors $\mathbf{A}, \mathbf{B}, \ldots$
$\mathrm{diag}(\mathbf{A})$	diagonal of matrix
$\mathrm{true}(\mathbf{A})$	Upper diagonal part of matrix
$\mathrm{diag}(\mathbf{a})$	diagonal matrix with entries in $\mathbf{a}$
$\mathrm{blkdiag}(\mathbf{A}, \mathbf{B}, \ldots)$	block diagonal matrix with blocks $\mathbf{A}, \mathbf{B}, \ldots$
$[\mathbf{a}, \mathbf{b}, \ldots]$	horizontal concatenation
$[\mathbf{a}; \mathbf{b}; \ldots]$	vertical concatenation
E_{CPD}	maximal relative error on factor matrices
C_n^k	binomial coefficient

Part I

Introduction

1 Introduction

1.1 Big data, information and compressed sensing

Gargantuan amounts of data are generated at increasingly fast rates in a quest to understand the universe, to make business decisions, to solve engineering problems, to provide entertainment, to improve quality of life and so on. For example, NASA's MODIS satellites monitor every place on earth every one to two days, generating about 65 GB of data each day [204]. To validate the existence of the Higgs boson in 2012, the servers at the Large Hadron Collider (LHC) at CERN sift through 600 million events per second, each event generating 1 MB of data, to find few significant ones [58], [59]. In 2012, the 1000 genomes project has already collected whole-genome sequences for thousands of individuals which corresponds to 260 TB of data [68]. In scientific computing, high-dimensional integration can easily lead to more numerical values than the number of atoms in the observable universe [217], which is estimated at $\mathcal{O}\left(10^{82}\right)$. Intel estimates that an autonomous car will generate up to 4 TB each hour [180], [206].

All this data, which can be represented by or stored in graphs, images, tables, relational databases, unstructured text files etc., is still a raw product: the actual goal is to extract useful information. Such information can be a suggestion for a movie to watch next, as is done in recommender systems based on previous choices and the behavior of millions of other users. Weather simulations integrate data from satellites, terrain maps and weather balloons in order to predict whether it will rain tomorrow. In hospitals, MRI scans are used to accurately outline the region affected by a tumor. A farmer can see if crops are ready to be harvested from the analysis of hyperspectral images. Time series from sensors distributed across a factory can be used to locate faulty machines by monitoring vibration patterns.

While data is already available in abundance and even more is generated every second, one of the first steps is often to throw away large parts of this data, which can be done with little or no loss of information. For example, from the 600 million events generated per second by the LHC, only 100

to 200 events are kept for further inspection [58]. An image taken with a 12 MP smartphone camera would require about 36 MB of data, but thanks to JPEG compression about one tenth is required. Similarly, various codecs such as mp3, aac or H.264 are used to compress audio and video. To store its massive amounts of data, Facebook developed the faster, lossless Zstandard compression tool [109].

If compression is performed anyway, the question arises why all this data is generated, measured or computed in the first place. In compressed sensing (CS), one tries to reconstruct signals without measuring the part of the data that is thrown away. The key underlying assumption is that the information content of many signals is low, i.e., that the data is *sparse* and can be represented by few coefficients given a certain basis such as a Fourier or wavelet basis. The single-pixel camera [102], for example, uses thousands of measurements from a single pixel instead of a single measurement from millions of pixels to take a picture. The low information content also allows multirate sampling techniques to reconstruct signals with fewer samples than dictated by the Shannon–Nyquist bound[1]. From an algorithm design point-of-view, the use of compressed representations of the data can be very beneficial as the latency of memory access or communication with hard disks or other computers across a network is still high: while processing power has adhered Moore's law up until very recently, memory access time and communication cost stayed behind and actually become the bottleneck.

1.2 Information through factorization

A plethora of techniques is available to extract information from data, depending on the type of data, the application domain and the assumptions made. In this thesis, we focus on data that can be represented as matrices and tensors and use factorizations, or *decompositions*, to extract information.

1.2.1 The cave of shadows: matrices

ὁμοίους ἡμῖν, ἦν δ' ἐγώ: τοὺς γὰρ τοιούτους πρῶτον μὲν ἑαυτῶν τε καὶ ἀλλήλων οἴει ἄν τι ἑωρακέναι ἄλλο πλὴν τὰς σκιὰς τὰς ὑπὸ τοῦ πυρὸς εἰς τὸ καταντικρὺ αὐτῶν τοῦ σπηλαίου προσπιπτούσας;

"They're like us," I said. "For in the first place, do you suppose such men would have seen anything of themselves and one another other than the shadows cast by the fire on the side of the cave facing them?"

— Plato, Republic, 381 B.C. [30], [227]

[1] This bound states that the sampling rate should be at least twice the signal bandwidth, which may result in excessive amounts of data.

Decompositions of matrices are key tools to analyze data, to discover underlying sources, to remove noise or for compression. Consider the important example of blind source separation (BSS) in which one attempts to uncover the underlying (unknown) sources $\mathbf{S}$ given a (noisy) linear mixture $\mathbf{X}$ of these sources:

$$\mathbf{X} = \mathbf{M} \cdot \mathbf{S}.$$

By factorizing $\mathbf{X}$ the mixing matrix $\mathbf{M}$ and sources $\mathbf{S}$ can be recovered. This decomposition is not unique, however, as any invertible matrix $\mathbf{W}$ can plugged in without losing the equality:

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

Hence, without additional constraints on the factors $\mathbf{M}$ and $\mathbf{S}$, the original sources cannot be recovered.

More generally, a matrix decomposition writes a matrix $\mathbf{X}$ as a product of other matrices with a certain structure, e.g.,

$$\mathbf{X} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^{\mathrm{T}} \quad \text{or} \quad \mathbf{X} = \mathbf{A} \cdot \mathbf{B}.$$

In the case of the singular value decomposition (SVD), $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{S}$ is diagonal. In the case of the QR decomposition, $\mathbf{A}$ has orthonormal columns and $\mathbf{B}$ is upper triangular. The Cholesky factorization writes a symmetric positive definite matrix as $\mathbf{X} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ in which $\mathbf{L}$ is lower triangular. In nonnegative matrix factorization, $\mathbf{A}$ and $\mathbf{B}$ have nonnegative entries. While a generic matrix $\mathbf{X}$ has full rank and the matrices $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}$, $\mathbf{A}$ and $\mathbf{B}$ are either square or have the same dimensions as $\mathbf{X}$, one often uses a low-rank approximation. Let $\mathbf{X}$ be an $I \times J$ matrix, then the SVD can be seen as a sum of R rank-1 terms

$$\mathbf{X} = \sum_{r=1}^{R} \sigma_r \mathbf{u}_r \mathbf{v}_r^{\mathrm{T}},$$

in which σ_r is the rth entry on the (ordered) diagonal of $\mathbf{S}$ and R is the rank of $\mathbf{X}$. In many applications, the signals of interest are the ones contributing to the R largest singular values σ_r, while the remaining singular values are considered as 'noise' contributions and are set to zero. Hence, $\mathbf{X}$ can be written as the product of an $I \times R$ matrix, an $R \times R$ diagonal matrix and an $R \times J$ matrix. This means that the IJ entries in $\mathbf{X}$ can be represented compactly with $\mathcal{O}\left(R(I+J)\right)$ parameters if $R \ll I, J$. The Eckhart–Young–Mirsky theorm states that this truncated SVD actually gives the best approximation in a least-squares sense [103].

Presented with many decompositions and choices for the rank, the *lex parsimonia*, also known as Occam's razor, dictates that the simplest one, i.e., the

one making the fewest assumptions, should be chosen. For example, in data analysis or signal processing, the goal is to find the underlying phenomena or sources while disregarding irrelevant 'noise' sources. To achieve this, models are often trained on a training set, and then validated on an unused part of the data. Another approach is to use the Bayesian or Akaike information criterion, which penalizes the model complexity. In the case of matrix decompositions, model simplicity can be translated to low-rank assumptions or can be achieved by constraining factors to impose additional structure.

1.2.2 The great beyond: tensors

> συνηθείας δὴ οἶμαι δέοιτ᾽ ἄν, εἰ μέλλοι τὰ ἄνω ὄψεσθαι. καὶ πρῶτον μὲν τὰς σκιὰς ἂν ῥᾷστα καθορῷ, καὶ μετὰ τοῦτο ἐν τοῖς ὕδασι τά τε τῶν ἀνθρώπων καὶ τὰ τῶν ἄλλων εἴδωλα, ὕστερον δὲ αὐτά: ἐκ δὲ τούτων τὰ ἐν τῷ οὐρανῷ καὶ αὐτὸν τὸν οὐρανὸν νύκτωρ ἂν ῥᾷον θεάσαιτο, προσβλέπων τὸ τῶν ἄστρων τε καὶ σελήνης φῶς, ἢ μεθ᾽ ἡμέραν τὸν ἥλιόν τε καὶ τὸ τοῦ ἡλίου.
>
> "Then I suppose he'd have to get accustomed, if he were going to see what's up above. At first he'd most easily make out the shadows; and after that the phantoms of the human beings and the other things in water; and later, the things themselves. And from there he could turn to beholding the things in heaven and heaven itself, more easily at night – looking at the light of the stars and the moon — than by day — looking at the sun and sunlight."
>
> — Plato, Republic, 381 B.C. [30], [227]

Similar to the escaped prisoner who looks beyond the shadows, we focus in this thesis on tensors, which are a higher-order generalization of vectors (first order) and matrices (second order). While tensors in their broadest definition are elements in a tensor product of vector spaces, we represent tensors as multiway arrays of numerical values. Each mode of a tensor describes an aspect or variable in the data. For example, for movie recommendation a tensor with modes person × movie × time can be used and each entry describes the rating a certain person gives for a movie at a certain point in time. In chemometrics, spectroscopy data for multiple samples is combined, resulting in a tensor with modes excitation spectrum × emission spectrum × sample. A tensor representation of a database with images of faces under a variety of illumination conditions has modes pixels × person × illumination. More examples can be found in numerous overview papers and books; see, e.g., [65], [126], [129], [170], [243], [250].

As for matrices, decompositions are important tools to extract information or to compress data. It is possible to flatten or unfold the tensor into a matrix and use matrix decompositions, but the resulting models may fail

to grasp possibly vital multilinear structure. Instead, we focus on a number of tensor decompositions that are most relevant for data analysis and signal processing, and exploit this structure, such as the canonical polyadic decomposition (CPD), the low multilinear rank approximation (LMLRA), the multilinear singular value decomposition (MLSVD) and the block term decomposition (BTD). In other fields such as, e.g., scientific computing and quantum information theory, decompositions such as tensor trains (TT), hierarchical Tucker (HT) and tensor networks (TN) are often used; see [126], [129], [211].

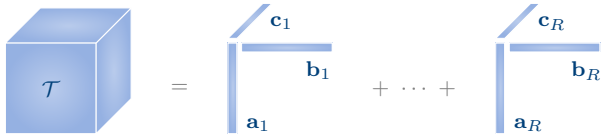


Figure 1.1: A (canonical) polyadic decomposition writes a tensor as a (minimal) sum of R rank-1 terms.

The polyadic decomposition (PD) writes an Nth-order tensor $\mathcal{T}$ as a sum of rank-1 terms, each of which is an outer product, denoted by $\otimes$, of N nonzero vectors, as depicted in Figure 1.1. For example, for a third-order tensor, we can write

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

If R is the minimal number of rank-1 terms required to make the equality hold, R is the rank of $\mathcal{T}$ and the PD is called canonical, hence the name CPD[2]. As only NR vectors need to be stored, the number of parameters is $\mathcal{O}(NIR)$ for an Nth-order cubical tensor with dimensions $I \times I \times \cdots \times I$, which is often far smaller than the number of entries, which scales exponentially in N, i.e., as I^N. Hence, if a low-rank tensor approximation can be used, the exponential dependency on N becomes linear and the so-called curse of dimensionality is broken. This curse, its consequences and remedies are discussed in detail in Chapter 3. An important property of the CPD is that it is unique[3] under mild conditions in contrast to matrix decompositions; see [243] and references therein. This uniqueness is key in many applications in signal processing, chemometrics, data analysis, machine learning, psychometrics and so on. To make the model more interpretable, constraints such as nonnegativity,

[2]Other names such as PARAFAC, CANDECOMP, separation rank decomposition, tensor rank decomposition, r-term decomposion, Hitchcock's decomposition and Kruskal tensor can be found in literature as well.

[3]When using the term unique, we actually mean essentially unique, i.e., unique up to trivial scaling and permutation indeterminacies.

orthogonality or Vandermonde structure can be imposed. A framework for constrained decompositions is discussed in Chapter 2.

The multilinear rank of a tensor is defined as the tuple $(R_1, R_2, \ldots, R_N)$ collecting the ranks of the mode-n unfoldings of a tensor for $n = 1, \ldots, N$, i.e., the matrices containing the columns (mode-1 vectors), rows (mode-2 vectors) and more generally the mode-n vectors as their columns. This rank definition is related to the multilinear singular value decomposition (MLSVD) [78], which computes N orthonormal bases for the subspaces spanned by the mode-n vectors and an ordered, all-orthogonal core tensor $\mathcal{S}$; see Figure 1.2. Again, a low multilinear rank approximation (LMLRA)[4] is often used to find the subspaces of interest, for denoising or to compress the data [79]. In contrast to the matrix case, simply truncating the decomposition by setting the multilinear singular values to zero does not result in the optimal LMLRA although the error can be bounded [78] and is often small in signal processing applications.

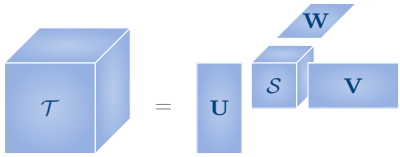


Figure 1.2: A low multilinear rank approximation writes a tensor as a multilinear transformation of a core tensor $\mathcal{S}$ which is multiplied in each mode by a factor matrix, which forms a basis for the respective subspace.

In this thesis we build towards a set of algorithms to compute these tensor decompositions efficiently, even for large-scale data. While matrix algorithms have a long history which has led to highly optimized software packages such as the basic linear algebra subprograms (BLAS) and the linear algebra package (LAPACK), there is still much progress to be made in the development of highly efficient tensor algorithms. On the low-level end, much effort is put into computing efficient tensor contractions; see, e.g., tensor contraction engine [21], Facebook tensor comprehensions [108], tensor contraction code generator [273], libtensor [146], TBLIS [198], cyclops tensor framework [255] and TiledArray [53]. Computing these contractions and handling tensor data efficiently is considered so important that dedicated processors, such as the tensor processing unit (TPU) [155] are being developed. In the case of higher-level algorithms, decompositions such as the MLSVD, tensor trains or the hierarchical Tucker decomposition are often computed using techniques from numerical linear algebra; see, e.g., [79], [306], Chapter 3 and Appendix A. The CPD and the more general BTD are usually computed using optimization-based algorithms [82], [243], [260]. In Chapter 2, we give an overview of

[4]The name Tucker decomposition can be found in literature as well.

these algorithms and how they can be generalized to be used in a data fusion framework.

1.3 Research aims and overview

When dealing with large-scale tensors, one approach to lower the computation time and lower the burden on memory is to distribute the data and compute the updates in the optimization algorithm in parallel. While this lowers the computational cost, the asymptotic complexity[5] remains of the same order as only the constants are made smaller. In this thesis, we aim to fundamentally lower the complexity by combining tensor techniques with a compressed sensing type approach. We show that this can be done by making two main assumptions: the sparsity requirement is translated to the assumption that the tensor has a low rank or a low multilinear rank, and as a compressed measurement we take a subset, or sample, of the entries, an efficient representation using few parameters, or a more general compressed measurement such that the tensor is defined implicitly. The goal is to derive various techniques and algorithms to achieve this complexity reduction such that a laptop or desktop computer is sufficient to decompose large-scale tensors. (The derived algorithms can of course be implemented for a parallel or distributed environment, but this again only lowers the constants in the complexity.) To illustrate the performance of these new techniques, all chapters contain experiments on synthetic and/or real-life data, and, additionally, we include two applications with more in-depth results. In total five major (groups of) algorithms are derived, each focusing on a different type of tensor and explained in a separate chapter, and one minor, yet important algorithm for the computation of an MLSVD using randomization is included in Appendix A. To allow other researchers to use these new algorithms, software implementations, documentation and demos have been developed and released in Tensorlab 3.0 and 4.0.

This thesis comprises three major parts: introduction, algorithms and applications. The elaborate introduction part consists of three chapters, including this chapter which provides a general introduction. Chapter 2 gives a more technical yet broadly accessible overview of the optimization framework used in this thesis, as well as of techniques to extend the results to coupled and constrained tensor decompositions in a (structured) data fusion framework. Moreover, an overview of different techniques for decomposing large-scale tensors is given. Chapter 3 is a conceptual introduction explaining the curse of dimensionality and how to alleviate or break this curse using decompositions of incomplete tensors.

[5] When talking about complexity, we mean the per-iteration complexity for optimization-based algorithms.

The second part discusses five techniques that we propose to tackle the curse by using a compressed representation of the data. This can be an incomplete tensor (Chapter 4), a tensor of which blocks are sampled repeatedly (Chapter 5), a structured tensor, i.e., a tensor which can be represented using fewer parameters than the number of entries (Chapter 6), a tensor in which new slices arrive at a certain rate (Chapter 7), or a tensor which is defined implicitly as the solution of a linear system (Chapter 8). For each type, efficient algebraic and/or optimization-based algorithms are derived and illustrated.

In the third part, we explore two applications in depth. First, we explain in Chapter 9 how incomplete tensors and linear constraints can be used to avoid the expensive data generation part when simulating the evolution of microstructures in multicomponent alloys. Second, the face recognition example briefly mentioned in Chapter 8 is elaborated in Chapter 10 and we illustrate how implicitly determined tensors can play a role in machine learning applications, a strategy that we successfully repeated for irregular heart beat classification [38].

Every subsequent chapter is a slightly adapted version of a book chapter or a paper that has been submitted for review and contains research results obtained by myself in collaboration with various coauthors. As such, every chapter is self contained with a motivation, preliminaries, literature overview, methods, experiments and discussion.

1.3.1 Detailed overview

A brief introduction to all chapters is given below. The overall structure of the main chapters in this thesis is shown in Figure 1.3.

Part I: Introduction

In the remainder of this introduction part, a technical introduction and a conceptual introduction are given. In Chapter 2 we give a broadly accessible tutorial on how a numerically sophisticated optimization framework for data fusion problems can be implemented for tensor problems. In practice, the decomposition in rank-1 (CPD) or low multilinear rank (LMLRA or BTD) terms is computed by minimizing the least squares error between the given tensor $\mathcal{T}$ and its factorization $\mathcal{M}(\mathbf{z})$ which depends on the variables $\mathbf{z}$:

$$\min_{\mathbf{z}} \frac{1}{2} ||\mathcal{M}(\mathbf{z}) - \mathcal{T}||^2 . \tag{1.1}$$

For example, when computing a CPD of a third-order tensor we have $\mathcal{M}(\mathbf{z}) = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!]$ with $\mathbf{z} = \left[\text{vec}\left(\mathbf{A}^{(1)}\right); \text{vec}\left(\mathbf{A}^{(2)}\right); \text{vec}\left(\mathbf{A}^{(3)}\right)\right]$. The Gauss–Newton (GN) algorithm has attractive properties for computing $\mathbf{z}$ for tensor problems: it converges fast and allows the multilinear structure to be exploited elegantly [260]. We show step-by-step how this framework can be

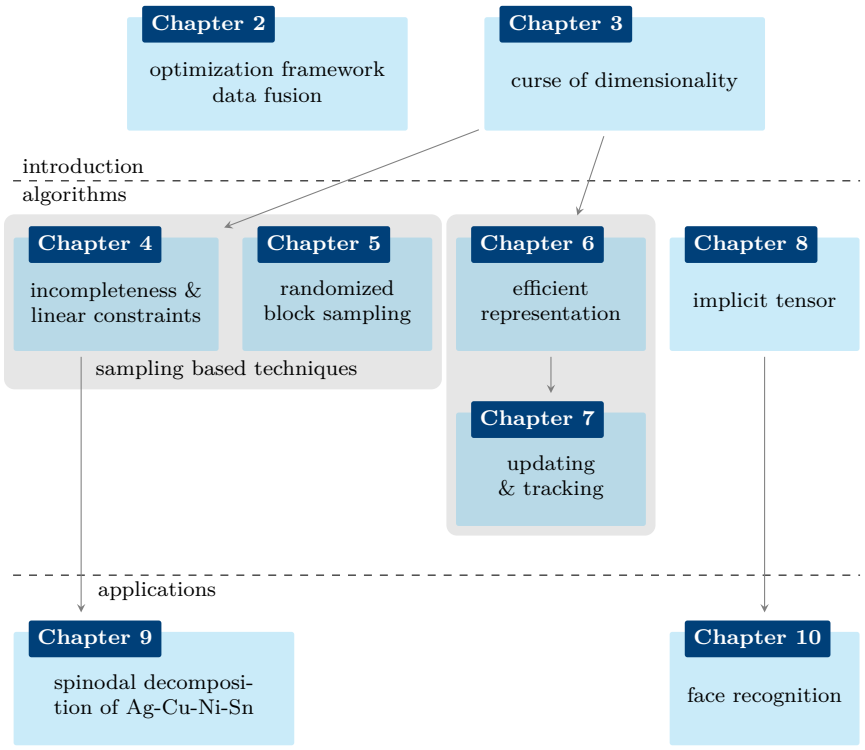


Figure 1.3: Schematic overview of the relation between chapters. *As all main chapters use the optimization framework from Chapter 2, the arrows are omitted.*

generalized to include parametric, symmetry and bound constraints, and how multiple tensors and matrices can be factorized jointly by coupling factor matrices or their underlying variables. While the least squares formulation in (1.1) makes the statistical assumption of normally distributed residuals, other assumptions can be more suitable, e.g., for count or audio data. To accommodate these assumptions, we present a simple extension to Kullback–Leiber and Itakura–Saito divergences. We provide pointers to uniqueness results for coupled matrix and/or tensor decompositions and conclude this chapter by an overview of algorithms to compute the CPD of large-scale tensors.

Being higher-order generalizations of vectors and matrices, tensors tend to grow quickly due to the higher number of modes. In fact, the number of entries in an Nth-order tensor, and therefore the storage and computational complexity, increases exponentially in N. This exponential increase, and the problems associated with it, is often termed the curse of dimensionality. In Chapter 3, we explain that replacing a tensor by a decomposition, e.g., a CPD, an LMLRA or a TT approximation, in further analyses or computations can alleviate or even break this curse, as these representations have only a linear dependency on the order, or an exponential dependency with lower constants. For example, only the N factor matrices of the CPD of a tensor need to be stored or handled. However, the computation of this decomposition may still be subject to the curse. In this case, the curse can be alleviated or broken again by exploiting incomplete tensors. In the philosophy of compressed sensing, only few entries are sampled — possibly adaptively — and used to compute the decomposition through iterative techniques. We discuss cross approximation techniques which are used in scientific computing to compute, e.g., a TT approximation of a tensor with more entries than the number of atoms in the universe [217]. The concepts are illustrated for a multidimensional harmonic retrieval problem in which few samples are taken. Furthermore, a first example from materials science is given, in which the melting temperature of an alloy with ten compounds is modeled. Given the high order of the resulting tensor, it is impossible to sample all $\mathcal{O}\left(10^{18}\right)$ entries and we show that using 100 000 samples is sufficient to accurately represent the tensor using a low-rank CPD with approximately 4 500 variables, while the computation takes only a few minutes.

Part II: Algorithms

While Chapter 3 illustrates the importance of compressed sensing type approaches and tensor decompositions in order to alleviate or even break the curse of dimensionality, we discuss the necessary algorithms for computing a CPD starting from an incomplete tensor in Chapter 4. We develop a new algorithm (CPDI) that has a complexity linear in the number of known or sampled entries building upon the GN framework from Chapter 2. By using the correct Gramian that takes the incompleteness into account and a sta-

tistical preconditioner, we show that CPDI outperforms the state-of-the-art approaches from [6], [262], especially for difficult problems with few samples per variables, or structured missing entries, which are the interesting cases in a large-scale setting. When few entries are known, additional assumptions are often made, e.g., that the factor vectors are smooth or are a (sparse) combination of elements in a dictionary. Both examples can be expressed using linear constraints, i.e., by assuming that each factor $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$, can be written as $\mathbf{B}^{(n)}\mathbf{C}^{(n)}$ in which $\mathbf{B}^{(n)}$ is a known matrix and $\mathbf{C}^{(n)}$ is an unknown coefficient matrix. We develop two specialized algorithms for decomposing incomplete tensors into rank-1 terms with such linear constraints on each factor: the data-independent CPDLI algorithm which has a per-iteration complexity independent of the number of samples by using a precomputed projection of the data, and a data-dependent algorithm which has a complexity linear in the number of samples. The effectiveness of the combination of linear constraints and incomplete tensors is illustrated for a materials science example in which the Gibbs free energy of an alloy is modeled. This application is explained in more detail in Chapter 9.

A different sampling strategy is adopted in Chapter 5: instead of using a fixed sample of known entries, a random block of entries is chosen in every iteration, leading to the randomized block sampling (RBS) algorithm for the CPD. By using blocks, or subtensors, efficient (matrix-free) implementations for dense tensors can be reused as only few variables are affected thanks to the 'locality principle' of a CPD, and a sufficiently good approximation of the Hessian can be constructed, leading to fast convergence. (This is in contrast to stochastic gradient descent, which usually has mere sublinear convergence.) We show that using a simple step restriction schedule, the error can be reduced almost down to the level of the full tensor decomposition, while requiring a fraction of the computation time and memory. A stopping criterion based on the Cramér–Rao bound is derived. The RBS algorithm is illustrated on a standard laptop for a synthetic dataset requiring 8 TB of storage, and a real-life 'electronic nose' dataset of 12.5 GB, both of which can be decomposed in a few minutes.

Orthogonal compression is a well-known technique to speed up the computation of a CPD: following the CANDELINC model, it is sufficient to decompose the core tensor [46], [57]. Unfortunately, this compression is only valid for unconstrained and uncoupled[6] models. For example, orthogonal compression using a factor $\mathbf{U}$ does not preserve nonnegativity constraints, i.e., $\mathbf{A} = \mathbf{U}\mathbf{C} \geq \mathbf{0}$ does not imply that $\mathbf{C} \geq \mathbf{0}$. Similarly, by exploiting the structure of sparse tensors, the complexity can be reduced to linear in the number of nonzeros. Both compression and sparse tensors are actually examples of structured tensors, i.e., tensors that can be represented efficiently

[6] If the tensors are coupled but unconstrained otherwise, joint compression can be used, such that both mode-n spaces are preserved [49], [83].

using fewer parameters than their total number of entries. In Chapter 6, we present a framework to compute tensor decompositions such as the CPD, LL1, LMLRA or BTD, while exploiting this structure. The key is to rewrite the objective function and gradients such that a small number of core operations become apparent: the norm, inner product, and matricized tensor times Khatri–Rao or Kronecker product. We show that implementing these specific operations can be done transparently for tensor optimization software, hence constraints and coupling can be handled trivially as the original factor matrices are kept. We show this for large-scale nonnegative tensor factorization of low and high-order tensors using (randomized) MLSVD and TT compression steps. We discuss the blind separation of exponential polynomials through implicit Hankelization using 500 000 samples, which would require almost a terabyte of storage if the tensor would be generated explicitly. This is possible through the concept of implicit tensorization[7], which allows tensorization techniques to be used while keeping vector or matrix complexity.

Apart from high volumes, big data is also characterized by a high velocity. New data can, for example, arrive frequently and the analysis, which may involve tensor decompositions, should be done in real time. As the amount of data grows fast, storing all data may become problematic. For dynamically changing processes, old data can grow stale and should be removed or receive a reduced importance automatically. In Chapter 7, we present a GN type algorithm that fulfills these needs by building upon the structured tensor framework developed in Chapter 6: the CPD computed from all data at time step $k-1$ is used as a substitute for this 'old' data, and at time step k, the factor matrices computed at time step $k-1$ are used together with the new slice(s) of data as the efficient representation. An algebraic initialization is used which exploits the fact that the change of the factor matrices is limited. We show that a solution nearly as good as batch methods can be achieved using a small fraction of the computation and memory cost, as only factor matrices and a new slice are processed and stored in memory. Using rectangular and exponential windowing strategies, we show that dynamically changing decompositions can be tracked.

In the case of incomplete tensors, the compressed measurement is simply a selection of entries from a tensor. In Chapter 8 more generalized compressed measurements are taken into account by solving

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{subject to} \quad \operatorname{unvec}(\mathbf{x}) = \mathcal{X} = \left[\!\left[\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right]\!\right], \tag{1.2}$$

for $\mathbf{U}^{(n)}$, $n = 1, \ldots, N$. Hence, $\mathbf{b}$ is a compressed measurement of the tensor $\mathcal{X}$ using the known matrix $\mathbf{A}$. Alternatively, one can view (1.2) as a decomposition of a tensor that is given implicitly as a solution of a linear system

[7] Compared to scientific computing literature, a broader definition of tensorization is used here, as it entails all techniques that map vector and matrix data to a tensor.

of equations. Also, in the case $\mathcal{X}$ is a rank-1 tensor, (1.2) is a multilinear generalization of linear systems of equations: instead of $\mathbf{A}\mathbf{x} = \mathbf{A} \cdot_2 \mathbf{x}^{\mathrm{T}} = \mathbf{b}$, we have, e.g., for $N = 3$ and $\mathbf{x} = \mathbf{w} \otimes \mathbf{v} \otimes \mathbf{u}$

$$\mathcal{A} \cdot_2 \mathbf{u}^{\mathrm{T}} \cdot_3 \mathbf{v}^{\mathrm{T}} \cdot_4 \mathbf{w}^{\mathrm{T}} = \mathbf{b}.$$

The straightforward way of computing the factors $\mathbf{U}^{(n)}$, $n = 1, \ldots, N$, is by solving system (1.2) for $\mathbf{x}$ and then reshaping $\mathbf{x}$ into a tensor which can then be decomposed using algebraic or optimization-based algorithms. However, we show that this approach fails if $\mathbf{A}$ does not have full column rank, i.e., $\mathbf{A}$ is fat and/or rank-deficient. Moreover, numerical errors can accumulate using this naive, two-step approach. In Chapter 8, we propose algebraic (in the case $R = 1$) and optimization-based algorithms to compute the decomposition directly as a constrained problem. We derive generic unique conditions for the resulting decomposition: for random $\mathbf{A}$ and $\mathbf{U}^{(n)}$, $n = 1, \ldots, N$, the factor matrices can be recovered uniquely with probability one if the number of equations, i.e., the number of rows of $\mathbf{A}$, is strictly greater than the number of free variables. The performance of the algorithms is illustrated using synthetic data and the general applicability of problem (1.2) is shown for three examples from diverse application domains: a face recognition problem from machine learning, the determination of tensors with prescribed multilinear singular values which allows the theoretical properties of tensors to be investigated, and a BSS problem for convolutive mixtures with constant modulus signals. The detection of irregular heartbeats using electrocardiograms (ECG) is another example handled in a separate paper [38].

Part III: Applications

The relevance of the newly developed algorithms is already illustrated for a variety of applications in Parts I and II: a recommender system using GPS data (Chapter 2), modeling the melting point (Chapter 3) and Gibbs free energy of alloys with many components (Chapter 4), classification of hazardous gasses (Chapter 5), large-scale BSS of exponential polynomials (Chapter 6) and convolutive mixtures of constant modulus signals (Chapter 8), face recognition (Chapter 8) and the computation of tensors with prescribed multilinear singular values (Chapter 8). In this part, we focus on two specific applications: overcoming the curse of dimensionality for high-performance alloys in materials science and the recognition of faces using Kronecker product equations[8] (KPE).

The microstructure of a material can strongly influence its physical properties such as strength, hardness, temperature behavior and so on. To simulate the evolution of these microstructures, the phase-field method can be used, which builds on two types of partial differential equations (PDEs). An ad-

[8] A KPE problem is identical to a rank-1 LS-CPD.

vantage of this method is that the driving force is the Gibbs (free) energy, which can be formulated in terms of the composition of the microstructure, the pressure and the temperature, and can be computed using CALPHAD models. For high-entropy alloys, superalloys or lead-free solders, the number of compounds, and therefore the number of composition variables, can be high, which becomes problematic as each composition variable corresponds to a mode in the tensor with Gibbs free energy values: the storage requirements for all combinations of compositions scales exponentially in the number of compounds. In Chapter 9, we build on the results from Chapters 3 and 4 to break this curse of dimensionality: from few samples a low-degree multivariate polynomial model is created using the CPDLI algorithm. This model replaces the expensive data generation routine successfully, as is verified by simulating the spinodal decomposition of a Ag-Cu-Ni-Sn alloy in its liquid phase.

Face recognition is a nonobtrusive biometric method that can be used to unlock your phone, to identify persons, for tracking purposes and so on. If a database with images of persons under varying parameters such as illumination, viewing angle or expression, is available, these (vectorized) images can be stored in a tensor with pixels, persons and some other parameters as modes. The TensorFaces approach proposed in [296] computes the truncated MLSVD of this tensor and a person in a new image is recognized by computing a person coefficient which is compared against the database. In Chapter 10, we show how the estimation of this person coefficient can be cast as a KPE as defined in Chapter 8. By solving a single KPE rather than many linear systems, a higher recognition rate is achieved and new persons can be added without requiring images for all illuminations, viewing angles, etc. Via coupled KPEs the performance can be improved further as multiple images of the person to be recognized under varying conditions are used.

Conclusion, appendices and software

In Chapter 11, an overview of the various algorithms developed in this thesis is given and their properties are discussed and compared. The main contributions from this thesis are summarized per chapter, and pointers to interesting future research directions are given.

This thesis contains two appendices. Appendix A gives a brief overview of the history and philosophy underlying Tensorlab, the Matlab toolbox for tensor computations and complex optimization developed in our research group, and discusses the algorithms and techniques that have been added in the third release of Tensorlab in March 2016. New faster and/or more accurate computational kernels for incomplete, structured, coupled and symmetric tensor decompositions are discussed, as well as specialized algebraic and optimization-based algorithms for the decomposition in multilinear rank $(L_r, L_r, 1)$ terms. Large-scale techniques such as RBS (Chapter 5) and a

new randomized MLSVD algorithm are presented. In Appendix B detailed derivations for all expressions for the data-dependent and data-independent versions of the CPDLI algorithms are discussed and the rationale behind the preconditioner is explained.

Software is an important aspect to bridge the gap between theory and applications. Therefore, we have released the third and fourth[9] version of Tensorlab. We have integrated the algorithms developed in this thesis into the larger structured data fusion framework, which allows most[10] of these algorithms to be used to solve coupled and constrained factorization problems. Apart from new large-scale algorithms and a more user-friendly SDF language, we have included new tensorization methods, specialized algorithms for the decomposition in multilinear rank $(L_r, L_r, 1)$ or $(M_r, N_r, \cdot)$ terms, support for other divergences, improved solvers for coupled and symmetric problems, support for systems with tensor-structured solutions, new visualization routines and an initial graphical user interface. An elaborate list of new features and changes can be found at www.tensorlab.net/versions.html. Extensive documentation[11] and tutorials[12] have been written to help users navigate all available techniques.

[9] The release of Tensorlab 4.0 is expected in early summer 2018.
[10] The exception is the RBS method, which requires specialized sampling methods.
[11] www.tensorlab.net/doc
[12] www.tensorlab.net/demos

Numerical optimization-based algorithms for data fusion

2

ABSTRACT | Combining various sources of information to discover hidden patterns is key in data analysis. These sources can often be represented as matrices and/or multiway arrays, or tensors, which can be factorized jointly, e.g., as sums of simple terms, to gain insight into the data. In this chapter, an overview of (the rationale behind) numerically well-founded optimization techniques based on a Gauss–Newton framework is given, which has superior convergence properties and allows all multilinear structure to be exploited. Prior knowledge in the form of parametric, box or soft constraints as well as regularization can be incorporated easily. We show how matrices and/or tensors can be coupled through (partially) shared factors or through common underlying variables. The framework is further extended to more general divergences allowing more suitable statistical assumptions. Finally, as tensor problems become large-scale quickly due to the curse of dimensionality, techniques used to alleviate or overcome this curse are discussed.

This chapter is based on N. Vervliet and L. De Lathauwer, "Numerical optimization based algorithms for data fusion", Technical Report 18-11, ESAT-STADIUS, KU Leuven, Belgium. (Accepted)., 2018.

2.1 Introduction

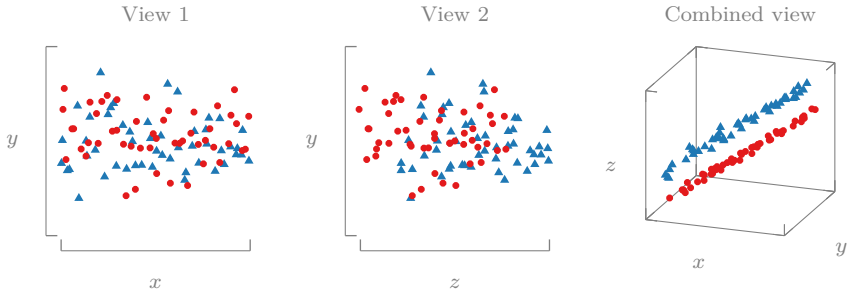


Figure 2.1: Clustering the points from a 3D space is impossible when only one of the left two views of the data is given. However, if the views are analyzed together, it becomes clear that both datasets can be separated by a plane as shown in the combined view.

Consider the two views of two point clouds in a 3D space in Figure 2.1. When looking at either one of the views separately, it is impossible to distinguish the cloud of triangles from the cloud of circles. However, when the information from both views is combined, a 3D view can be constructed (Figure 2.1, right) and it becomes clear that both point clouds can be separated by a plane. In this simple example two views are *fused*, which fits in the larger framework of jointly analyzing multiple datasets. This is prevalent in data analysis as cheap measurement hardware and an enormous increase in computational power fuel the current information age, and have led to gargantuan amounts of data from very heterogeneous sources. To discover useful insights, combining information from all these sources is key as shown in chemometrics [290], neuroscience [9], [288], link prediction [105], [312], multidimensional harmonic retrieval [263], [269], multirate sampling [269], array processing [268], and so on.

The number of dimensions or *order* of the data are an important source of variation: while one and two-way data, which can be represented naturally as vectors and matrices, are commonly used, many data sources are in fact multiway and can be represented by multiway arrays or *higher-order tensors* [65], [243]. To discover latent factors, these tensors can be factorized into simple terms such as a rank-1 term or a low multilinear rank term, leading to decompositions such as the canonical polyadic decomposition (CPD) [56], [139], the decomposition in multilinear rank-$(L_r, L_r, 1)$ terms (LL1) [48], [77], the multilinear singular value decomposition (MLSVD) [78], [284] or the block term decomposition (BTD) [77]. Many of these tensor decompositions are especially attractive because of their mild uniqueness conditions [65], [243]. This has, for example, led to tensorization techniques which transform vector

and matrix data to higher-order tensors [65], [85]. To improve interpretability and to further relax uniqueness conditions, additional constraints such as nonnegativity, orthogonality or symmetry can be added [65], [67], [181], [243], [265], [270]. These constraints can be seen as a form of prior knowledge. Similarly, by jointly analyzing multiple matrices and tensors, milder uniqueness conditions can be derived. (Pointers to such uniqueness results are given in section 2.5.)

While algebraic or semi-algebraic methods exist for (coupled and constrained) CPDs [99], [183], [264], [265], [270], [271], most methods are based on numerical optimization of a nonlinear least squares (NLS) objective function [3], [5], [45], [47], [107], [139], [218], [220], [226], [258], [262], [278], [306]. Rather than using an alternating least squares (ALS) approach, which is popular thanks to its simplicity of implementation and its speed for simple problems, we focus on Gauss–Newton (GN) type algorithms. These GN algorithms have several advantages over ALS. First, GN algorithms using a trust region converge to a stationary point, which is usually a (local) minimum, for any starting point under mild conditions. While ALS often converges in practice, its convergence is proven only for specific cases [287]. If the algorithm converges, GN converges often quadratically, while ALS converges only linearly. In practice, GN is often more robust, meaning that for many problems fewer initializations are required [5], [260], [278], and GN is less susceptible for swamps, i.e., long periods of little improvement [72], [200], thanks to the use of (approximate) curvature information [260]. For the unconstrained CPD, inexact GN algorithms have the same asymptotic per-iteration complexity as ALS [260], but GN requires many fewer iterations. In contrast to ALS, which breaks the multilinear structure up in N substeps, GN type algorithms exploit all structure in every iteration. Finally, the GN framework allows constraints, symmetry, coupling and regularization to be implemented easily [262], as illustrated in this chapter.

As the NLS objective implicitly assumes normally distributed residuals, objective functions based on other divergences may be more suitable if other statistical assumptions are more appropriate. For example, in the case of count data, a Poisson distribution may be more appropriate and a Kullback–Leibler (KL) divergence can be used instead [62], [136]. The KL divergence can also be used to improve music reconstruction [310]. More general distributions such as Tweedie distribution are discussed in combination with coupled factorizations in [69], [249], [310]. For nonnegative tensor factorization, algorithms using alpha and beta divergences are discussed in [66], [167], [225].

Apart from a large variety of datasets, one often has to deal with a large volume as well. Especially in the case of higher-order tensors, the cost of constructing, storing and performing computations with large-scale tensors can be daunting. This is due to the curse of dimensionality: the number of entries increases exponentially with the order of the tensor. To deal with this

curse, various techniques have emerged going from sampling, exploitation of structure such as sparsity or structure that results from implicit tensorization, randomization and parallelization. An overview of such techniques is given in section 2.6.

2.1.1 Outline

After a discussion of the notation and some definitions in the remainder of this section, the most important optimization concepts are discussed in section 2.2, including a derivation of quasi-Newton and Gauss–Newton algorithms with line search or a trust region, and inexact variants which are important for large-scale implementations. The GN algorithm is then specialized for the computation of the unconstrained CPD in section 2.3 and is compared to ALS. In subsection 2.3.4, an extension to more general divergences including Kullback–Leibler and Itakura–Saito is made. Section 2.4 discusses how parametric constraints and box constraints can be incorporated easily in the framework. The use of regularization to implement soft constraints is discussed, as well as imposing symmetry. In section 2.5 some examples of coupling are discussed and a brief overview of uniqueness results is given. We also show how both hard and approximate coupling can be implemented in the GN framework. As many tensor problems are large-scale, an overview of techniques to handle these large-scale tensors is given in section 2.6.

2.1.2 Notation and definitions

To denote scalars, (column) vectors, matrices and tensors, the notations a, $\mathbf{a}$, $\mathbf{A}$ and $\mathcal{T}$ are used, respectively. For example, the ith entry of a vector $\mathbf{a}$ is denoted as a_i, and the rth column of a matrix $\mathbf{A}$ by $\mathbf{a}_r$. For simplicity of notation, only real, third-order tensors with dimensions $I \times J \times K$ are used in this chapter, i.e., $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$. A polyadic decomposition (PD) of $\mathcal{T}$ with factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$ and $\mathbf{C} \in \mathbb{R}^{K \times R}$ is denoted as $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, which is a shorthand notation for

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

with $\otimes$ the outer product. If R is minimal, the decomposition is called canonical[1] (CPD) and R is the rank of the tensor. In practice, a rank-R approximation is often computed. The results in this chapter can be extended

[1] Even though determining the rank R is an NP-hard problem, one should check if (chosen) number of rank-1 terms R is actually minimal to use the term CPD. In this thesis, we assume the PD is probably canonical and use the term CPD in accordance with the terminology used in the application domain to avoid aggravating the terminology.

easily to complex and/or higher-order tensors; see [260], [262]. A mode-n vector is the generalization of a column ($n = 1$) or a row ($n = 2$) and is defined by fixing all but one index in $\mathcal{T}$, e.g., the mode-3 vectors are defined as $\mathcal{T}(i, j, :)$ using Matlab style notation. A mode-n unfolding of a tensor $\mathcal{T}$ is defined as the matrix $\mathbf{T}_{(n)}$ collecting all mode-n vectors as its columns, ordered such that the first index not equal to n runs faster than the second, e.g., $\mathbf{T}_{(1)}(:, (k-1)K + j) = \mathcal{T}(:, j, k)$ and $\mathbf{T}_{(2)}(:, (k-1)K + i) = \mathcal{T}(i, :, k)$. The vectorization operator $\text{vec}(\mathcal{T})$ stacks all mode-1 (column) vectors into a column vector. The reverse operation $\text{unvec}(\mathbf{t})$ reshapes $\mathbf{t}$ into a tensor $\mathcal{T}$ with dimensions $I \times J \times K$. The following products are required. The tensor matrix product in mode n is denoted by $\mathcal{T} \cdot_n \mathbf{A}$ and is defined in terms of the mode-n unfolding as $(\mathcal{T} \cdot_n \mathbf{A})_{(n)} = \mathbf{A}\mathbf{T}_{(n)}$. The Kronecker product and Khatri–Rao product are denoted by $\otimes$ and $\odot$, respectively, and are defined for matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{p \times q}$ and $\mathbf{C} \in \mathbb{R}^{l \times n}$ as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix},$$
$$\mathbf{A} \odot \mathbf{C} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{c}_1 & \mathbf{a}_2 \otimes \mathbf{c}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{c}_n \end{bmatrix}.$$

The Hadamard, or element-wise, product is denoted by $*$. The transpose, pseudoinverse and Frobenius norm are denoted by $\cdot^{\mathrm{T}}$, $\cdot^{\dagger}$ and $||\cdot||_{\mathrm{F}}$, respectively. $\mathbf{I}_n$ denotes the $n \times n$ identity matrix and $\mathbf{1}_n$ is a length-n column vector with ones. The column-wise concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\begin{bmatrix}\mathbf{a}; \mathbf{b}\end{bmatrix}$ and is a shorthand for $\begin{bmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$.

2.2 Numerical optimization for tensor decompositions

Matrix decompositions and tensor decompositions such as the MLSVD [78] or a tensor train (TT) approximation [211] are usually computed via algebraic algorithms such as the singular value decomposition (SVD). Similarly, a CPD can be computed using algebraic methods such as the generalized eigenvalue decomposition (GEVD) [99], [183]. The performance of these algebraic methods depends on the tensor to be decomposed and on which slices are used. In practice, an optimization approach is often taken as it has several advantages: it allows all multilinear information to be exploited, it is highly efficient and it is more robust to noise. The result of the algebraic algorithm can still be used as an initialization, though. Due to its importance for tensor decompositions into rank-1 terms, a high level overview of basic optimization concepts is given in this section; see, e.g., [160], [209] for a more in-depth

discussion and [8], [258] for extensions to complex tensors.

Among the optimization techniques, nonlinear least squares (NLS) methods are commonly used in a tensor context. The objective of an NLS problem is to find $\mathbf{z}^* \in \mathbb{R}^N$ such that the squared error between a data vector $\mathbf{t}$ and a nonlinear model $m(\mathbf{z})$ is minimized. Mathematically, this can be written as

$$\mathbf{z}^* = \arg\min_{\mathbf{z}} f(\mathbf{z}) \qquad \text{with} \qquad f(\mathbf{z}) = ||m(\mathbf{z}) - \mathbf{t}||_{\mathrm{F}}^2 , \tag{2.1}$$

in which $f(\mathbf{z})$ is called the objective or loss function.

Example 1: When computing a rank-R CPD of a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ with $\mathbf{t} = \mathrm{vec}\,(\mathcal{T})$, the $N = (I+J+K)R$ variables are the factor matrices, i.e., $\mathbf{z} = \big[\mathrm{vec}\,(\mathbf{A})\,;\mathrm{vec}\,(\mathbf{B})\,;\mathrm{vec}\,(\mathbf{C})\big]$. The model is then given by $m(\mathbf{z}) = \mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket)$.

To find a (possibly local) optimizer $\mathbf{z}^*$ for the loss function (2.1), an initial guess $\mathbf{z}_0$ is iteratively refined by taking a step of length α_k in direction $\mathbf{p}_k$:

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \alpha_k \mathbf{p}_k .$$

This is repeated until some stopping criterion is satisfied. Two important approaches for choosing the step length and direction are discussed in subsection 2.2.1. Both approaches rely on a local linear or quadratic approximation $\tilde{f}$ of the objective function to determine the step direction, as described in subsection 2.2.2. This approximation leads to a linear system which can be solved using direct or iterative techniques. The latter are critical to implement highly efficient algorithms for larger tensor problems; see subsection 2.2.3.

2.2.1 Line search and trust region

Line search and trust region algorithms are the two main approaches to update the variables such that $\mathbf{z}_k$ is closer to a (possibly local) optimum for the objective (2.1). In the case of line search, an update is computed as

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \alpha_k \mathbf{p}_k , \tag{2.2}$$

in which $\mathbf{p}_k$ is the step direction determined by solving an easier subproblem, denoted by $\tilde{f}(\mathbf{p}_k)$, and α_k is the step length along the direction $\mathbf{p}_k$. After finding $\mathbf{p}_k$, the optimal α_k along the line determined by $\mathbf{p}_k$ is found by solving the following optimization problem in a single variable:

$$\alpha_k = \arg\min_{\alpha > 0} f(\mathbf{z}_{k-1} + \alpha \mathbf{p}_k).$$

Usually, it is not necessary to find the optimal α as long as the objective function is decreased sufficiently. See [209] for more details on conditions for sufficient decrease.

If a trust region approach is used, the step direction and length are determined simultaneously by solving the constrained subproblem

$$\mathbf{p}_k = \arg\min_{\mathbf{p}} \tilde{f}(\mathbf{p}) \quad \text{subject to} \quad ||\mathbf{p}||_{\mathrm{F}} \leq \Delta, \tag{2.3}$$

in which $\tilde{f}$ again is a local approximation to the objective function. The variables are updated as

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

hence $\alpha_k = 1$. While the constrained optimization problem (2.3) can be solved exactly, other approaches such as the dogleg method and plane search are often used as well; see, e.g., [209].

2.2.2 Determining step direction $\mathbf{p}_k$

Compared to nonlinear problems, linear problems are usually easy to solve. Therefore, the nonlinear objective function $f(\mathbf{z})$ is locally approximated by a function $\tilde{f}$ such that $\mathbf{p}$ can be found from a linear system. More mathematically, the function $f(\mathbf{z})$ is locally approximated by a Taylor series at the current guess $\mathbf{z}_k$:

$$f(\mathbf{z}_k + \mathbf{p}) \approx f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla_{\mathbf{z}} f(\mathbf{z}_k)}_{\mathbf{g}_k} + \frac{1}{2}\mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla^2_{\mathbf{z}} f(\mathbf{z}_k)}_{\mathbf{H}_k} \cdot \mathbf{p} + \dots \tag{2.4}$$

in which $\mathbf{p} = \mathbf{z} - \mathbf{z}_k$. The derivatives $\mathbf{g}_k$ and $\mathbf{H}_k$ are the gradient and Hessian, respectively, both evaluated at $\mathbf{z}_k$:

$$\mathbf{g}_k = \begin{bmatrix} \frac{\partial f}{\partial z_1} & \frac{\partial f}{\partial z_2} & \cdots & \frac{\partial f}{\partial z_N} \end{bmatrix}^{\mathrm{T}},$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial^2 f}{\partial z_1^2} & \frac{\partial^2 f}{\partial z_1 \partial z_2} & \cdots & \frac{\partial^2 f}{\partial z_1 \partial z_N} \\ \frac{\partial^2 f}{\partial z_2 \partial z_1} & \frac{\partial^2 f}{\partial z_2^2} & \cdots & \frac{\partial^2 f}{\partial z_2 \partial z_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial z_N \partial z_1} & \frac{\partial^2 f}{\partial z_N \partial z_2} & \cdots & \frac{\partial^2 f}{\partial z_N^2} \end{bmatrix}.$$

A quadratic model in $\mathbf{p}$ is obtained by limiting the Taylor series in (2.4) to the first three terms. The optimal value $\mathbf{p}_k$ for $\tilde{f}$ is then given by

$$\mathbf{p}_k = \arg\min_{\mathbf{p}} \tilde{f}(\mathbf{p}) \qquad \text{with} \qquad \tilde{f}(\mathbf{p}) = f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}}\mathbf{g}_k + \frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{H}_k\mathbf{p}.$$

To compute $\mathbf{p}_k$ the gradient of $\tilde{f}$, i.e., $\nabla_{\mathbf{p}}\tilde{f} = \mathbf{g}_k + \mathbf{H}_k\mathbf{p}_k$, is set to zero, which results in the linear system

$$\mathbf{H}_k\mathbf{p}_k = -\mathbf{g}_k. \tag{2.5}$$

The computed step $\mathbf{p}_k$ is called the Newton step.

Unfortunately, the Hessian $\mathbf{H}_k$ is often difficult or expensive to compute explicitly. To overcome this, $\mathbf{H}_k$ is often approximated which results in quasi-Newton (qN) type algorithms such as nonlinear conjugate gradients (NCG) and BFGS, and Gauss–Newton (GN) type algorithms. Some examples of such approximations are:

- For gradient or steepest descent, $\mathbf{H}_k = \mathbf{I}$ and the step direction is simply the direction of the steepest slope.
- For NCG, $\mathbf{H}_k = \mathbf{I} - \gamma\mathbf{p}_{k-1}\boldsymbol{\delta}^{\mathrm{T}}$, i.e., the identity plus a rank-1 correction. The values γ and $\boldsymbol{\delta}$ depend on the current and previous gradient, i.e., $\mathbf{g}_{k-1}$ and $\mathbf{g}_k$, and the previous step $\mathbf{p}_{k-1}$.
- For BFGS, $\mathbf{H}_k = \mathbf{H}_{k-1} + \mathbf{U}_{k-1} + \mathbf{V}_{k-1}$ in which $\mathbf{U}_{k-1}$ and $\mathbf{V}_{k-1}$ are symmetric rank-1 matrices constructed using the previous values for the gradient, for the approximation $\mathbf{H}_{k-1}$, and for the update of $\mathbf{z}$.
- For GN, $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k$ with $\mathbf{J}_k$ the Jacobian matrix, and $\mathbf{H}_k$ is then called the Gramian (of the Jacobian).
- For Levenberg–Marquardt (LM), $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}$ with $\mathbf{J}_k$ the Jacobian matrix and $\lambda \geq 0$ a chosen constant.

Using the Newton step, quadratic convergence can be achieved near local optima under certain conditions, while the convergence is merely linear for gradient descent. In the case of qN algorithms, the convergence improves to superlinear. Quadratic convergence can again be achieved using GN and LM, as the Gramian often approximates the exact Hessian well for many optimization problems. Example 2 and example 3 illustrate the differences in convergence speed. In combination with trust region approaches, it can be shown that GN is globally convergent, meaning that the algorithm will converge to a (local) minimum from any initial value $\mathbf{z}_0$ thanks to the fact that $\mathbf{H}_k$ is positive semidefinite which ensures that $\mathbf{p}_k$ is a descent direction. (In fact, GN with a trust region is related to LM, as it sets λ indirectly through the trust region radius Δ [209].) Moreover, we show in section 2.3

that GN allows the multilinear structure in CPD problems to be exploited elegantly. Because of its good properties, the remainder of this chapter is focused on GN type algorithms.

Example 2 (linear versus quadratic convergence): Consider a tensor $\mathcal{T} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!]$ of size $250 \times 250 \times 250$ with rank 10. The random factor matrices are chosen to create a rather difficult problem and are constructed such that each column has norm one and the inner product with the other columns is 0.8, i.e., $\|\mathbf{a}_r^{(n)}\|_{\mathrm{F}} = 1$ and $\mathbf{a}_r^{(n)^{\mathrm{T}}} \mathbf{a}_s^{(n)} = 0.8$ for $r \neq s$ and $n = 1, 2, 3$. Starting from a random initialization, GN (`cpd_nls`) and ALS (`cpd_als`) from Tensorlab [305] are used to compute the optimum. The former algorithm is able to achieve up to quadratic convergence near a local optimum, while the latter achieves up to linear convergence. A typical convergence profile can be seen in Figure 2.2 and in the table below: while ALS needs about 100 iterations to make the error approximately 100 times smaller, the number of correct digits doubles every iteration using GN until the machine precision limits further improvement at iteration 37.

GN		ALS	
k	f	k	f
34	$2.23 \cdot 10^{-6}$	700	$1.41 \cdot 10^{-15}$
35	$5.08 \cdot 10^{-12}$	800	$2.22 \cdot 10^{-17}$
36	$5.42 \cdot 10^{-24}$	900	$3.09 \cdot 10^{-19}$
37	$4.60 \cdot 10^{-31}$	1000	$4.89 \cdot 10^{-21}$

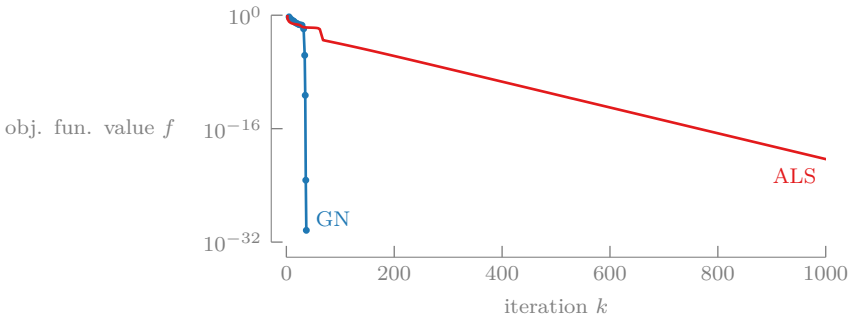


Figure 2.2: As the GN algorithm converges quadratically near a local minimum, only a few iterations are required to converge to the optimum up to machine precision. ALS, which converges linearly, requires many iterations to obtain a similar precision.

Example 3 (second-order information and noise): Consider a rank-10 tensor $\mathcal{T} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!]$ of size $200 \times 200 \times 200$ in which the random factor matrices have norm-one columns and inner product 0.8, i.e., $\|\mathbf{a}_r^{(n)}\|_{\mathrm{F}} = 1$ and $\mathbf{a}_r^{(n)^{\mathrm{T}}} \mathbf{a}_s^{(n)} = 0.8$ for $r \neq s$. Noise is added such that the SNR is 20 dB. Starting from a random initialization, GN (`cpd_nls`) and ALS (`cpd_als`) are again used to compute a rank-10 approximation. In Figure 2.3, the maximal relative error on the factor matrices, i.e.,

$$E_{\mathrm{CPD}} = \max_n \frac{\|\mathbf{A}^{(n)} - \hat{\mathbf{A}}^{(n)}\|_{\mathrm{F}}}{\left\|\mathbf{A}^{(n)}\right\|_{\mathrm{F}}}$$

is reported. (Scaling and permutation indeterminacies are assumed to be resolved.) Initially, ALS improves the function value f faster compared to GN, but ALS requires many iterations with very small changes in f to reduce E_{CPD} to the same level as the GN algorithm. This result also illustrates that one should avoid stopping too early when using ALS. Fast convergence in terms of E_{CPD} is again observed for GN.

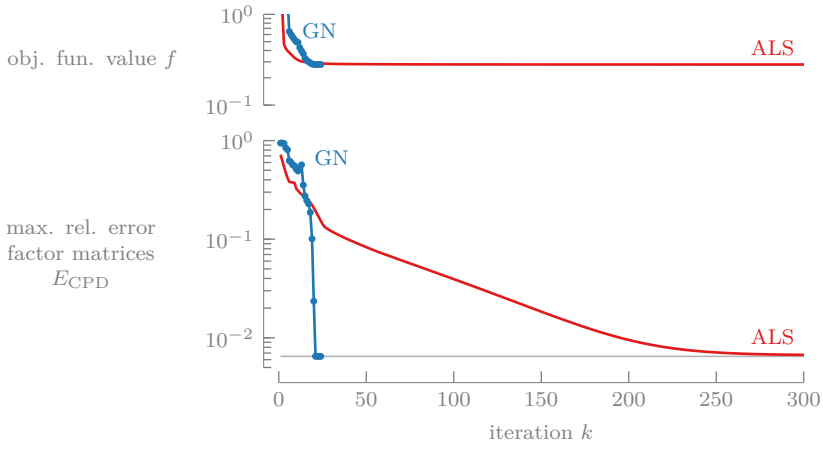


Figure 2.3: While the improvement in objective function value levels off after a few iterations because of the perturbations by noise (SNR is 20 dB) for both GN and ALS, the error on the factor matrices can still be improved. *Thanks to the use of (approximate) second-order information in GN, an accurate solution is found quickly, while many iterations with almost no improvement in the objective function are required for ALS. The results are shown for a rank-10 tensor with correlated factor matrices in all modes.*

2.2.3 Solving $\mathbf{Hp} = -\mathbf{g}$

Each iteration of the optimization algorithm, the solution $\mathbf{p}$ of the system $\mathbf{Hp} = -\mathbf{g}$ is required. Directly solving the system by simply taking the inverse of $\mathbf{H}$ may not be a good idea for a number of numerical reasons and is not even possible if $\mathbf{H}$ is not invertible. This is for example the case when computing a CPD via GN, as $\mathbf{H} = \mathbf{J}^{\mathsf{T}}\mathbf{J}$ has $2R$ zero eigenvalues due to the scaling indeterminacy. To mitigate this, the pseudoinverse

$$\mathbf{p} = -\mathbf{H}^{\dagger}\mathbf{g}$$

or the LDL factorization [119] can be used. (The latter can only be used if $\mathbf{H}$ is symmetric, which is the case for GN.)

The direct techniques outlined above work fine if the number of variables is low (typically a few hundreds), but become expensive for large-scale problems. Therefore, rather than a direct method, an inexact, iterative solver can be used, e.g., conjugate gradients (CG). Starting from an initial guess $\mathbf{p}^{(0)}$, e.g., the Cauchy point [209], the solution of $\mathbf{Hp} = -\mathbf{g}$ is computed using only one matrix-vector product of the form $\mathbf{y}^{(l)} = \mathbf{H}\mathbf{x}^{(l-1)}$ in each CG iteration. In the lth iteration, the guess $\mathbf{p}^{(l)}$ and $\mathbf{x}^{(l)}$ are both updated using a linear combination of their respective previous value and $\mathbf{y}^{(l)}$. (See, e.g., [281] for more details on the CG algorithm). To compute the product $\mathbf{Hx}$, it is not necessary to construct $\mathbf{H}$ explicitly, allowing the structure to be exploited. As shown in section 2.3, the product $\mathbf{Hx}$ can be computed cheaply in the case of a CPD. The CG algorithm requires $\mathrm{it}_{\mathrm{CG}}$ iterations to achieve a certain relative error, e.g., 10^{-6} or 10^{-8}. To improve the convergence and therefore reduce $\mathrm{it}_{\mathrm{CG}}$ and the number of required matrix-vector products $\mathbf{Hx}$, a preconditioner $\mathbf{M}$ is often used, and the following system is solved instead:

$$\mathbf{M}^{-1}\mathbf{Hp} = -\mathbf{M}^{-1}\mathbf{g}.$$

If the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered than those of $\mathbf{H}$, the preconditioned CG (PCG) converges faster [281]. Ideally, $\mathbf{M}^{-1}$ is also cheap to apply.

> **Example 4:** Consider a rank-5 tensor $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of size $40 \times 40 \times 40$ with highly correlated factor vectors and perturbed by noise such that the SNR is 20 dB. Suppose the current guess is $\mathbf{z}_{k-1}$. The step $\mathbf{p}_k$ is computed from $\mathbf{Hp}_k = -\mathbf{g}$ using the pseudoinverse, CG and PCG with a block-Jacobi preconditioner (see subsection 2.3.2). As shown in Figure 2.4, fewer iterations and less time are needed when using PCG. (A stopping tolerance of 10^{-8} on the relative residual is used for the latter two algorithms.)

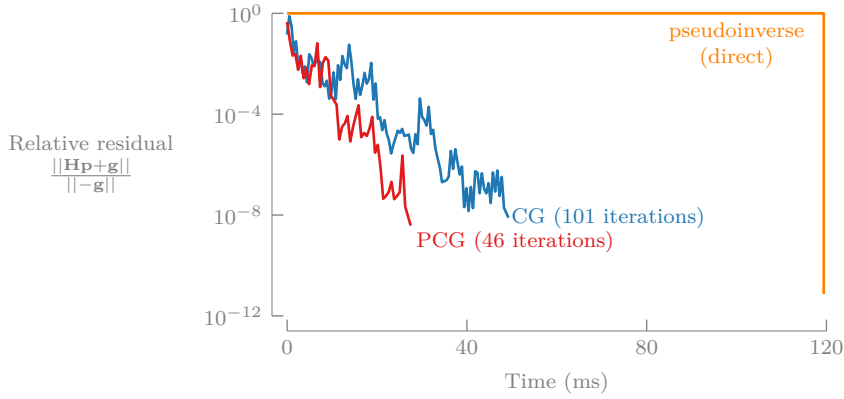


Figure 2.4: While the direct method finds the solution of $\mathbf{Hp} = -\mathbf{g}$ in a single step, it is outperformed by the iterative methods in terms of time. *The preconditioning in PCG reduces the number of iterations needed as well as overall cost, if it is cheap to apply. The Gramian* $\mathbf{H}$ *and gradient* $\mathbf{g}$ *are computed for a random GN iteration when computing a rank-5 CPD of* $40 \times 40 \times 40$ *tensor with highly correlated columns. The stopping tolerance is set to* 10^{-8}.

2.3 Canonical polyadic decomposition

Throughout this chapter, nearly all concepts are derived for Gauss–Newton type algorithms using a trust region. Apart from the favorable convergence properties, a big advantage is that it allows the multilinear structure of a CPD to be exploited easily. In combination with an inexact solver for the system (2.5), an efficient algorithm can be derived, as shown in this section. (Note that one typically uses an optimization framework, e.g., the complex optimization framework [258], [259] which is built-in into Tensorlab [305]. Hence, only relevant expressions for the objective function, gradient, Gramian-vector product and preconditioner are required.)

Concretely, the following objective function is used:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f \quad \text{with} \quad f = \frac{1}{2} \| \underbrace{[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}}_{\mathcal{R}} \|_{\mathrm{F}}^2. \tag{2.6}$$

This is a quadratic objective function in the (vectorized) residual $\mathbf{r} = \mathrm{vec}\,(\mathcal{R})$:

$$f = \frac{1}{2} \mathbf{r}^{\mathrm{T}} \mathbf{r},$$

with $\mathbf{r}$ a multilinear function in the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. This formulation proves useful when deriving the gradient and the Gramian of the

Jacobian.

After a brief overview of important concepts in multilinear algebra and derivatives in subsection 2.3.1, the ingredients for a GN type CPD algorithm are derived in subsection 2.3.2. In subsection 2.3.3, we show how an ALS type algorithm fits in the optimization framework. Finally, more general objective functions are discussed in subsection 2.3.4.

2.3.1 Intermezzo: multilinear algebra

Via multilinear algebra, the gradient and Gramian-vector products required when computing the optimum of (2.6) using GN can be derived without resorting to element-wise expressions. A brief overview of the most important identities needed are listed here.

The following identities involving Kronecker and Khatri–Rao products are used, assuming matrices of compatible dimensions:

$$(\mathbf{A} \otimes \mathbf{B})\mathrm{vec}\,(\mathbf{X}) = \mathrm{vec}\,(\mathbf{B}\mathbf{X}\mathbf{A}^{\mathrm{T}}), \tag{2.7}$$

$$(\mathbf{A} \odot \mathbf{B})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{D}) = (\mathbf{A}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{D}), \tag{2.8}$$

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}. \tag{2.9}$$

To use these identities in combination with unfolded tensors, we define the permutation matrices $\mathbf{\Pi}^{(n)} \in \{0,1\}^{IJK \times IJK}$ which permute the nth mode to the first mode:

$$\begin{aligned}\mathbf{\Pi}^{(1)} &: (i,j,k) \mapsto (i,j,k),\\ \mathbf{\Pi}^{(2)} &: (i,j,k) \mapsto (j,i,k),\\ \mathbf{\Pi}^{(3)} &: (i,j,k) \mapsto (k,i,j).\end{aligned}$$

The inverse operation is $\mathbf{\Pi}^{(n)^{\mathrm{T}}}$ and moves the first mode to the nth mode. Note that $\mathbf{\Pi}^{(n)}$ is a purely mathematical concept and is never constructed explicitly.

Example 5: Given $\mathcal{T} = \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket$, we have

$$\begin{aligned}\mathbf{\Pi}^{(3)}\mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket) &= \mathrm{vec}\,(\llbracket \mathbf{C}, \mathbf{A}, \mathbf{B} \rrbracket)\\ \mathbf{\Pi}^{(2)}\mathrm{vec}\,(\mathcal{T}) &= \mathrm{vec}\,\left(\mathbf{T}_{(2)}\right)\\ \mathbf{\Pi}^{(2)^{\mathrm{T}}}\mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket) &= \mathrm{vec}\,(\llbracket \mathbf{B}, \mathbf{A}, \mathbf{C} \rrbracket).\end{aligned}$$

Example 6 (mtkrprod): A common operation when computing a CPD is the "matricized tensor (times) Khatri–Rao product" (MTKRPROD

or MTTKRP). For example, for the first mode, we have

$$\mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}).$$

Using identity (2.7), this can be rewritten as

$$\operatorname{vec}\left(\mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) = ((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} \otimes \mathbf{I}_I)\operatorname{vec}(\mathcal{T}).$$

Additionally, using permutation matrices, we have for the second mode

$$\operatorname{vec}\left(\mathbf{T}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) = ((\mathbf{C} \odot \mathbf{A})^{\mathrm{T}} \otimes \mathbf{I}_J)\boldsymbol{\Pi}^{(2)}\operatorname{vec}(\mathcal{T}).$$

Similarly, for a CPD, we have

$$\operatorname{vec}\left(\mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}\right) = \boldsymbol{\Pi}^{(2)^{\mathrm{T}}}((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)\operatorname{vec}(\mathbf{B}).$$

Finally, it is worthwhile to refresh some matrix derivatives. For matrices $\mathbf{A}$ and $\mathbf{B}$ with dimensions $I \times J$ and $J \times K$, respectively, we have

$$\frac{\partial \operatorname{vec}(\mathbf{AB})}{\partial \operatorname{vec}(\mathbf{A})} = \mathbf{B}^{\mathrm{T}} \otimes \mathbf{I}_I \qquad \frac{\partial \operatorname{vec}(\mathbf{AB})}{\partial \operatorname{vec}(\mathbf{B})} = \mathbf{I}_K \otimes \mathbf{A}. \tag{2.10}$$

2.3.2 Gauss–Newton type algorithms[2]

An inexact GN algorithm for computing a CPD using objective function (2.6) is derived here. (The algorithm itself is presented in [260].) As standard trust region techniques such as the dogleg method (see [209]) are used, we focus on solving the linear system (2.5) using PCG, which requires the computation of the gradient and Gramian-vector products $\mathbf{Hx}$ as shown in subsection 2.2.3. This subsection is concluded by a brief discussion of a preconditioner.

Gradient

As the objective function f in (2.6) is a quadratic function in the residual $\mathbf{r}$, which is in turn a multilinear function of the variables $\mathbf{z}$, the chain rule can be used to derive the gradient. Recall that the variables are $\mathbf{z} = \left[\operatorname{vec}(\mathbf{A}); \operatorname{vec}(\mathbf{B}); \operatorname{vec}(\mathbf{C})\right]$. The gradient can be partitioned accordingly, i.e., $\mathbf{g} = \left[\mathbf{g}_{\mathrm{A}}; \mathbf{g}_{\mathrm{B}}; \mathbf{g}_{\mathrm{C}}\right]$. The derivative w.r.t. $\operatorname{vec}(\mathbf{A})$ is then given by

$$\mathbf{g}_{\mathrm{A}} = \frac{1}{2}\frac{\partial \mathbf{r}^{\mathrm{T}}\mathbf{r}}{\partial \operatorname{vec}(\mathbf{A})} = \left(\frac{\partial \mathbf{r}}{\partial \operatorname{vec}(\mathbf{A})}\right)^{\mathrm{T}}\mathbf{r} = \mathbf{J}_{\mathrm{A}}^{\mathrm{T}}\mathbf{r}. \tag{2.11}$$

The matrix $\mathbf{J}_{\mathrm{A}}$ is called the Jacobian matrix. The ith row in this Jacobian contains the derivative of the ith residual r_i, $i = 1, \ldots, IJK$, w.r.t. all entries

[2] A derivation of an inexact GN type algorithm solely relying on multilinear algebra is given in this subsection. The result is identical to the algorithm presented in [260].

in $\mathbf{A}$, i.e.,

$$\mathbf{J}_{\mathrm{A}} = \begin{bmatrix} \frac{\partial r_1}{\partial a_{11}} & \frac{\partial r_1}{\partial a_{21}} & \cdots & \frac{\partial r_1}{\partial a_{IR}} \\ \frac{\partial r_2}{\partial a_{11}} & \frac{\partial r_2}{\partial a_{21}} & \cdots & \frac{\partial r_2}{\partial a_{IR}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial r_{IJK}}{\partial a_{11}} & \frac{\partial r_{IJK}}{\partial a_{21}} & \cdots & \frac{\partial r_{IJK}}{\partial a_{IR}} \end{bmatrix}.$$

In the case of a CPD, the Jacobian can be computed easily using (2.10):

$$\mathbf{J}_{\mathrm{A}} = \frac{\partial \mathbf{r}}{\partial \mathrm{vec}\,(\mathbf{A})} = \frac{\partial \mathrm{vec}\left(\mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}\right)}{\partial \mathrm{vec}\,(\mathbf{A})} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I. \tag{2.12}$$

Similarly, the Jacobians w.r.t. the other factor matrices can be computed as

$$\mathbf{J}_{\mathrm{B}} = \mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J\right), \tag{2.13}$$

$$\mathbf{J}_{\mathrm{C}} = \mathbf{\Pi}^{(3)^{\mathrm{T}}} \left((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K\right). \tag{2.14}$$

Finally, by combining (2.11) and (2.12)–(2.14) and exploiting the multilinear identities as in example 6, we find

$$\mathbf{g}_{\mathrm{A}} = \mathrm{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right), \tag{2.15}$$

$$\mathbf{g}_{\mathrm{B}} = \mathrm{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right), \tag{2.16}$$

$$\mathbf{g}_{\mathrm{C}} = \mathrm{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right). \tag{2.17}$$

These MTKRPROD can be computed efficiently without explicitly constructing the Khatri–Rao products $\mathbf{C} \odot \mathbf{B}$ and so on. Various implementations emerged that avoid permuting the tensor in memory [224], [293], that avoid communication [20], that reuse intermediate results [224], that exploit sparsity by only computing rows of the Khatri–Rao product corresponding to nonzero entries [16], [251], or that exploit structure in the tensor such as tensor train or Hankel structure [16], [303].

Gramian-vector products

When using (preconditioned) CG, the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ is solved iteratively: each CG iteration a Gramian-vector product $\mathbf{y} = \mathbf{H}\mathbf{x}$ is computed; see subsection 2.2.3. The Gramian is given by the inner product of the Jacobian with itself, hence

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} \quad \text{with} \quad \mathbf{J} = \begin{bmatrix} \mathbf{J}_{\mathrm{A}} & \mathbf{J}_{\mathrm{B}} & \mathbf{J}_{\mathrm{C}} \end{bmatrix},$$

in which $\mathbf{J}_{\mathrm{A}}$, $\mathbf{J}_{\mathrm{B}}$ and $\mathbf{J}_{\mathrm{C}}$ are defined as in (2.12)–(2.14). The block structure of $\mathbf{J}$ results in a block matrix $\mathbf{H}$ as can be seen in Figure 2.5, which gives

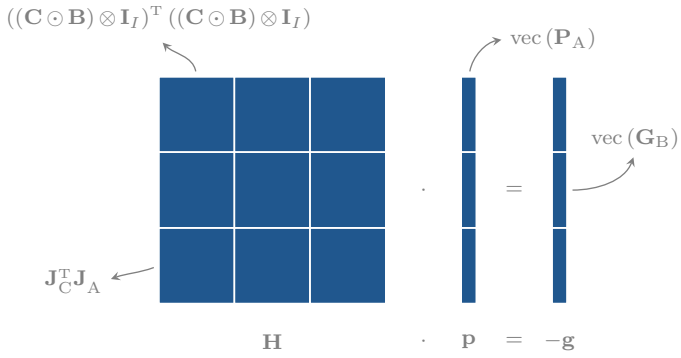


Figure 2.5: Grafical representation of the system $\mathbf{Hp} = -\mathbf{g}$, which is solved for $\mathbf{p}$ every iteration of the GN algorithm.

a graphical overview of the system. If the vectors $\mathbf{y}$ and $\mathbf{x}$ are partitioned according to factor matrices as well, i.e.,

$$\begin{aligned}\mathbf{y} &= \left[\mathbf{y}_\mathrm{A};\quad \mathbf{y}_\mathrm{B};\quad \mathbf{y}_\mathrm{C}\right],\\ \mathbf{x} &= \left[\mathrm{vec}\left(\mathbf{X}_\mathrm{A}\right);\quad \mathrm{vec}\left(\mathbf{X}_\mathrm{B}\right);\quad \mathrm{vec}\left(\mathbf{X}_\mathrm{C}\right)\right],\end{aligned}$$

the products $\mathbf{y}_\mathrm{A}$, $\mathbf{y}_\mathrm{B}$ and $\mathbf{y}_\mathrm{C}$ can be computed as

$$\mathbf{y}_\mathrm{A} = \mathbf{H}_{\mathrm{AA}}\mathrm{vec}\left(\mathbf{X}_\mathrm{A}\right) + \mathbf{H}_{\mathrm{AB}}\mathrm{vec}\left(\mathbf{X}_\mathrm{B}\right) + \mathbf{H}_{\mathrm{AC}}\mathrm{vec}\left(\mathbf{X}_\mathrm{C}\right),$$

and so on, with $\mathbf{H}_{\mathrm{AA}} = \mathbf{J}_\mathrm{A}^\mathrm{T}\mathbf{J}_\mathrm{A}$ etc. By exploiting the multilinear identities from subsection 2.3.1, efficient expressions can be derived easily, e.g.,

$$\begin{aligned}\mathbf{H}_{\mathrm{AA}}\mathrm{vec}\left(\mathbf{X}_\mathrm{A}\right) &= \mathrm{vec}\left(\mathbf{X}_\mathrm{A}((\mathbf{C}^\mathrm{T}\mathbf{C}) * (\mathbf{B}^\mathrm{T}\mathbf{B}))\right),\\ \mathbf{H}_{\mathrm{BC}}\mathrm{vec}\left(\mathbf{X}_\mathrm{C}\right) &= \mathrm{vec}\left(\mathbf{B}((\mathbf{X}_\mathrm{C}^\mathrm{T}\mathbf{C}) * (\mathbf{A}^\mathrm{T}\mathbf{A}))\right).\end{aligned}$$

Note that the complexity of computing $\mathbf{y}$ is low as only inner products between matrices, and matrix products with $R \times R$ matrices are needed. The complexity therefore is a function of the sum of the tensor dimensions, i.e., $\mathcal{O}\left(I + J + K\right)$, which is usually much lower than the complexity of the gradient and function evaluation which both depend on the total number of entries in the tensor, i.e., $\mathcal{O}\left(IJK\right)$.

Example 7: The expression for $\mathbf{H}_{\mathrm{BC}}\mathrm{vec}\left(\mathbf{X}_\mathrm{C}\right)$ can be derived by applying identities (2.7) and (2.8) and properties involving permutation

matrices:

$$\begin{aligned}\mathbf{H}_{\mathrm{BC}}\mathrm{vec}\left(\mathbf{X}_{\mathrm{C}}\right) &= \mathbf{J}_{\mathrm{B}}^{\mathrm{T}} \cdot \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K)\mathrm{vec}\left(\mathbf{X}_{\mathrm{C}}\right) \\ &= \mathbf{J}_{\mathrm{B}}^{\mathrm{T}} \cdot \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}\mathrm{vec}\left(\mathbf{X}_{\mathrm{C}}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}\right) \\ &= ((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)^{\mathrm{T}}\boldsymbol{\Pi}^{(2)} \cdot \mathrm{vec}\left(\mathbf{A}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{B})^{\mathrm{T}}\right) \\ &= ((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)^{\mathrm{T}}\mathrm{vec}\left(\mathbf{B}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{A})^{\mathrm{T}}\right) \\ &= \mathrm{vec}\left(\mathbf{B}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{A})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{A})\right) \\ &= \mathrm{vec}\left(\mathbf{B}((\mathbf{X}_{\mathrm{C}}^{\mathrm{T}}\mathbf{C}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A}))\right).\end{aligned}$$

Preconditioner

By using a preconditioner, the number of CG iterations, and hence the number of Gramian-vector products, can be reduced significantly. Instead of (2.5), the modified system

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

is solved instead, in which $\mathbf{M}^{-1}$ is chosen such that the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered than those of $\mathbf{H}$. As shown in [260], a block-Jacobi preconditioner is effective for the CPD: $\mathbf{M}$ is then a block diagonal approximation of $\mathbf{H}$:

$$\mathbf{M} = \begin{bmatrix} \mathbf{W}^{(1)} \otimes \mathbf{I}_I & & \\ & \mathbf{W}^{(2)} \otimes \mathbf{I}_J & \\ & & \mathbf{W}^{(3)} \otimes \mathbf{I}_K \end{bmatrix},$$

with $\mathbf{W}^{(1)} = (\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})$, $\mathbf{W}^{(2)} = (\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})$ and $\mathbf{W}^{(3)} = (\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})$. Applying the inverse of $\mathbf{M}$ is cheap as it only requires the inverse of typically small $R \times R$ matrices $\mathbf{W}^{(n)}$ thanks to identity (2.9). Note the similarities with the ALS results in subsection 2.3.3: the multiplication $\mathbf{M}^{-1}$ can be seen as a parallel ALS step in which all linear systems are solved simultaneously, rather than consecutively [260].

2.3.3 Alternating least squares

Another approach to solving (2.6) is to use alternating least squares (ALS), which is a popular approach thanks to its simplicity. By fixing all but one factor matrix, a linear least squares objective function is obtained. Hence,

each iteration, three LS problems are solved consecutively:

$$\mathbf{A}_k = \arg\min_{\mathbf{A}} \frac{1}{2} \left|\left|\mathbf{A}(\mathbf{C}_{k-1} \odot \mathbf{B}_{k-1})^{\mathrm{T}} - \mathbf{T}_{(1)}\right|\right|_{\mathrm{F}}^2,$$
$$\mathbf{B}_k = \arg\min_{\mathbf{B}} \frac{1}{2} \left|\left|\mathbf{B}(\mathbf{C}_{k-1} \odot \mathbf{A}_k)^{\mathrm{T}} - \mathbf{T}_{(2)}\right|\right|_{\mathrm{F}}^2,$$
$$\mathbf{C}_k = \arg\min_{\mathbf{C}} \frac{1}{2} \left|\left|\mathbf{C}(\mathbf{B}_k \odot \mathbf{A}_k)^{\mathrm{T}} - \mathbf{T}_{(3)}\right|\right|_{\mathrm{F}}^2.$$

(Hence the updated variables from the previous LS substep are used.) It can be verified that the solution of the first LS step is given by

$$\mathbf{A} = \mathbf{T}_{(1)} \left((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}\right)^{\dagger}.$$

(We drop the subscripts $k-1$ for simplicity of notation.) As the pseudoinverse of a tall matrix $\mathbf{X}$ with full column rank can be written as $\mathbf{X}^{\dagger} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$, the solution can be rewritten using (2.8) as

$$\begin{aligned} \mathbf{A} &= \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}) \left((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{B})\right)^{-1} \\ &= \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}) \left((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})\right)^{-1}, \end{aligned} \tag{2.18}$$

which is the classical ALS result. The matrix $\mathbf{W}^{(1)} = (\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})$ is an $R \times R$ matrix and can be inverted cheaply for low-rank problems.

It is fruitful to see the solution of the LS systems in the larger optimization framework, as illustrated in Figure 2.6. (We again drop the subscript $k-1$ for $\mathbf{B}$ and $\mathbf{C}$, and set $\mathbf{W} = \mathbf{W}^{(1)}$.) Each subproblem, a step $\mathbf{p}_k = \mathrm{vec}\,(\mathbf{P}_k)$ for updating $\mathbf{A}$ is computed from the system

$$(\mathbf{W} \otimes \mathbf{I}_I)\mathrm{vec}\,(\mathbf{P}_k) = -\mathrm{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right),$$

in which $\mathbf{R}_{(1)} = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}$ is the unfolded residual. The step $\mathbf{P}_k$ is found by inverting $(\mathbf{W} \otimes \mathbf{I}_I)$, which is exactly $\mathbf{H}_{\mathrm{AA}}$ from subsection 2.3.2, using identities (2.7) and (2.9). The variables are then updated as

$$\begin{aligned} \mathbf{A}_k &= \mathbf{A}_{k-1} + \alpha_k \mathbf{P}_k \\ &= \mathbf{A}_{k-1} - \alpha_k \left(\mathbf{A}_{k-1}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}\right)(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1} \\ &= \mathbf{A}_{k-1} - \alpha_k \mathbf{A}_{k-1}\mathbf{W}\mathbf{W}^{-1} + \alpha_k \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1} \\ &= (1 - \alpha_k)\mathbf{A}_{k-1} + \alpha_k \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1}. \end{aligned}$$

Setting $\alpha_k = 1$ gives the result in (2.18). However, to somewhat mitigate the slow convergence of ALS for ill-conditioned problems, exact or approximate line search methods selecting different values for α_k have been discussed in [45], [139], [228], [257].

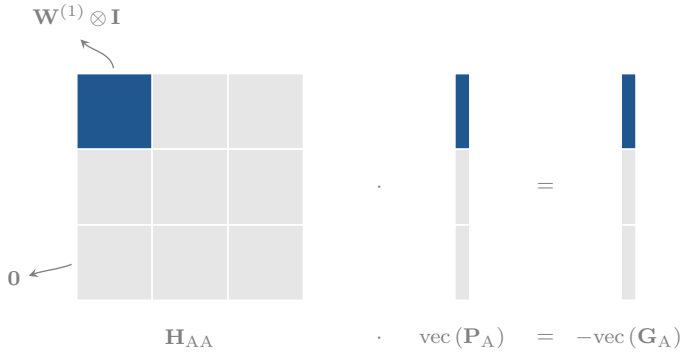


Figure 2.6: Every nth ALS subiteration, the system $\left(\mathbf{W}^{(n)} \otimes \mathbf{I}\right) \mathrm{vec}\,(\mathbf{P}_n) = -\mathrm{vec}\,(\mathbf{G}_n)$, with $n = 1, 2, 3$, is solved. *(We define $\mathbf{P}_1 = \mathbf{P}_A$, $\mathbf{P}_2 = \mathbf{P}_B$ and $\mathbf{P}_3 = \mathbf{P}_C$, and use a similar definiton for $\mathbf{G}_n$.) The subproblem for $n = 1$ is shown.*

2.3.4 More general objective functions

The Gauss–Newton algorithm is defined for the Euclidean distance, i.e., for the minimization of the least squares error. However, depending on the assumptions made for the data, other distance measures or divergences can be more appropriate. For example, in the case of count data, the Kullback–Leibler divergence can be more appropriate as its minimizer coincides with the minimizer of the negative log likelihood function for Poisson distributed data [62]. Algorithms alternating between variables and/or using multiplicative updates have been presented for generalized divergences including Tweedie and alpha and beta divergences; see, e.g., [62], [66], [136], [167], [225], [249], [310]. Similar to [237], we extend the ideas of GN to more general divergences such that a positive semidefinite Hessian approximation is used and the multilinear structure is exploited. Additional constraints such as nonnegativity can be handled using the techniques from section 2.4.

Table 2.1: The (element-wise) objective function f_i and its derivatives for some divergences.

	f_i	$\frac{\mathrm{d}f_i}{\mathrm{d}m_i}$	$\frac{\mathrm{d}^2 f_i}{\mathrm{d}m_i^2}$
Euclidean	$\frac{1}{2}(m_i - t_i)^2$	$m_i - t_i$	1
Kullback–Leibler	$m_i - t_i \log m_i + t_i \log t_i - t_i$	$\frac{m_i - t_i}{m_i}$	$\frac{t_i}{m_i^2}$
Itakura–Saito	$\frac{t_i}{m_i} - \log t_i + \log m_i - 1$	$\frac{m_i - t_i}{m_i^2}$	$\frac{2t_i - m_i}{m_i^3}$

The least squares objective function (2.1) can be rewritten as

$$\min_{\mathbf{z}} \sum_{i=1}^{N} f_i(m_i(\mathbf{z})) \qquad \text{with} \qquad f_i = \frac{1}{2}(m_i(\mathbf{z}) - t_i)^2,$$

with N the number of data points, which is equal to the number of tensor entries in this case, i.e., $N = IJK$. If we replace the Euclidean distance with another divergence, only the function f_i changes. Some examples are given in Table 2.1. As explained in subsection 2.2.2, a second-order Taylor series expansion $\tilde{f}(\mathbf{p})$ at $\mathbf{z}_k$ can be constructed as

$$f(\mathbf{z}) \approx \tilde{f}(\mathbf{p}) = f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}} \cdot \nabla_{\mathbf{z}} f(\mathbf{z}_k) + \frac{1}{2}\mathbf{p}^{\mathrm{T}} \cdot \nabla^2_{\mathbf{z}} f(\mathbf{z}_k) \cdot \mathbf{p}.$$

Using the chain rule, the gradient $\mathbf{g}$ is then given by

$$\mathbf{g} = \nabla_{\mathbf{z}} f = \sum_{i=1}^{N} \frac{\mathrm{d}f_i}{\mathrm{d}m_i} \nabla_{\mathbf{z}} m_i, \tag{2.19}$$

in which $\nabla_{\mathbf{z}} m_i$ is exactly the ith (transposed) row of the Jacobian matrix $\mathbf{J}$ as derived in subsection 2.3.2. Hence, when collecting all derivatives w.r.t. the model m_i in a vector $\mathbf{j}_{\mathrm{m}}$, the gradient is given by

$$\mathbf{g} = \mathbf{J}^{\mathrm{T}} \mathbf{j}_{\mathrm{m}}.$$

As shown in Table 2.1, the vector $\mathbf{j}_{\mathrm{m}}$ is simply the vectorized residual for least squares problems. Even though the expressions are more involved for other divergences, as can be seen in Table 2.1, the construction of $\mathbf{j}_m$ is the only change, which means efficient implementations for the MTKRPROD can be reused.

The Hessian is computed by taking the derivative of the gradient in (2.19) w.r.t. the variables $\mathbf{z}$, again using the chain rule, hence

$$\nabla^2_{\mathbf{z}} f = \sum_{i=1}^{N} \nabla_{\mathbf{z}} m_i \cdot \frac{\mathrm{d}^2 f_i}{\mathrm{d}m_i^2} \cdot (\nabla_{\mathbf{z}} m_i)^{\mathrm{T}} + \nabla^2_{\mathbf{z}} m_i \cdot \frac{\mathrm{d}f_i}{\mathrm{d}m_i}.$$

In the case of GN, the second term is assumed to be small near the optimum and neglecting this term still gives a good approximation of the Hessian. This is also a reasonable assumption for the other divergences: $\sum_{i=1}^{N} \nabla^2_{\mathbf{z}} m_i$ depends only on the model and is a sparse matrix as shown in [260], and $\frac{\mathrm{d}f_i}{\mathrm{d}m_i}$ becomes zero in the optimum, i.e., if $m_i = t_i$ as illustrated in Table 2.1. Therefore, the Hessian can be approximated as

$$\nabla^2_{\mathbf{z}} f \approx \mathbf{H} = \mathbf{J}^{\mathrm{T}} \mathbf{D} \mathbf{J},$$

in which $\mathbf{D}$ is an $IJK \times IJK$ diagonal matrix with the derivatives $\frac{\mathrm{d}^2 f_i}{\mathrm{d}m_i^2}$ as entries on the diagonal. Compared to the least squares problem, which has $\mathbf{D} = \mathbf{I}_{IJK}$, only the entries on the diagonal of $\mathbf{D}$, which are given in Table 2.1, are different. As $\mathbf{H}$ can again be a positive semidefinite approximation to the Hessian, favorable convergence conditions apply here as well, especially in combination with globalization strategies such as trust regions.

The multilinear structure of the CPD can again be exploited when computing the step direction from the system $\mathbf{Hp} = -\mathbf{g}$. The computation of the gradient requires three MTKRPROD operations using the tensor $\tilde{\mathcal{R}} = \text{unvec}\,(\mathbf{j}_\text{m})$ instead of the residual tensor $\mathcal{R}$. For example, instead of (2.15) we have

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

The expressions for (2.16) and (2.17) are similar. When using PCG, the approximate Hessian-vector products $\mathbf{y} = \mathbf{Hx}$ can be computed as

$$\begin{aligned}
\tilde{\mathcal{X}} &= \tilde{\mathcal{D}} * (\llbracket \mathbf{X}_\text{A}, \mathbf{B}, \mathbf{C} \rrbracket + \llbracket \mathbf{A}, \mathbf{X}_\text{B}, \mathbf{C} \rrbracket + \llbracket \mathbf{A}, \mathbf{B}, \mathbf{X}_\text{C} \rrbracket) \\
\mathbf{y}_\text{A} &= \text{vec}\left(\tilde{\mathbf{X}}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) \\
\mathbf{y}_\text{B} &= \text{vec}\left(\tilde{\mathbf{X}}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) \\
\mathbf{y}_\text{C} &= \text{vec}\left(\tilde{\mathbf{X}}_{(3)}(\mathbf{B} \odot \mathbf{A})\right),
\end{aligned}$$

in which $\tilde{\mathcal{D}} = \text{unvec}\,(\text{diag}(\mathbf{D}))$ and $\tilde{\mathcal{X}}$ is an auxiliary variable. To compute the sum of the CPDs, i.e., $\tilde{\mathcal{X}}$, the intermediate results can be reused. If $\tilde{\mathcal{D}}$ is sparse, the computational complexity can be reduced by only computing the entries corresponding to nonzero entries in $\tilde{\mathcal{D}}$ and by using specialized routines for the MTKRPROD of sparse tensors; see, e.g., [16], [251].

2.4 Constrained decompositions

Constraints can be used to incorporate prior knowledge: are the factor vectors smooth, nonnegative, linear combinations of elements in a dictionary, or do they have a Vandermonde structure? In this section, hard and soft constraints are discussed. In the former type, the constraint always holds, while constraints can be violated in the latter type and violations are penalized. Techniques to add parametric and projection-based bound, or *box*, constraints to the GN optimization problem are discussed first in subsections 2.4.1 and 2.4.2 as examples of hard constraints. Next, soft constraints are added through regularization in subsection 2.4.3. Finally, symmetry constraints are discussed as an example in subsection 2.4.4.

Example 8: A rank-2 tensor $\mathcal{T}$ with smooth factor vectors is constructed as follows. The first factor vector in the first mode $\mathbf{a}_1$ is a sum

of two exponentials. All other factor vectors are random polynomials with a maximal degree equal to four. (See Figure 2.7.) Then Gaussian, white, i.i.d. noise is added such that the SNR is -20 dB, which loosely means that the noise is ten times stronger than the signal. We exploit the prior knowledge that each factor vector (also $\mathbf{a}_1$) can be approximated by low-degree polynomials by imposing polynomial constraints (with a maximal degree of four). As the results in Figure 2.7 show, the rank-1 terms can be recovered thanks to the polynomial constraints, while the result using unconstrained CPD is heavily perturbed by noise. Albeit $\mathbf{a}_1$ is a sum of exponentials, it can be approximated well by a polynomial, as is clear from the figure.

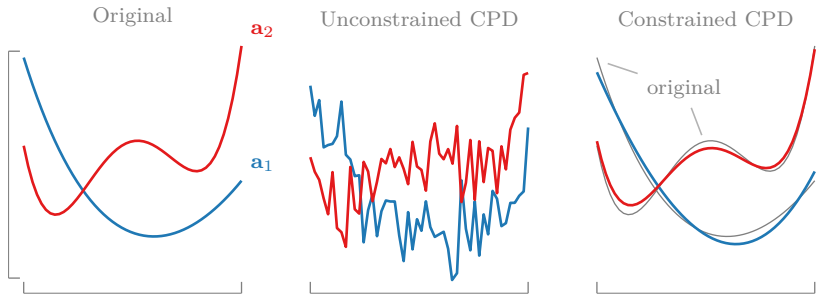


Figure 2.7: By imposing polynomial constraints, which we assume as prior knowledge, the smooth factor vectors can be recovered using a CPD of a noisy tensor. *Without constraints, the results are heavily perturbed by the noise. The plots show the factor vectors in the first mode. The SNR is* -20 dB.

2.4.1 Parametric constraints[3]

A large number of constraints imposed on factor matrices can be modeled as a function or *transformation* of some parameter vector $\mathbf{z} = \text{vec}(\mathbf{Z})$:

1. nonnegativity can be implemented by squaring values, i.e., $\mathbf{A} = \mathbf{Z} * \mathbf{Z}$ [232], or by taking the absolute value, i.e., $\mathbf{A} = |\mathbf{Z}|$;
2. a matrix with entries in a given interval can, for example, be modeled using sigmoidal constraints, e.g., $a_{ir} = z_{ir}/\sqrt{1 + z_{ir}^2}$ for $-1 \leq a_{ir} \leq 1$;
3. a matrix $\mathbf{A}$ with normalized columns can be implemented by defining each entry as $a_{ir} = z_{ir}/\sqrt{\sum_{k=1}^{I} z_{kr}^2}$;

[3]The idea of parametric constraints is given in [260]. This subsection provides more details and new illustrative examples compared to [260].

4. a matrix $\mathbf{A}$ in which each column $\mathbf{a}_r$ is a polynomial of degree d evaluated in the given points t_i, e.g., $a_{ir} = q_{0r} + q_{1r}t_i + q_{2r}t_i^2 + \ldots + q_{dr}t_i^d$, can be modeled as a chosen basis matrix $\mathbf{M}$ times a coefficient matrix $\mathbf{Q}$, i.e., $\mathbf{A} = \mathbf{MQ}$, with, e.g., for a monomial basis

$$a_{ir} = \underbrace{\begin{bmatrix} 1 & t_i & t_i^2 & \ldots & t_i^d \end{bmatrix}}_{\mathbf{M}(i,:)} \underbrace{\begin{bmatrix} q_{0r} & q_{1r} & q_{2r} & \ldots & q_{dr} \end{bmatrix}^{\mathrm{T}}}_{\mathbf{q}_r^{\mathrm{T}}};$$

5. the LL1 decomposition [77] or PARALIND [48] can be seen as a CPD with repeated factor vectors in the third mode, i.e., $\mathbf{C} = [\mathbf{z}_1, \ldots, \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_2, \ldots, \mathbf{z}_R, \ldots, \mathbf{z}_R]$ in which $\mathbf{z}_r$ is repeated L_r times, which can be written as the matrix-matrix product

$$\mathbf{C} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_R \end{bmatrix} \begin{bmatrix} \mathbf{1}_{L_1}^{\mathrm{T}} & & & \\ & \mathbf{1}_{L_2}^{\mathrm{T}} & & \\ & & \ddots & \\ & & & \mathbf{1}_{L_R}^{\mathrm{T}} \end{bmatrix};$$

6. a matrix with the entries from $\mathbf{z}$ on its diagonal, i.e., $\mathbf{A} = \operatorname{diag}(\mathbf{z})$;

7. a matrix $\mathbf{A}$ with constant values on the anti-diagonals, i.e., a Hankel matrix, depends on a single generating vector $\mathbf{z}$ as $a_{ir} = z_{i+r-1}$;

8. orthogonality can be imposed via Householder reflectors $\mathbf{z}_r \in \mathbb{R}^{I-r+1}$, $r = 1, \ldots, R$, as variables [281];

9. a matrix $\mathbf{A}$ in which the rth column is a Gaussian bell curve with mean m_r and variance σ_r^2, i.e., $a_{ir} = \exp\left(\frac{(t_i - m_r)^2}{2\sigma_r^2}\right)$, uses variables $\mathbf{z} = \begin{bmatrix} \mathbf{m}; \boldsymbol{\sigma} \end{bmatrix} \in \mathbb{R}^{2R}$ and a given point set $\mathbf{t}$.

These examples represent various types of constraints that often occur. The first two examples are element-wise constraints, the third is a column-wise constraint and the fourth and fifth are matrix-matrix product type constraints. The sixth and seventh example are placement type constraints as values from $\mathbf{z}$ are put at certain positions in a matrix. For placement type constraints some entries can be constants, e.g., zero as in the sixth example. The eight example represents the most general case. Finally, while example 9 actually is a column-wise constraint and can be implemented as a single

transformation, it can also be seen as chain of five simpler ones as follows

$$\begin{aligned}(\mathbf{Z}_1)_{ir} &= t_i - m_r \\ (\mathbf{z}_2)_r &= \sigma_r^2 \\ \mathbf{z}_3 &= 2\mathbf{z}_2 \\ (\mathbf{Z}_4)_{ir} &= (\mathbf{Z}_1)_{ir}/(\mathbf{z}_3)_r \\ (\mathbf{Z}_5)_{ir} &= \exp((\mathbf{Z}_4)_{ir}).\end{aligned}$$

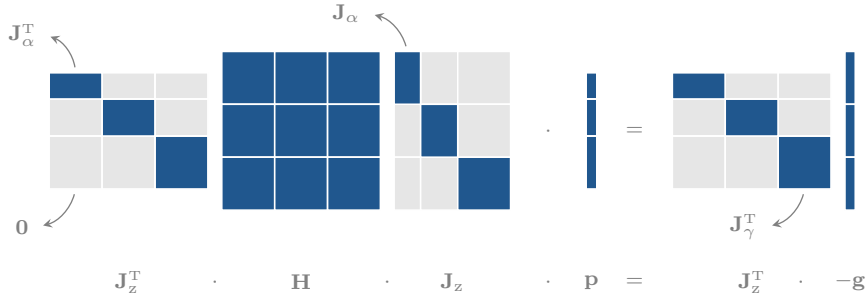


Figure 2.8: To impose parametric constraints on the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, which depend on the variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively, an additional block diagonal matrix $\mathbf{J}_\mathrm{z}$ is introduced in the system used to find $\mathbf{p}$, i.e., the update vector for the variables.

When constraints are imposed on factor matrices, the underlying variables are updated rather than the factor matrices, hence $\mathbf{z}$ and $\mathbf{p}$ in (2.2) and (2.3) now correspond to variables instead of factor matrices. The parametric constraints can be incorporated easily in the GN framework by exploiting the chain rule for derivation as shown in [262]: the derivative w.r.t. the underlying variables $\mathbf{z}$ is given by the derivative w.r.t. the factor matrix multiplied by the derivative of the factor matrix w.r.t. the variable $\mathbf{z}$. Mathematically, the system used to compute the step $\mathbf{p}$ becomes

$$\mathbf{J}_\mathrm{z}^\mathrm{T}\mathbf{H}\mathbf{J}_\mathrm{z}\mathbf{p} = -\mathbf{J}_\mathrm{z}^\mathrm{T}\mathbf{g}, \tag{2.20}$$

in which $\mathbf{J}_\mathrm{z}$ is a block-diagonal matrix having the Jacobians w.r.t. the underlying variables as blocks; see Figure 2.8 for a schematic overview. Hence, if $\mathbf{A}$ depends on $\boldsymbol{\alpha} \in \mathbb{R}^{N_a}$, $\mathbf{B}$ on $\boldsymbol{\beta} \in \mathbb{R}^{N_b}$ and $\mathbf{C}$ on $\boldsymbol{\gamma} \in \mathbb{R}^{N_b}$, then $\mathbf{z} = \left[\boldsymbol{\alpha}; \boldsymbol{\beta}; \boldsymbol{\gamma}\right]$

and $\mathbf{J}_z = \text{blkdiag}(\mathbf{J}_\alpha, \mathbf{J}_\beta, \mathbf{J}_\gamma)$, in which $\mathbf{J}_\alpha$ is the Jacobian of $\mathbf{A}$ w.r.t. $\boldsymbol{\alpha}$:

$$\mathbf{J}_\alpha = \begin{bmatrix} \frac{\partial a_{11}}{\partial \alpha_1} & \frac{\partial a_{11}}{\partial \alpha_2} & \cdots & \frac{\partial a_{11}}{\partial \alpha_{N_a}} \\ \frac{\partial a_{21}}{\partial \alpha_1} & \frac{\partial a_{21}}{\partial \alpha_2} & \cdots & \frac{\partial a_{21}}{\partial \alpha_{N_a}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial a_{IR}}{\partial \alpha_1} & \frac{\partial a_{IR}}{\partial \alpha_2} & \cdots & \frac{\partial a_{IR}}{\partial \alpha_{N_a}} \end{bmatrix}$$

and so on. The matrix $\mathbf{J}_\alpha$ depends only on the function used to transform the variables $\boldsymbol{\alpha}$ into the factor $\mathbf{A}$ and is therefore independent of the decomposition or divergence used. The structure of $\mathbf{J}_\alpha$ depends on the type of constraint, for example:

- for unconstrained factors $\mathbf{J}_\alpha = \mathbf{I}$,
- for element-wise transformations, $\mathbf{J}_\alpha$ is a diagonal matrix,
- for placement type transformations, $\mathbf{J}_\alpha$ is binary matrix such that $(\mathbf{J}_\alpha)_{ij} = 1$ if variable z_j is put in the ith position in $\mathbf{A}$, i.e., $\text{vec}\,(\mathbf{A})_i = z_j$,
- for column-wise transformations, $\mathbf{J}_\alpha$ is a block diagonal matrix,
- for matrix product type constraints[4], $\mathbf{J}_\alpha$ is a Kronecker product; see (2.10).

Example 9: Consider a Toeplitz type constraint, i.e., a matrix with constant diagonals, with variables $\boldsymbol{\alpha} = [\alpha_1; \alpha_2; \alpha_3]$ and a zero upper triangular part:

$$\mathbf{A} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & \alpha_1 & 0 \\ \alpha_3 & \alpha_2 & \alpha_1 \end{bmatrix}.$$

This is a placement type constraint with constants and its binary Jacobian therefore is given by

$$\mathbf{J}_\alpha^{\mathrm{T}} = \begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{array} \begin{array}{c} \begin{array}{ccccccccc} a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{32} & a_{13} & a_{23} & a_{33} \end{array} \\ \begin{bmatrix} 1 & & & & 1 & & & & 1 \\ & 1 & & & & 1 & & & \\ & & 1 & & & & & & \end{bmatrix} \end{array}.$$

[4] In some cases, the construction of the matrices in a matrix product type constraint can be avoid by exploiting the structured, resulting in more efficient implementations, e.g., through forward-adjoint oracles [91].

These structures can then be exploited when computing products with vectorized matrices: for the gradients we have

$$\mathbf{J}_{\mathrm{z}}^{\mathrm{T}}\mathbf{g} = \begin{bmatrix} \mathbf{J}_{\alpha}^{\mathrm{T}}\mathrm{vec}\left(\mathbf{G}_{\mathrm{A}}\right) \\ \mathbf{J}_{\beta}^{\mathrm{T}}\mathrm{vec}\left(\mathbf{G}_{\mathrm{B}}\right) \\ \mathbf{J}_{\gamma}^{\mathrm{T}}\mathrm{vec}\left(\mathbf{G}_{\mathrm{C}}\right) \end{bmatrix}. \tag{2.21}$$

To compute the Gramian-vector products required to solve system (2.20) using PCG, we use the following three steps:

$$\tilde{\mathbf{x}} = \mathbf{J}_{\mathrm{z}}\mathbf{x} \tag{2.22}$$
$$\tilde{\mathbf{y}} = \mathbf{H}\tilde{\mathbf{x}} \tag{2.23}$$
$$\mathbf{y} = \mathbf{J}_{\mathrm{z}}\tilde{\mathbf{y}}. \tag{2.24}$$

The vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ have the dimensions of the factor matrices, while $\mathbf{x}$ and $\mathbf{y}$ have the dimensions of the underlying variables. Hence, (2.22)–(2.24) can be seen as an expansion of the current CG iterate $\mathbf{x}$ to the factors, followed by a Gramian-vector product without constraints as discussed in subsection 2.3.2. Finally the factors are contracted again.

Example 10 (nonnegativity constraints): When implementing nonnegativity by squaring the variables $\boldsymbol{\alpha} = \mathrm{vec}\left(\mathbf{D}\right)$, i.e., $\mathbf{A} = \mathbf{D} * \mathbf{D}$, the Jacobian w.r.t. $\boldsymbol{\alpha}$ is given by $\mathbf{J}_{\alpha} = \mathrm{diag}(2\boldsymbol{\alpha})$. (The other factors are not transformed.) Hence, the gradient w.r.t. $\boldsymbol{\alpha}$ in (2.21) can be computed as

$$\mathbf{J}_{\alpha}^{\mathrm{T}}\mathrm{vec}\left(\mathbf{G}_{\mathrm{A}}\right) = \mathrm{vec}\left(2\mathbf{D} * \mathbf{G}_{\mathrm{A}}\right).$$

For the Gramian-vector product $\mathbf{y}_{\alpha} = \mathrm{vec}\left(\mathbf{Y}_{\alpha}\right)$, we compute (2.22)–(2.24) as

$$\begin{aligned} \tilde{\mathbf{X}}_{\mathrm{A}} &= 2\mathbf{D} * \mathbf{X}_{\alpha} \\ \mathrm{vec}\left(\tilde{\mathbf{Y}}_{\mathrm{A}}\right) &= \mathbf{H}_{\mathrm{AA}}\mathrm{vec}\left(\tilde{\mathbf{X}}_{\mathrm{A}}\right) + \mathbf{H}_{\mathrm{AB}}\mathrm{vec}\left(\mathbf{X}_{\mathrm{B}}\right) + \mathbf{H}_{\mathrm{AC}}\mathrm{vec}\left(\mathbf{X}_{\mathrm{C}}\right) \\ \mathbf{Y}_{\alpha} &= 2\mathbf{D} * \tilde{\mathbf{Y}}_{\mathrm{A}}. \end{aligned}$$

Example 11 (polynomial constraints): When implementing polynomial constraints with known basis matrix $\mathbf{M}$ and unknown coefficients $\mathbf{Q}$, hence $\boldsymbol{\alpha} = \mathrm{vec}\left(\mathbf{Q}\right)$ and $\mathbf{A} = \mathbf{M}\mathbf{Q}$, the Jacobian w.r.t. $\boldsymbol{\alpha}$ is given by $\mathbf{J}_{\alpha} = \mathbf{I}_R \otimes \mathbf{M}$. (The other factors are not transformed.) Hence, the gradient w.r.t. $\boldsymbol{\alpha}$ in (2.21) can be computed using (2.10) as

$$\mathbf{J}_{\alpha}^{\mathrm{T}}\mathrm{vec}\left(\mathbf{G}_{\mathrm{A}}\right) = \mathrm{vec}\left(\mathbf{M}^{\mathrm{T}}\mathbf{G}_{\mathrm{A}}\right).$$

For the Gramian-vector product $\mathbf{y}_\alpha = \text{vec}\,(\mathbf{Y}_\alpha)$, we compute (2.22)–(2.24) as

$$\begin{aligned}\tilde{\mathbf{X}}_\text{A} &= \mathbf{M}\mathbf{X}_\alpha \\ \text{vec}\,(\tilde{\mathbf{Y}}_\text{A}) &= \mathbf{H}_\text{AA}\text{vec}\,(\tilde{\mathbf{X}}_\text{A}) + \mathbf{H}_\text{AB}\text{vec}\,(\mathbf{X}_\text{B}) + \mathbf{H}_\text{AC}\text{vec}\,(\mathbf{X}_\text{C}) \\ \mathbf{Y}_\alpha &= \mathbf{M}^\text{T}\tilde{\mathbf{Y}}_\text{A}.\end{aligned}$$

Example 12 (chaining constraints): If a factor matrix $\mathbf{A}$ is the result of chaining multiple transformations (see, e.g., the ninth example in the beginning of subsection 2.4.1), the Jacobian $\mathbf{J}_\alpha$ is the product of the Jacobians for each transformation, which follows from the chain rule. Consider a polynomial constraint with basis matrix $\mathbf{M}$ and nonnegative coefficients, implemented by squaring variables:

$$\mathbf{A} = \mathbf{M}\mathbf{Q} \qquad \text{with} \quad \mathbf{Q} = \mathbf{D} * \mathbf{D}.$$

The Jacobian $\mathbf{J}_\alpha$ of $\text{vec}\,(\mathbf{A})$ w.r.t. $\boldsymbol{\alpha} = \text{vec}\,(\mathbf{D})$ is then given by

$$\mathbf{J}_\alpha = \mathbf{J}^\text{A}_\text{Q}\mathbf{J}^\text{Q}_\text{D},$$

in which $\mathbf{J}^\text{A}_\text{Q} = \mathbf{I}_R \otimes \mathbf{M}$ is the Jacobian of $\text{vec}\,(\mathbf{A})$ w.r.t. $\text{vec}\,(\mathbf{Q})$ and $\mathbf{J}^\text{Q}_\text{D} = \text{diag}(2\boldsymbol{\alpha})$ is the Jacobian of $\text{vec}\,(\mathbf{Q})$ w.r.t. $\boldsymbol{\alpha} = \text{vec}\,(\mathbf{D})$; see examples 10 and 11. Hence,

$$\mathbf{J}_\alpha = (\mathbf{I}_R \otimes \mathbf{M})\text{diag}(2\boldsymbol{\alpha}).$$

Therefore, the gradient w.r.t. $\boldsymbol{\alpha}$ in (2.21) can be computed as

$$\mathbf{J}_\alpha^\text{T}\text{vec}\,(\mathbf{G}_\text{A}) = \text{vec}\,(\mathbf{M}^\text{T}(2\mathbf{D} * \mathbf{G}_\text{A}))\,.$$

The Gramian-vector products can be derived in a similar way as in the previous examples.

2.4.2 Projection-based bound constraints

It is often useful to constrain variables between certain bounds. For example, when using the KL divergence in subsection 2.3.4, the model m_i should be nonnegative because of the logarithm, which can be achieved using nonnegativity constraints on the factor matrices. Instead of squaring the variables as in example 10, bound or box constraints can be used as an alternative. Mathematically, the problem

$$\min_\mathbf{z} \frac{1}{2}\,||[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}||^2_\text{F} \quad \text{subject to} \quad \mathbf{l} \leq \mathbf{z} \leq \mathbf{u}$$

is solved, in which $\leq$ hold elementwise and $\mathbf{l} \leq \mathbf{u}$. If a certain variable z_i is unbounded below or above, the lower bound $l_i = -\infty$ or the upper bound $u_i = +\infty$ is used, respectively. To enforce these constraints, an active set method can be used. The active set $\mathcal{A}$ is defined as the set of indices corresponding to the variables for which the current estimates $\mathbf{z}$ are at the bounds, i.e.,

$$\mathcal{A} = \{i \mid l_i = z_i \text{ or } z_i = u_i\}, \tag{2.25}$$

and its complement is the inactive set $\mathcal{I}$. While determining active sets is not trivial in general, simple bound constraints do allow an easy method to be used [160]. At the beginning of each iteration, the active set is determined using (2.25). Suppose that the variables $\mathbf{z}$ are sorted such that $\mathbf{z} = \left[\mathbf{z}_\mathcal{I}; \mathbf{z}_\mathcal{A}\right]$ and that $\mathbf{p}$ and $\mathbf{g}$ are partitioned accordingly, then, instead of the system (2.5), the altered system

$$\left(\begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{|\mathcal{A}|} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_{|\mathcal{I}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{H} \begin{bmatrix} \mathbf{I}_{|\mathcal{I}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \mathbf{p} = -\mathbf{g}$$

is solved, which can be simplified as

$$\mathbf{H} \begin{bmatrix} \mathbf{p}_\mathcal{I} \\ \mathbf{0} \end{bmatrix} = - \begin{bmatrix} \mathbf{g}_\mathcal{I} \\ \mathbf{0} \end{bmatrix}$$
$$\mathbf{p}_\mathcal{A} = -\mathbf{g}_\mathcal{A}.$$

Therefore, the implementations for efficient Gramian-vector products can be reused. The only change is that every CG iteration the entries of $\mathbf{x}$ in the active set are set to zero, i.e., let $\tilde{\mathbf{x}} = \left[\mathbf{x}_\mathcal{I}; \mathbf{0}\right]$, after which the standard Gramian-vector product $\tilde{\mathbf{y}} = \mathbf{H}\tilde{\mathbf{x}}$ is computed as in subsection 2.3.2. The result $\mathbf{y}$ is finally constructed by concatenating the entries corresponding to the inactive set from $\tilde{\mathbf{y}}$ and the entries corresponding to the active set from $\mathbf{x}$, hence, $\mathbf{y} = \left[\tilde{\mathbf{y}}_\mathcal{I}; \mathbf{x}_\mathcal{A}\right]$. When updating the variables, instead of the computed $\mathbf{p}$, a projected step $\tilde{\mathbf{p}}$ is used to ensure that the constraints are not violated:

$$\tilde{p}_i = \begin{cases} l_i - z_i & \text{if } z_i + p_i \leq l_i \\ u_i - z_i & \text{if } z_i + p_i \geq u_i \\ p_i & \text{otherwise} \end{cases}.$$

Example 13 (nonnegative CPD): Consider a random rank-10 tensor $\mathcal{T}$ of size $50 \times 50 \times 50$, constructed using factor matrices with entries drawn from the uniform distribution $\mathcal{U}(0, 1)$ such that all factor elements are nonnegative. The nonnegative CPD of the noiseless tensor $\mathcal{T}$

is computed starting from random factor matrices with entries drawn from the same uniform distribution, except for the first factor matrix, which is initialized to zero. Two methods are compared: an active set method with lower bound $\mathbf{l} = \mathbf{0}$ and no upper bound ($\mathbf{u} = +\infty$), and a method with parametric constraints, i.e., by squaring variables (see example 10). The former method is implemented as `cpd_nls` with the `nlsb_gndl` solver, while the latter uses the SDF framework through `sdf_nls` and `struct_nonneg` [305]. While the active set method recovers the original factor matrices up to machine precision, `sdf_nls` fails to perform a single step, as the gradient $\mathbf{g}$ is exactly zero for this method which means this initialization is a local minimum for the problem with parametric constraints. The inability to change variables that are initialized at zero or that become zero during the optimization process, is a common problem with nonnegativity constraints implemented using squared variables.

2.4.3 Regularization and soft constraints

Regularization is often used to prevent overfitting or to incorporate prior knowledge into the optimization problem. Factor matrices or variables can be regularized by adding terms to the objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \left|\left|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}\right|\right|^2 + \lambda_{\mathrm{A}} h_{\mathrm{A}}(\mathbf{A}) + \lambda_{\mathrm{B}} h_{\mathrm{B}}(\mathbf{B}) + \lambda_{\mathrm{C}} h_{\mathrm{C}}(\mathbf{C}),$$

in which λ_{A}, λ_{B} and λ_{C} are hyperparameters which have to be chosen by the user or automatically, e.g., through cross validation. Examples of possible functions h_{A}, h_{B} and h_{C} are

- L_2 norm regularization: $h(\mathbf{z}) = \frac{1}{2} \left|\left|\mathbf{z}\right|\right|_2^2$,
- L_1 norm regularization: $h(\mathbf{z}) = \left|\left|\mathbf{z}\right|\right|_1$, and
- L_0 pseudonorm regularization: $h(\mathbf{z}) = \left|\left|\mathbf{z}\right|\right|_0$.

The latter two norms are often used to achieve a sparser solution, as (small) nonzeros are penalized more heavily compared to the L_2 norm. (Relaxations of the latter two norms are used in practice to avoid problems with discontinuities in the derivatives.) As the derivative of a sum is the sum of the derivatives, adding regularization results in an additional term for the gradient and the Hessian approximation, as illustrated in Figure 2.9. Concretely, let $\tilde{\mathbf{g}}$ and $\tilde{\mathbf{H}}$ be the gradient and Gramian for the least squares term $\frac{1}{2} \left|\left|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}\right|\right|_{\mathrm{F}}^2$. Then, in the case only $\mathbf{A}$ is regularized, the gradient

and Hessian approximation are altered as

$$\mathbf{g}_{\mathrm{A}} = \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \nabla_{\mathrm{vec}(\mathbf{A})} h$$
$$\mathbf{H}_{\mathrm{AA}} = \tilde{\mathbf{H}}_{\mathrm{AA}} + \lambda \nabla^2_{\mathrm{vec}(\mathbf{A})} h,$$

while $\mathbf{g}_{\mathrm{B}}$ and $\mathbf{g}_{\mathrm{C}}$, and all other blocks of the Gramian remain unchanged. For example, for L_2 regularization this becomes

$$\mathbf{g}_{\mathrm{A}} = \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \mathrm{vec}\,(\mathbf{A})$$
$$\mathbf{H}_{\mathrm{AA}} = \tilde{\mathbf{H}}_{\mathrm{AA}} + \lambda \mathbf{I}.$$

Similar terms can be added in the case other factors are regularized as well. Instead of the exact Hessian $\nabla^2_{\mathrm{vec}(\mathbf{A})} h$, an approximation can be used; see subsection 2.2.2.

Using regularization, soft constraints can be imposed. In contrast to the parameter and projection-based constraints, violations of the constraint are allowed, but penalized. The penalty is controlled by the hyperparameter λ and the regularization function h.

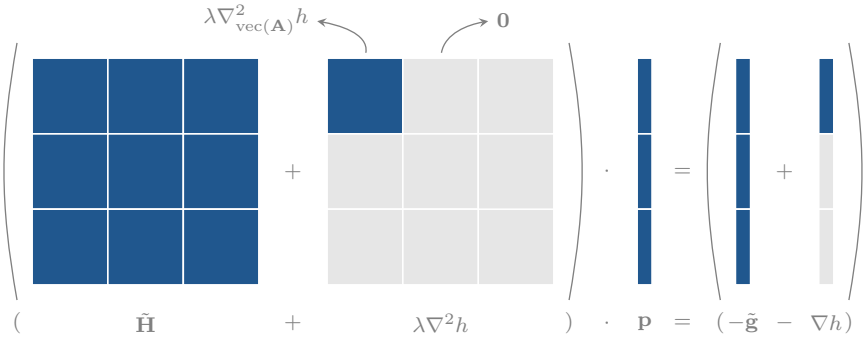


Figure 2.9: When adding regularization or implementing soft constraints, a block is added to the Hessian approximation and to the gradient.

Example 14 (soft orthogonality constraints): Instead of using Householder reflectors as parameters in order to enforce orthogonality, a regularization term can be added to impose orthogonality as a soft constraint:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \left|\left|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}\right|\right|^2_{\mathrm{F}} + \frac{\lambda}{4} \left|\left|\mathbf{A}^{\mathrm{T}}\mathbf{A} - \mathbf{I}_R\right|\right|^2_{\mathrm{F}}.$$

The additional contributions to the gradient and Gramian can be de-

rived easily by noting that $\mathbf{A}^{\mathrm{T}}\mathbf{A} = [\![\mathbf{A}^{\mathrm{T}}, \mathbf{A}^{\mathrm{T}}]\!]$ is a symmetric PD (see subsection 2.4.4) and that $\partial(\mathbf{A}^{\mathrm{T}}) = (\partial\mathbf{A})^{\mathrm{T}}$. Therefore, the gradient w.r.t. $\mathbf{A}$ is given by

$$\mathbf{g}_{\mathrm{A}} = \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \mathrm{vec}\left(\mathbf{A}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A} - \mathbf{I}\right)\right).$$

To compute the Gramian-vector product, we find after some simplifications that

$$\mathbf{y}_{\mathrm{A}} = \tilde{\mathbf{y}}_{\mathrm{A}} + \lambda \mathrm{vec}\left(\mathbf{A}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}_{\mathrm{A}} + \mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}\right)\right).$$

2.4.4 Symmetry[5]

In the previous subsections, we (implicitly) assumed that each factor matrix is a function of a different set of variables which results in a block-diagonal matrix $\mathbf{J}_{\mathrm{z}}$. Variables can be shared across different modes, however. An extreme example is the symmetric CPD of a tensor, e.g., $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{A}]\!]$. Incorporating symmetry in the GN framework amounts to summing blocks of $\mathbf{H}$ and $\mathbf{g}$ as shown in Figure 2.10. As many blocks are identical, only some need to be computed. Note that the contributions to the gradient are only identical if the tensor $\mathcal{T}$ is symmetric as well. The idea of summing the proper blocks can be extended easily to cases in which multiple factors share the same variables by summing the blocks corresponding to the variables rather than the blocks corresponding to the factors [306].

Example 15: Consider a tensor $\mathcal{T}$ that is approximated using a CPD with symmetry constraints in the first two modes, i.e., $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{A}, \mathbf{C}]\!]$; see Figure 2.10. In this case, the gradient is given by

$$\begin{aligned}\mathbf{g}_{\mathrm{A}} &= \mathbf{R}_{(1)}\left(\mathbf{C} \odot \mathbf{A}\right) + \mathbf{R}_{(2)}\left(\mathbf{C} \odot \mathbf{A}\right)\\ \mathbf{g}_{\mathrm{C}} &= \mathbf{R}_{(3)}\left(\mathbf{A} \odot \mathbf{A}\right).\end{aligned}$$

Only if $\mathcal{T}$ is also symmetric in the first two modes, $\mathbf{R}_{(1)} = \mathbf{R}_{(2)}$ and $\mathbf{g}_{\mathrm{A}}$ can be computed as $\mathbf{g}_{\mathrm{A}} = 2\mathbf{R}_{(1)}\left(\mathbf{C} \odot \mathbf{A}\right)$. To compute the Gramian-vector products $\mathbf{y} = \mathbf{H}\mathbf{x}$, with $\mathbf{y} = \left[\mathbf{y}_{\mathrm{A}}; \mathbf{y}_{\mathrm{C}}\right]$ and $\mathbf{x} = \left[\mathbf{x}_{\mathrm{A}}; \mathbf{x}_{\mathrm{C}}\right]$, we can use

$$\begin{aligned}\mathbf{y}_{\mathrm{A}} &= 2\mathbf{X}_{\mathrm{A}}\left(\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right) * \left(\mathbf{C}^{\mathrm{T}}\mathbf{C}\right)\right) + 2\mathbf{A}\left(\left(\mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}\right) * \left(\mathbf{C}^{\mathrm{T}}\mathbf{C}\right)\right)\\ &\quad + 2\mathbf{A}\left(\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right) * \left(\mathbf{X}_{\mathrm{C}}^{\mathrm{T}}\mathbf{C}\right)\right)\\ \mathbf{y}_{\mathrm{C}} &= 2\mathbf{C}\left(\left(\mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}\right) * \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)\right) + 2\mathbf{X}_{\mathrm{C}}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right).\end{aligned}$$

[5] The symmetry results are part of the SDF framework from [260] and have been extended in [306]. New examples are presented in this subsection.

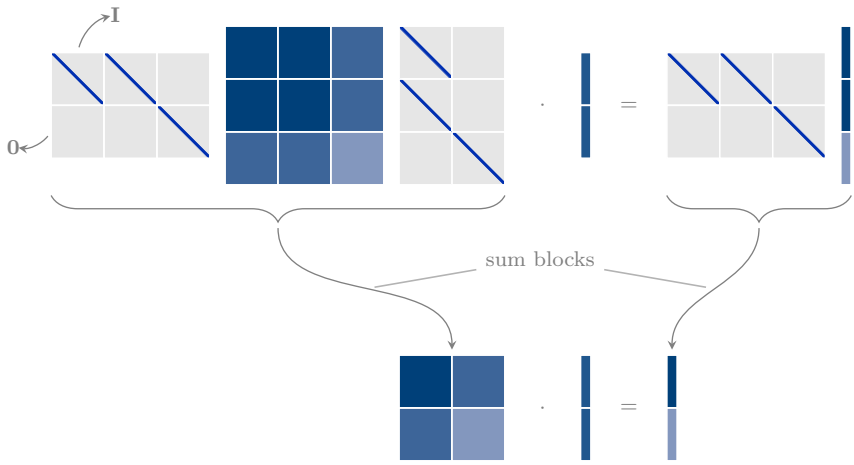


Figure 2.10: For symmetric tensors, e.g., $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{C}]\!]$, the system (2.5) can be solved more cheaply by summing blocks corresponding to the same variable. *As many — not all — blocks in the same color are identical, the sums can be simplied; see example 15.*

2.5 Coupled decompositions

A decomposition of a third-order tensor can be seen as the joint decomposition of its matrix slices. More generally, one may consider the joint decomposition of matrices and/or tensors (possibly of different order) that (do not necessarily have the same dimensions and) only share part of their structure. For example, only a few columns of a factor matrix are shared, or a factor in one dataset is a function of a factor in another dataset. Such joint analysis is important for applications like data fusion, e.g., in multimodal data analysis in biomedical applications. For example, in [288] a tensor resulting from EEG measurements and a matrix resulting from fMRI are coupled: the time factor for the fMRI data is the convolution of the time factor in the EEG data with the hemodynamic response function. An example from recommender systems is given in example 16.

> **Example 16 (GPS):** Predicting whether a person has attended or will attend an activity, e.g., buy dinner or go shopping, at a certain location, e.g., the mall or a local grocery store, is an important part of recommender systems. Using the GPS dataset [316], a tensor with modes user × location × activity can be constructed in which entries can be missing; see Figure 2.11. To predict the unknown entries, a CPD of the incomplete tensor can be computed. To improve the prediction quality, additional information can be used such as features for each

location, the relation between users and the similarity between activities etc. [316]. This extra information is given as a set of matrices, which are factorized jointly with the tensor by coupling the factor matrices corresponding to the same modes [88], [262]. When a new user is added, no information about visits to a location for a certain activity is yet available, hence an entire slice of the tensor is missing. However, because of (known) relations with other users or shared interests, it is still possible to make predictions using the coupled decompositions. See [88], [262] for more information.

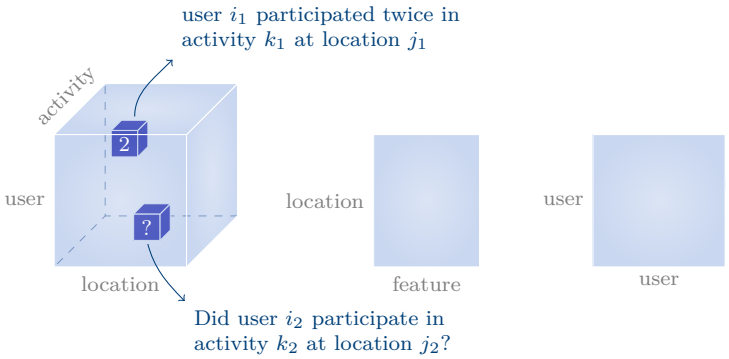


Figure 2.11: Did or will a user attend a certain activity at a certain location? By augmenting the GPS data tensor with information such as features for each location, relations between users and whether a user has been at a certain location, the unknown entries in the tensor can be predicted more accurately.

While the obvious use of coupled decompositions is data fusion, they can also be employed in a divide-and-conquer type of approach to data analysis. The coupled decomposition acts as a sort of compound eye: each facet sees its own simple part of the data, and the coupling creates the overall view. Example 17 is a very basic illustration of a combination of two sampling rates, yielding a higher accuracy than the individual sampling rates. Multirate techniques show promise for big data applications in which classical Nyquist sampling is not feasible (the Nyquist sampling rate is twice the signal bandwidth, which may result in excessive amounts of data). In [268], [269] coupled decompositions are used for multirate harmonic retrieval (each of the decompositions takes care of one sampling rate). In [267] coupled decompositions yield algebraic uniqueness conditions and linear algebra-based algorithms for tensor completion (each of the decompositions takes care of one fully observed subtensor). Noteworthy is also a connection between convolutive extensions of CPD and "instantaneous" coupled CPD [272].

A key advantage of coupling datasets in a signal separation context is that the conditions for uniqueness are usually very mild. While a matrix decom-

position is not unique without requiring additional stringent and sometimes unnatural constraints such as orthogonality, the CPD, which can be seen as the joint factorization of matrix slices, is essentially unique under relatively mild conditions [96]–[98], [179]. When tensors and matrices are factorized jointly, the conditions can be relaxed further, and a decomposition may be recovered uniquely even if none of the decompositions of the tensors/matrices individually is unique [266], [271]. A special case is the coupled matrix tensor factorization (CMTF) for which uniqueness conditions are derived in [266]. In the case only a few columns of a factor are shared between the matrix and tensor, i.e., in the case of partial coupling, [81] discusses the remaining indeterminacies in the matrix decomposition.

While the focus in the remainder of this section lies on coupling tensors and matrices that have a CPD structure, many other decompositions can be used. The structured data fusion (SDF) framework [262] discusses GN type algorithms to couple tensors factorized using CPDs, MLSVDs or BTDs. Other examples can be found in [63], [65], [310].

2.5.1 Exact coupling[6]

Consider two tensors $\mathcal{T}_1$ and $\mathcal{T}_2$ which are both approximated by a rank-R CPD. To jointly factorize these tensors, the objective function

$$\min_{\mathbf{z}} \frac{\omega_1}{2} \left|\left|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}_1\right|\right|_{\mathrm{F}}^2 + \frac{\omega_2}{2} \left|\left|[\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] - \mathcal{T}_2\right|\right|_{\mathrm{F}}^2 \tag{2.26}$$

can be used with $\mathbf{z} = \left[\mathrm{vec}\,(\mathbf{A})\,; \mathrm{vec}\,(\mathbf{B})\,; \mathrm{vec}\,(\mathbf{C})\,; \mathrm{vec}\,(\mathbf{D})\,; \mathrm{vec}\,(\mathbf{E})\,; \mathrm{vec}\,(\mathbf{F})\right]$. The weights ω_1 and ω_2 are hyperparameters and have to be chosen by the user or, e.g., through cross validation. In (2.26), the decompositions are uncoupled. The two decompositions are coupled exactly if one or more factors are (partially) shared, or if these factors depend on the same underlying variables. We discuss a few types of exact coupling:

- coupling factor matrices, e.g., $\mathbf{A} = \mathbf{D}$;
- partial coupling, e.g., $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3\end{bmatrix}$ and $\mathbf{D} = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{d}_3\end{bmatrix}$;
- coupling through variables $\mathbf{A} = h_1(\boldsymbol{\alpha})$ and $\mathbf{D} = h_2(\boldsymbol{\alpha})$.

For these three cases, the equality constraints can be imposed by substitution into the objective function. For example, if $\mathbf{A} = \mathbf{D}$, (2.26) becomes

$$\min_{\mathbf{z}} \frac{\omega_1}{2} \left|\left|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}_1\right|\right|_{\mathrm{F}}^2 + \frac{\omega_2}{2} \left|\left|[\![\mathbf{A}, \mathbf{E}, \mathbf{F}]\!] - \mathcal{T}_2\right|\right|_{\mathrm{F}}^2 .$$

[6] The coupling results are part of the SDF framework from [260] and have been extended in [306]. New examples are presented in this subsection.

Hence, by eliminating $\mathbf{D}$, the variables are

$$\mathbf{z} = \left[\text{vec}\left(\mathbf{A}\right); \text{vec}\left(\mathbf{B}\right); \text{vec}\left(\mathbf{C}\right); \text{vec}\left(\mathbf{E}\right); \text{vec}\left(\mathbf{F}\right)\right].$$

Then, because derivation is a linear operator, the system used to solve for $\mathbf{p}$ is simply

$$\left(\omega_1\mathbf{H}_1 + \omega_2\mathbf{H}_2\right)\mathbf{p} = -\left(\omega_1\mathbf{g}_1 + \omega_2\mathbf{g}_2\right)$$

in which $\mathbf{g}_i$ and $\mathbf{H}_i$ are the gradient and Gramian, respectively, of the ith term w.r.t. all variables [262], [306]. Both $\mathbf{g}_i$ and $\mathbf{H}_i$ contain zero blocks for factor matrices or variables not used in the factorization for the ith dataset, as shown in Figure 2.12. As explained in section 2.4, the gradient and the Gramian-vector products can be adapted easily. Note that for computational efficiency, $\mathbf{y} = \mathbf{H}\mathbf{x}$ is computed as $\mathbf{y} = (\omega_1\mathbf{H}_1\mathbf{x}) + (\omega_2\mathbf{H}_2\mathbf{x})$ rather than summing the blocks from $\mathbf{H}_1$ and $\mathbf{H}_2$ first [306].

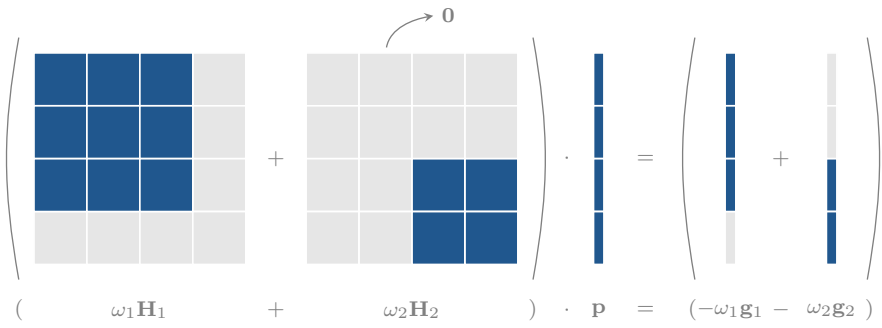


Figure 2.12: In the case of coupled tensor matrix factorization in which $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and $\mathbf{M} \approx \mathbf{C}\mathbf{D}^{\mathrm{T}}$, the system (2.5) is simply the sum of the two systems [306].

Example 17: Consider a function $h(x, y)$ which is sampled on two equidistant grids on $[-1, 1]$: grid 1 and grid 2 have 31 and 49 points in each dimension, respectively. Let $\mathbf{H}_1 \in \mathbb{R}^{31\times 31}$ and $\mathbf{H}_2 \in \mathbb{R}^{49\times 49}$ be two noisy measurements of $h(x, y)$ on these grids such that the SNR is 20 dB. The goal is to approximate $h(x, y)$ as

$$h(x, y) = \sum_{r=1}^{3} a_r(x)b_r(y)$$

with $a_r(x)$ and $b_r(y)$ low degree polynomials. As explained in section 2.4, this is a matrix product type constraint, hence $\mathbf{H}_1$ and $\mathbf{H}_2$

are factorized as

$$\mathbf{H}_1 \approx \mathbf{A}\mathbf{B}^{\mathrm{T}} \qquad \mathbf{H}_2 \approx \mathbf{C}\mathbf{D}^{\mathrm{T}}$$

with

$$\mathbf{A} = \mathbf{M}_1\mathbf{Q}_1 \qquad \mathbf{B} = \mathbf{M}_1\mathbf{Q}_2 \tag{2.27}$$
$$\mathbf{C} = \mathbf{M}_2\mathbf{Q}_1 \qquad \mathbf{D} = \mathbf{M}_2\mathbf{Q}_2 \tag{2.28}$$

in which $\mathbf{M}_1$ and $\mathbf{M}_2$ are, in this example, Legendre bases evaluated in the 31 and 49 points corresponding to the first and second grid, respectively. Although $\mathbf{H}_1$ and $\mathbf{H}_2$ are sampled on different grids and have a different size, the underlying variables, i.e., the coefficients $\mathbf{Q}_1$ and $\mathbf{Q}_2$, are shared, hence the following coupled problem can be solved

$$\min_{\mathbf{Q}_1,\mathbf{Q}_2} \frac{\omega_1}{2\cdot\Omega}\left|\left|\mathbf{H}_1 - \mathbf{A}\mathbf{B}^{\mathrm{T}}\right|\right|_{\mathrm{F}}^2 + \frac{\omega_2}{2\cdot\Omega}\left|\left|\mathbf{H}_2 - \mathbf{C}\mathbf{D}^{\mathrm{T}}\right|\right|_{\mathrm{F}}^2 \tag{2.29}$$
subject to (2.27) and (2.28).

(The factor $\Omega = \omega_1 + \omega_2$ is a normalization factor, used to avoid numerical problems with large weights.) The constraints are again eliminated by substituting (2.27) and (2.28) in the objective function (2.29) and the coupled problem is solved using `sdf_nls` [305]. To measure the performance a third grid is used with 100 equidistant points in each direction. Let $\mathbf{H}_3$ be a noiseless measurement on this third grid, then the relative validation error

$$E = \frac{\left|\left|\mathbf{H}_3 - \hat{\mathbf{H}}_3\right|\right|_{\mathrm{F}}}{\left|\left|\mathbf{H}_3\right|\right|_{\mathrm{F}}}$$

is used, in which $\hat{\mathbf{H}}_3$ is the reconstruction using the computed coefficients $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Figure 2.13 shows that the validation error E can be improved by using information from both measurements for a good choice of ω_1/ω_2.

2.5.2 Approximate coupling

Similar to soft constraints, two tensors can be coupled approximately by adding a regularization term. For example, if $\mathbf{A} \approx \mathbf{D}$, the objective function becomes

$$\min_{\mathbf{z}} \frac{\omega_1}{2}\left|\left|[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}_1\right|\right|_{\mathrm{F}}^2 + \frac{\omega_2}{2}\left|\left|[\![\mathbf{D},\mathbf{E},\mathbf{F}]\!] - \mathcal{T}_2\right|\right|_{\mathrm{F}}^2 + \frac{\lambda}{2}\left|\left|\mathbf{A}-\mathbf{D}\right|\right|_{\mathrm{F}}^2.$$

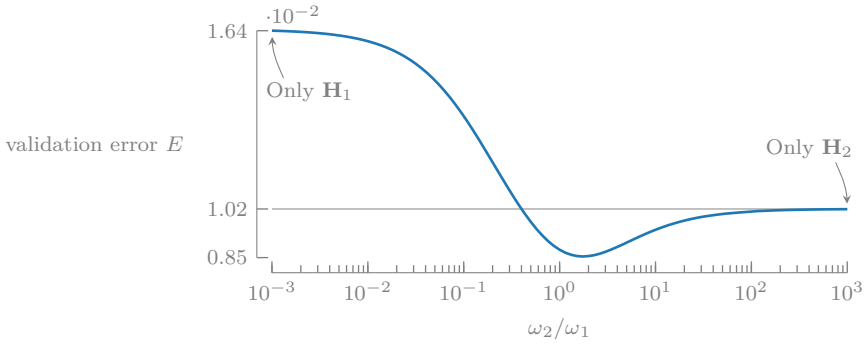


Figure 2.13: Over a certain range of ratios ω_1/ω_2, the validation error E is reduced when jointly factorizing both measurements $\mathbf{H}_1$ and $\mathbf{H}_2$ of $h(x, y)$, compared to using only one measurement.

Implementing this constraint can be done similarly to the approach in subsection 2.4.3. Note that there is an additional hyperparameter λ to be tuned. Other distance measures than the Frobenius norm can be used as well [49]. Soft coupling can also be imposed on the variables.

Example 18: The advanced coupled matrix tensor factorization (ACMTF) model [7] is an example of soft constraints. A tensor $\mathcal{T}$ and a matrix $\mathbf{M}$ share some factor vectors in the third and first mode, respectively, which can be modeled as

$$\mathcal{T} = \sum_{r=1}^{R} \sigma_r \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \qquad \mathbf{M} = \sum_{r=1}^{R} \tau_r \mathbf{c}_r \otimes \mathbf{d}_r$$

in which σ_r is zero if the rth rank-1 term is not present in the decomposition of $\mathcal{T}$ and similarly for τ_r. Hence, if both $\sigma_r \neq 0$ and $\tau_r \neq 0$, the rth column of factor matrix $\mathbf{C}$ is shared. (The norms of the factor vectors are assumed to be equal to one.) The idea in ACMTF is that the shared columns of $\mathbf{C}$ are determined automatically by solving the following optimization problem [7]:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \frac{\omega_1}{2} \left|\left| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{\sigma}^{\mathrm{T}}]\!] - \mathcal{T} \right|\right|_{\mathrm{F}}^2 + \frac{\omega_2}{2} \left|\left| [\![\mathbf{C}, \mathbf{D}, \boldsymbol{\tau}^{\mathrm{T}}]\!] - \mathbf{M} \right|\right|_{\mathrm{F}}^2 \\ & + \lambda \left(||\boldsymbol{\sigma}||_1 + ||\boldsymbol{\tau}||_1 \right) \\ \text{subject to} \quad & ||\mathbf{a}_r||_{\mathrm{F}} = ||\mathbf{b}_r||_{\mathrm{F}} = ||\mathbf{c}_r||_{\mathrm{F}} = ||\mathbf{d}_r||_{\mathrm{F}} = 1, \quad r = 1, \ldots, R. \end{aligned}$$

The L_1 norm is used to sparsify $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$. The normalization of the

factor vectors can be implemented using parametric constraints (subsection 2.4.1) or using soft constraints [7]. Note that these normalization constraints can always be fully satisfied.

2.6 Large-scale computations

Compared to matrix problems, a tensor problem is more easily large-scale, even for modest-order tensors. Various techniques have been proposed to factorize large tensors, e.g., by using one or more samples of the tensor, by exploiting structure, by using randomization or by distributing the problem and doing computations in parallel. In the remainder of this section, we focus on these concepts in the context of CPD computations.

Example 19 (curse of dimensionality): Physical properties such as the melting temperature, are important design parameters when designing new alloys. These properties vary among others with the fraction of each component in the material, e.g., for stainless steel, we can have iron, carbon and chromium. If there are three components, two independent fractions can be varied to determine the property for all compositions. If the fractions are discretized as $0, 0.01, 0.02, \ldots, 0.99$, the values for all combinations can be arranged as a 100×100 matrix. When a component is added, there are three independent fractions and a tensor with 100^3 values is obtained. In general, for $N+1$ components, one needs to store and process 100^N values [304]. This exponential increase in memory and computational complexity is called the curse of dimensionality.

2.6.1 Compression

A common strategy to reduce the complexity of computing a CPD is to compress the tensor first using an MLSVD [46]. It then suffices to decompose the core tensor, which follows from the CANDELINC model [57], as the orthogonal compression preserves norms. Hence, let $\mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$ be the truncated MLSVD of $\mathcal{T}$ with core size $\min(I, R) \times \min(J, R) \times \min(K, R)$. The CPD of $\mathcal{T}$ is then given by $\left[\!\left[\mathbf{U}\hat{\mathbf{A}}, \mathbf{V}\hat{\mathbf{B}}, \mathbf{W}\hat{\mathbf{C}}\right]\!\right]$ in which the factor matrices $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$, and $\hat{\mathbf{C}}$ correspond to the CPD of the small core tensor $\mathcal{S}$. For large tensors, using the SVD to compute the MLSVD can be too expensive, and it may be necessary to use randomized SVDs [135], [306] or cross approximation [51], [196], [214] instead. Coupled datasets can be compressed jointly [49], [83].

Rather than computing the MLSVD and then creating an orthogonal projection onto the dominant subspaces, PARACOMP [245] uses random compression matrices to create multiple, randomly compressed smaller tensors. The CPDs of each of these smaller tensors are computed separately, and the

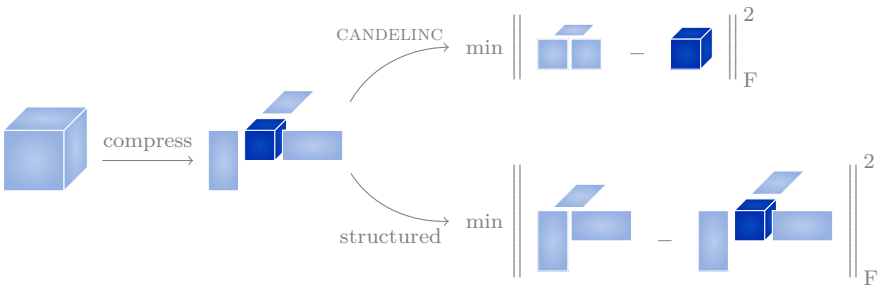


Figure 2.14: While the core tensor is decomposed in the CANDELINC model, the structured tensor framework replaces the tensor with its truncated MLSVD such that the original factor matrices are kept and constraints and coupling can be imposed easily.

results are merged using anchor rows to resolve permutation and scaling ambiguity. The multiway compressed sensing method [244] uses a single random compressed sample to recover the CPD, assuming that both the tensor and the factors are sparse.

When constraints or coupling are involved, decomposing the core tensor may no longer be possible as the constraint or coupling relation may not be preserved by the compression. Consider, for example, a nonnegativity constraint, i.e., $\mathbf{A} \geq 0$, in which $\geq$ holds entry-wise, and the compression matrix $\mathbf{U}$ such that $\mathbf{U}\hat{\mathbf{A}} = \mathbf{A}$, then the constraint $\hat{\mathbf{A}} \geq 0$ does not imply that $\mathbf{A} \geq 0$. The structured tensor decomposition framework proposed in [303] avoids this by exploiting the efficient representation of a tensor while keeping the original factor matrices, i.e., $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, in the optimization problem. Instead of working with the original $\mathcal{T}$, its truncated MLSVD $\mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$ is used and the multilinear structure of the MLSVD is exploited in all computations to reduce the computational complexity. Figure 2.14 illustrates the difference with the CANDELINC model. Efficient implementations that exploit the multilinear structure of the MLSVD are, e.g., given in [16], [303], [318]. The structured tensor framework also allows other compression types such as TT compression [303].

2.6.2 Sampling: incompleteness, randomization and updating

If the data is too large to be stored in memory or too expensive to measure or generate, it can be beneficial to avoid loading the entire tensor into memory at once, or even to avoid constructing all tensor entries. Here, we discuss techniques based on incomplete tensors, (randomized) block sampling and updating.

Incomplete tensors are used when some of the data is missing, e.g., due to sensor malfunction or artifact removal, or when the data is deliberately sampled in order to avoid the cost of generating all data. Two techniques are commonly used: single or repeated imputation, e.g., in an expectation maximization setting, and using a weight/sampling/observation tensor. The former technique imputes each unknown or missing entry with an estimate, thereby creating a full tensor which may not be feasible in a large-scale context; see [138], [220], [279]. The latter technique (implicitly) uses a binary weight tensor $\mathcal{W}$ in which an entry is one only if the corresponding entry in $\mathcal{T}$ is known, and solves

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

hence unknown entries are effectively ignored; see, e.g., [6], [157], [262], [279], [302].

When decomposing an incomplete tensor, only a part of the information is used. If all entries are available (explicitly or implicitly), it is also possible to repeatedly sample a number of entries and to compute an update for each sample, as is done in stochastic gradient descent [63], [112], [243], for example. In the case of a CPD, it is advantageous to sample subtensors, or *blocks*, thanks to the locality principle for a CPD: only a limited number of variables influences a block as can be seen in Figure 2.15 [300]. The ParCube algorithm [219] uses biased sampling to create a number of random blocks, decomposes these blocks using ALS and merges the results using anchor rows. In contrast to ParCube, the randomized block sampling (RBS) method [300] samples a block and computes only a single GN update using (2.5) and then samples a new block. RBS can achieve a high accuracy through step restriction strategies.

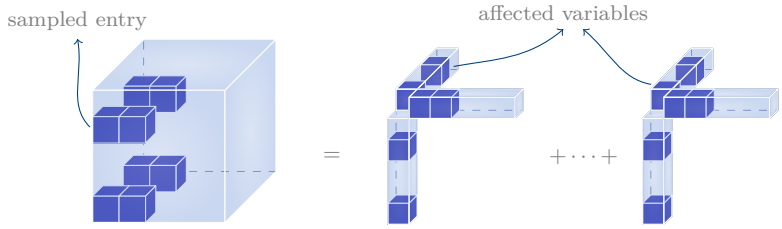


Figure 2.15: Thanks to the locality principle, only few variables in the rank-1 terms affect the sampled block or subtensor. *The CPD RBS method [300] samples a new random block every iteration, while ParCube [219] samples a number of blocks, decomposes them and uses anchor rows to merge the results.*

Updating algorithms can be used to track tensor decompositions that change over time [208], [274], [291], but can also be used to decompose large-

scale tensors as follows. Rather than loading the whole tensor into memory at once, a smaller subtensor is decomposed first. The subtensor can then be discarded, and a new slice is loaded and used to update the current decomposition. This process is then repeated until all slices have been added. Hence, at any given iteration, only the factorization constructed using the previous slices and one new slice are in memory; see [291].

2.6.3 Exploiting structure: sparsity and implicit tensorization

Tensors are often structured and can therefore be represented by few parameters, e.g., all values in a CPD depend on the factor matrices, all entries in a sparse tensor are defined by the positions and values of nonzeros, and a Hankel tensor is determined by one generating vector. By designing specialized algorithms, the per-iteration complexity can be reduced from proportional to the number of entries to proportional to the number of parameters [303].

For sparse tensors, a large number of ALS algorithms have been devised, many of which focus on the efficient implementation of the MTKRPROD operation: only the Khatri–Rao product contributions corresponding to nonzeros are computed to reduce the complexity. Instead of storing the multilinear index and the value for each nonzero [16], compressed formats such as the compressed sparse fiber (CSF) format can be used as well [251].

In case the tensor is given by the factors of a decomposition such as a CPD, an MLSVD or a TT, the multilinear structure can be exploited when computing, e.g., the inner product, norm and MTKRPROD, without actually constructing the full tensor [16], [303], [318]. A randomized technique requiring a CPD as input tensor is presented in [230]. A similar approach can be used for a tensor that is the result of a tensorization technique such as Hankelization or Löwnerization: the necessary expressions can, e.g., be computed efficiently by exploiting properties of Hankel and Toeplitz matrices through fast Fourier transforms (FFT) [303].

2.6.4 Parallelization

Distributed computing and parallelization can be used to alleviate the curse of dimensionality by allocating more computational resources. (The curse is not removed, however.) The entries of the tensor are distributed over multiple computational nodes which can be cores, processors or different computers in a cluster. Every iteration, each node computes an update for a part of the variables, after which the result is communicated to all nodes that require this update. (Not every node requires every variable due to the locality principle; see Figure 2.15.) Various algorithms exist that differ on the type of tensor (sparse or dense) and on how the data, and therefore the variables, are distributed across the nodes. This distribution depends on a trade-off between computational cost (computing the update), balancing computations such

that every node has the same amount of work, and communication cost (distributing updates). Coarse grained distribution schedules assign a set of rows from each factor matrix to a node and distribute the data accordingly such that each node has a part of the data; see, e.g., [64], [156], [186], [240]. Fine grained schedules compute optimal distributions to balance computational load and minimize computational cost; see, e.g., [158]. While coarse grained schedules are easy to determine, they can be suboptimal, e.g., due to an imbalance of the computational load. Albeit optimal schemes are obtained with fine grained schedules, they can be (too) expensive to determine. Medium grained schedules such as [253] determine a better distribution while limiting the cost of computing this distribution. Finally, some algorithms focus on minimizing the communication cost; see, e.g., [20].

Breaking the curse of dimensionality using decompositions of incomplete tensors

3

ABSTRACT | Tensors, or multiway arrays of numerical values, and their decompositions are common in domains like signal processing, data analysis and chemometrics. Being higher-order generalizations of vectors and matrices, tensors are often large-scale as their number of entries scales exponentially in the order. The computational and memory-related challenges created by this exponential dependence are referred to as the curse of dimensionality. In this chapter, we show that using a decomposition instead of the tensor can alleviate or even break this curse. Moreover, by sampling few entries, incomplete tensors can be used to alleviate or break the curse for the computation of these decompositions as well. We illustrate this for the canonical polyadic decomposition, and discuss similar concepts such as tensor trains and cross approximation, which are often used in scientific computing and quantum information theory. These concepts can be translated to a signal processing context, as is illustrated for multidimensional harmonic retrieval. In a materials science application, the melting temperature of an alloy is modeled using a low-rank CPD and we show that, using incomplete tensors, the curse is broken as the ninth-order tensor, which has $\mathcal{O}\left(10^{18}\right)$ entries, is decomposed using only 10^5 samples.

This chapter is based on 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, Sep. 2014. DOI: 10.1109/MSP.2014.2329429. An abstract has been added and the algorithms and figures have been updated for consistency.

3.1 Introduction

Higher-order tensors and their decompositions are abundantly present in domains such as signal processing (e.g., higher-order statistics [71], sensor array processing [242]), scientific computing (e.g., discretized multivariate functions [126], [129], [165], [211]) and quantum information theory (e.g., representation of quantum many-body states [210]). In many applications the, possibly huge, tensors can be approximated well by compact multilinear models or *decompositions*. Tensor decompositions are more versatile tools than the linear models resulting from traditional matrix approaches. Compared to matrices, tensors have at least one extra dimension. The number of elements in a tensor increases exponentially with the number of dimensions, and so do the computational and memory requirements. The exponential dependency (and the problems that are caused by it) is called the *curse of dimensionality*. The curse limits the order of the tensors that can be handled. Even for modest order, tensor problems are often large-scale. Large tensors can be handled, and the curse can be alleviated or even removed, by using a decomposition that represents the tensor, instead of the tensor itself. However, most decomposition algorithms require full tensors, which renders these algorithms infeasible for large datasets. If a tensor can be represented by a decomposition, this hypothesized structure can be exploited by using *compressed sensing* type methods working on *incomplete* tensors, i.e., tensors with only a few known elements.

In domains such as scientific computing and quantum information theory, tensor decompositions such as the Tucker decomposition and tensor trains have successfully been applied to represent large tensors. In the latter case, the tensor can contain more elements than the number of atoms in the universe [217] (estimated at $\mathcal{O}(10^{82})$). Algorithms to compute these decompositions using only a few *mode-n vectors* of the tensors have been developed to cope with the curse of dimensionality. In this chapter, we show on the one hand how decompositions already known in signal processing (e.g., the canonical polyadic decomposition and the Tucker decomposition) can be used for large and incomplete tensors, and on the other hand how existing decompositions and techniques from scientific computing can be used in a signal processing context. We conclude with a convincing proof-of-concept case study from materials science, in which the curse of dimensionality is effectively broken.

3.2 Notation and preliminaries

A general Nth order tensor of size $I_1 \times I_2 \times \cdots \times I_N$ is denoted by a calligraphic letter as $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$, and is a multiway array of numerical values $a_{i_1 i_2 \cdots i_N} = \mathcal{A}(i_1, i_2, \ldots, i_N)$. Tensors can be seen as a higher-order

generalization of vectors (denoted by a bold, lowercase letter, e.g., $\mathbf{a}$) and matrices (denoted by a bold, uppercase letter, e.g., $\mathbf{A}$). In the same way as matrices have rows and columns, tensors have mode-n vectors which are constructed by fixing all but one index, e.g., $\mathbf{a} = \mathcal{A}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N)$. The mode-1 vectors are the columns of the tensor, the mode-2 vectors are the rows of the tensor and so on. More generally, an nth order slice is constructed by fixing all but n indices. Tensors often need to be *reshaped*. An example is the mode-n matrix unfolding of a tensor $\mathcal{A}$ which arranges the mode-n vectors in a certain order as the columns of a matrix $\mathbf{A}_{(n)}$ [65], [170].

A number of products have to be defined when working with tensors. The outer product of two tensors $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\mathcal{B} \in \mathbb{C}^{J_1 \times J_2 \times \cdots \times J_M}$ is given as $(\mathcal{A} \otimes \mathcal{B})_{i_1 i_2 \cdots i_N j_1 j_2 \cdots j_M} = a_{i_1 i_2 \cdots i_N} b_{j_1 j_2 \cdots j_M}$. The mode-$n$ tensor-matrix product between a tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{B} \in \mathbb{C}^{J \times I_n}$ is defined as $(\mathcal{A} \cdot_n \mathbf{B})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 i_2 \cdots i_N} b_{j i_n}$. The Hadamard product $\mathcal{A} * \mathcal{B}$ for $\mathcal{A}, \mathcal{B} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_N}$ is the element-wise product. Finally, the Frobenius norm of a tensor $\mathcal{A}$ is denoted by $||\mathcal{A}||$ [65], [170].

3.3 Tensor decompositions

Most tensors of practical interest in applications are generated by some sort of process, e.g., a partial differential equation, a signal measured on a multi-dimensional grid, or the interactions between atoms. The resulting structure can be exploited by using decompositions which approximate the tensor using only a small number of parameters. By using tensor decompositions instead of full tensors, the curse of dimensionality can be alleviated or even removed. We look into three decompositions in this chapter: the canonical polyadic decomposition, the Tucker decomposition and the tensor train decomposition. We conclude with a more general concept from scientific computing and quantum information theory called tensor networks. For more theory and applications we direct the reader to the references, especially [65], [73], [126], [129], [165], [170].

3.3.1 Canonical polyadic decomposition

In a *polyadic decomposition* (PD), a tensor $\mathcal{T}$ is written as a sum of R rank-1 tensors, each of which can be written as the outer product of N factor vectors $\mathbf{a}_r^{(n)}$:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \mathbf{a}_r^{(2)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} \stackrel{\text{def}}{=} \left[\!\!\left[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right]. \tag{3.1}$$

The latter notation is a shorthand for the PD and the factor vectors $\mathbf{a}_r^{(n)}$ are the columns of the factor matrices $\mathbf{A}^{(n)}$ [170]. The PD is called *canonical*

(CPD) when R is the minimum number of rank-1 terms needed for (3.1) to be exact. In this case, R is the CP rank of the tensor. Ignoring the trivial indeterminacies due to scaling and ordering of the rank-1 terms, the CPD is unique under mild conditions [96]. The decomposition has many names such as PARAFAC (chemometrics), CANDECOMP (psychometrics) or R-term representation (scientific computing) [165], [170].

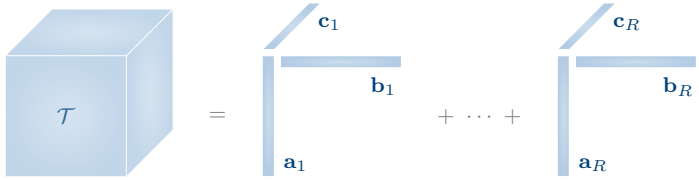


Figure 3.1: A *polyadic decomposition* of a third-order tensor $\mathcal{T}$ takes the form of a sum of R rank-1 tensors. *If R is the minimum number for the equality to hold, the decomposition is called* canonical*, and R is the rank of the tensor.*

In this decomposition only $R\left(\left(\sum_{n=1}^{N} I_n\right) - N + 1\right)$ variables are free (because of the scaling indeterminacy) which is $\mathcal{O}(NIR)$ assuming $I_n = I$, $n = 1, \ldots, N$. More importantly, it is linear in the number of dimensions N. This means the curse of dimensionality can be broken by using a CPD instead of a full tensor if the tensor admits a good CPD [26]. In many practical cases in signal processing R is low and $R \ll I$. In cases where the rank R cannot be derived from the problem definition, finding the rank is a hard problem. In practice, many CPDs will be fitted to the data until a sufficiently low approximation error is attained [26]. There is, however, no guarantee that this process yields the CP rank R, as the best rank-R approximation may not exist. This is due to the fact that the set of rank-R tensors is not closed, which means a sequence of rank-R' tensors with $R' < R$ can converge to a rank-R tensor while two or more terms grow without bounds. This problem is referred to as degeneracy [170], [175], [250]. By imposing constraints such as nonnegativity or orthogonality on the factor matrices, degeneracy can be avoided [65], [175], [262].

3.3.2 Tucker decomposition and low multilinear rank approximation

The Tucker decomposition of a tensor $\mathcal{T}$ is given as a multilinear transformation of a typically small core tensor $\mathcal{G} \in \mathbb{C}^{R_1 \times R_2 \times \cdots \times R_N}$ by factor matrices $\mathbf{A}^{(n)} \in \mathbb{C}^{I_n \times R_n}$, $n = 1, \ldots, N$:

$$\mathcal{T} = \mathcal{G} \cdot_1 \mathbf{A}^{(1)} \cdot_2 \mathbf{A}^{(2)} \cdots \cdot_N \mathbf{A}^{(N)} \stackrel{\text{def}}{=} \left[\!\left[\mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}\right]\!\right], \tag{3.2}$$

where the latter is a shorthand notation [170]. The N-tuple $(R_1, R_2, \ldots, R_N)$ for which the core size is minimal is called the multilinear rank. R_1 is the dimension of the column space, R_2 is the dimension of the row space, and more generally, R_n is the dimension of the space spanned by the mode-n vectors [170]. In general, the Tucker decomposition (3.2) is not unique, but the subspaces spanned by the vectors in the factor matrices are, which is useful in certain applications [65], [170]. In its original definition, the Tucker decomposition imposed orthogonality and ordering constraints on the factor matrices and the core tensor. In this definition, the Tucker decomposition can be interpreted as an higher-order generalization of the singular value decomposition (SVD) and can be obtained by reliable algorithms from numerical linear algebra (in particular algorithms for computing the SVD). In this context the names *multilinear SVD* (MLSVD) and *higher-order SVD* (HOSVD) are also used [78].

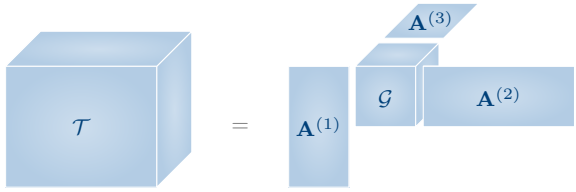


Figure 3.2: The Tucker decomposition of a third-order tensor $\mathcal{T}$ involves a multilinear transformation of a core tensor $\mathcal{G}$ by factor matrices $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$.

The number of parameters in the Tucker decomposition is $\mathcal{O}\left(NIR + R^N\right)$ when we take $I_n = I$ and $R_n = R$, $n = 1, \ldots, N$. This means the number of parameters in a Tucker decomposition still depends exponentially on the number of dimensions N. The curse of dimensionality is alleviated, however, as typically $R \ll I$. More generally, when a tensor is *approximated* by (3.2) where the size of the core tensor is chosen by the user, this decomposition is called a *low multilinear rank approximation* (LMLRA). As in PCA, a Tucker decomposition can be compressed or truncated by omitting small multilinear singular values [65], [170]. This reduction in R is beneficial given the exponential factor $\mathcal{O}\left(R^N\right)$ in the number of parameters, as the total number of parameters decreases exponentially. Note that truncated Tucker decomposition is just one, not necessarily optimal, way to obtain an LMLRA [78].

3.3.3 Tensor trains

Tensor trains (TT) are a concept from scientific computing and from quantum information theory where it is known as *matrix product states* (MPS)

[126], [129], [210], [211]. Each element in a tensor $\mathcal{T}$ can be written as

$$t_{i_1 i_2 \cdots i_N} = \sum_{r_1, r_2, \ldots, r_{N-1}} a^{(1)}_{i_1 r_1} a^{(2)}_{r_1 i_2 r_2} \cdots a^{(N)}_{r_{N-1} i_N},$$

with $r_n = 1, \ldots, R_n$, $n = 1, \ldots, N-1$. The matrices $\mathbf{A}^{(1)} \in \mathbb{C}^{I_1 \times R_1}$ and $\mathbf{A}^{(N)} \in \mathbb{C}^{R_{N-1} \times I_N}$ are the 'head' and 'tail' of the train; the core tensors $\mathcal{A}^{(n)} \in \mathbb{C}^{R_{n-1} \times I_n \times R_n}$, $n = 2, \ldots, N-1$, are the 'carriages' as can be seen in Figure 3.3. The auxiliary indices R_n, $n = 1, \ldots, N-1$ are called the *compression ranks* or the TT ranks [211]. It can be proven that the compression ranks are bounded by the CP rank of a tensor [216].

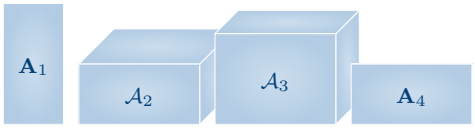


Figure 3.3: A fourth-order tensor $\mathcal{T}$ can be written as a *tensor train* by linking a matrix $\mathbf{A}^{(1)}$, two tensors $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$ (the carriages) and a matrix $\mathbf{A}^{(4)}$.

A TT combines the good properties of the CPD and the Tucker decomposition. The number of parameters in a TT is $\mathcal{O}\left(2IR + (N-2)IR^2\right)$ assuming $I_n = I$, $R_n = R$, $n = 1, \ldots, N$, which is linear in the number of dimensions, similar to a CPD [129], [211]. This means a TT is suitable for higher-order problems, as using it removes the curse of dimensionality. As for the Tucker decomposition, numerically reliable algorithms such as the SVD can be used to compute the decomposition [78], [211].

3.3.4 Tensor networks

The TT decomposition represents a higher-order tensor as a set of linked (lower-order) tensors and matrices, and is an example of a linear *tensor network*. A more general tensor network is a set of interconnected tensors. This can be visualized using *tensor network diagrams* (see Figure 3.4) [165], [210]. Each vector, matrix or tensor is represented as a dot. The order of each tensor is determined by the number of edges connected to it. An interconnection between two dots represents a *contraction*, which is the summation of the products over a common index. Tensor network diagrams are an intuitive and visual way to efficiently represent decompositions of higher-order tensors. An example is the *hierarchical Tucker* (HT) decomposition (see Figure 3.4), which is another important decomposition used in scientific computing [126], [129], [165]. More complicated tensor networks can also contain cycles, e.g., *tensor chains* and *projected entangled-pair states* (PEPS) from quantum physics [126], [210].

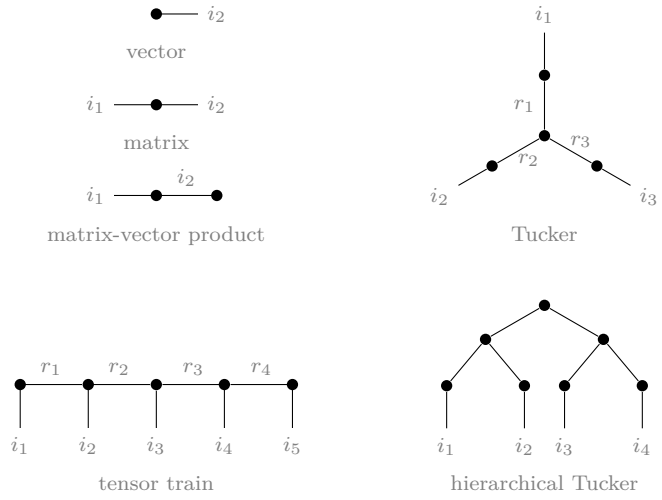


Figure 3.4: Different types of tensor networks.

3.4 Computing decompositions of large, incomplete tensors

To compute tensor decompositions, most algorithms require a full tensor and are therefore not an option for large and high-order datasets. The knowledge that the data is structured and can be represented by a small number of parameters can be exploited by sampling the tensor in only a few elements. Then, the decomposition is calculated using an *incomplete* tensor. There are two important situations where incomplete datasets are used. In the first case some elements are unknown, e.g., because of a broken sensor [6], or unreliable, e.g., because of Rayleigh scattering [250], and the matrix or tensor needs to be *completed* [110]. In the second case, the cost of acquiring a full tensor is too high in terms of money, time or storage requirements. By sampling the tensor in only a few elements, this cost can be reduced.

Compressed sensing (CS) methods are used to reconstruct signals using only a few measurements taken by a linear projection of the original dataset [54]. Many extensions of these methods to tensors have been developed [52], [65] and new methods tailored to tensors have emerged, e.g., [185], [244]. In this chapter we focus on a class of CS methods where decompositions of very large tensors are computed using only a small number of known elements. In particular, we first discuss methods to compute a CPD from a randomly sampled incomplete tensor. Then we discuss how matrices can be approximated by extracting only a few rows and columns. This idea can be extended to tensors, and we conclude by elaborating on two mode-n vector

sampling methods: one for the TT decomposition and one for the LMLRA.

3.4.1 Optimization-based algorithms

Most algorithms to compute a CPD use optimization[1] to find the factor matrices $\mathbf{A}^{(n)}$:

$$\min_{\mathbf{A}^{(1)},\mathbf{A}^{(2)},\ldots,\mathbf{A}^{(N)}} \frac{1}{2} \left|\left| \mathcal{T} - \left[\!\left[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] \right|\right|^2, \tag{3.3}$$

which is a least squares problem in each factor matrix separately. The popular *alternating least squares* (ALS) method alternately solves a least squares problem for one factor matrix while fixing the others. This method is easy to implement and works well in many cases, but has a linear convergence rate and tends to be slow when the factor vectors become more aligned. It is even possible that the algorithm does not converge at all [5], [260]. CP-OPT uses a nonlinear conjugate gradients method to solve (3.3) [5]. By using first-order information, the method also achieves linear convergence. Recently, some new methods based on nonlinear least squares (NLS) algorithms have been developed. These methods exploit the structure in the objective function's approximate Hessian. Due to the NLS framework, second-order convergence can be attained under certain circumstances [226], [260]. The latter two methods are both guaranteed to converge to a stationary point, which can be a local optimum, however.

Although efficient methods exist, the complexity of all methods working on full tensors is at least $\mathcal{O}\left(I^N\right)$, which becomes infeasible for large, high-dimensional tensors. To handle missing data, problem (3.3) can be adapted to

$$\min_{\mathbf{A}^{(1)},\mathbf{A}^{(2)},\ldots,\mathbf{A}^{(N)}} \frac{1}{2} \left|\left| \mathcal{W} * \left(\mathcal{T} - \left[\!\left[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] \right) \right|\right|^2,$$

where $\mathcal{W} \in \{0,1\}^{I_1 \times I_2 \times \cdots \times I_N}$ is a binary observation tensor with a one for every known element [6]. The popular ALS method has been extended by using an *expectation maximization* (EM) framework to impute each missing value with a value from the current CP model [250]. Because of the imputation, the ALS-EM method still suffers from curse of dimensionality. The CP-WOPT method is an extension of the CP-OPT method and uses only the known elements, thereby relaxing the curse of dimensionality [6]. Adaptions of the Jacobian and the Gramian of the Jacobian for incomplete tensors can be used in an inexact nonlinear least squares framework. Second-order convergence can again be attained under certain circumstances, while the

[1] We only focus on optimization-based algorithms for the CPD in this section; For examples of optimization-based algorithms for the LMLRA or the TT approximation, see, e.g., [125], [188].

computational complexity is still linear in the number of known elements [89].

The distribution of the known elements in the tensor can be random, although performance may decrease in case of missing mode-n vectors or slices [6], [89]. The elements should be known a priori, contrary to the mode-n vector-based algorithms. Constraints such as nonnegativity or a Vandermonde structure can easily be added to the factor matrices, which is useful for many signal processing applications [262].

3.4.2 Pseudo-skeleton approximation for matrices

Instead of randomly sampling a tensor, a more drastic approach can be taken by sampling only mode-n vectors and only using these mode-n vectors in the decomposition. These techniques originate from the *pseudoskeleton approximation* or the *CUR* decomposition for matrices. These decompositions state that a matrix $\mathbf{A} \in \mathbb{C}^{I_1 \times I_2}$ of rank R can be approximated using only R columns and R rows of this matrix:

$$\mathbf{A} = \mathbf{CGR}, \tag{3.4}$$

where $\mathbf{C} \in \mathbb{C}^{I_1 \times R}$ has R columns with indices $\mathcal{J}$ of $\mathbf{A}$, i.e., $\mathbf{C} = \mathbf{A}(:, \mathcal{J})$, $\mathbf{R} \in \mathbb{C}^{R \times I_2}$ has R rows with indices $\mathcal{I}$ of $\mathbf{A}$, i.e., $\mathbf{R} = \mathbf{A}(\mathcal{I}, :)$ and $\mathbf{G} = \hat{\mathbf{A}}^{-1}$, where $\hat{\mathbf{A}}$ contains the intersection of $\mathbf{C}$ and $\mathbf{R}$, i.e., $\hat{\mathbf{A}} = \mathbf{A}(\mathcal{I}, \mathcal{J})$. If $\text{rank}(\mathbf{A}) = R$, then (3.4) is exact when $\mathbf{C}$ has R linearly independent columns of $\mathbf{A}$ and $\mathbf{R}$ has R linearly independent rows of $\mathbf{A}$ (which implies that $\hat{\mathbf{A}}$ is nonsingular) [217]. Usually we are interested in the case where $\text{rank}(\mathbf{A}) > R$. A good choice for the submatrix $\hat{\mathbf{A}}$ (and consequently $\mathbf{C}$ and $\mathbf{R}$) is in this case the $R \times R$ submatrix having the largest volume, which is given by the modulus of its determinant, as it is sufficient to find an approximation within a provable and often reasonable bound [120], [121].

To determine the optimal submatrix $\hat{\mathbf{A}}$, the determinants of all possible submatrices have to be evaluated. This is computationally challenging and, moreover, all the elements of the matrix have to be known. A heuristic called *cross approximation* (CA) can be used to calculate a quasi-optimal maximal volume submatrix by only looking at a few rows and columns. The following general scheme can be used (based on [286]). An initial column index set $\mathcal{J} \subset \{1, 2, \ldots, I_2\}$ and an initial row index set $\mathcal{I} \subset \{1, 2, \ldots, I_1\}$ are chosen and $\mathbf{C}$ is defined as $\mathbf{A}(:, \mathcal{J})$. Then the submatrix $\hat{\mathbf{C}} = \mathbf{C}(\hat{\mathcal{I}}, :)$ with (approximately) the largest volume is calculated, for example, using a technique based on full pivoting [286]. Next, the process is repeated for the rows, i.e., the subset $\hat{\mathcal{J}}$ resulting in the maximal volume submatrix $\hat{\mathbf{R}} = \mathbf{R}(:, \hat{\mathcal{J}})$ in $\mathbf{R} = \mathbf{A}(\mathcal{I}, :)$ is calculated. Next, the index sets are updated as $\mathcal{J} = \mathcal{J} \bigcup \hat{\mathcal{J}}$ and $\mathcal{I} = \mathcal{I} \bigcup \hat{\mathcal{I}}$ and the process is repeated until a stopping criterion is met, e.g., when the

norm of the residual $||\mathbf{A} - \mathbf{CGR}||$ is small enough. To calculate the norm, only the extracted rows and columns are taken into account. To make this more concrete, we give a simple method selecting one column and row at a time in Algorithm 3.1. For more details, we refer the interested reader to [134], [214], [217], [286]. In the case R is not specified, but only a desired accuracy ϵ is given, adaptive cross approximation (ACA) techniques can be used, which adapt the index sets $\mathcal{I}$ and $\mathcal{J}$ as well as the rank R automatically based on ϵ. A key component in ACA is the stopping criterion based on a suitable error estimate that does not require all entries in the matrix; see, e.g., [22], [23], [134] for such criteria and more details.

Algorithm 3.1: Cross approximation for matrices [214].

1: Set $\mathcal{J} = \emptyset$, $\mathcal{I} = \emptyset$, $j_1 = 1$ and $p = 1$.
2: Extract the column $\mathbf{A}(:, j_p)$ and find the maximal volume submatrix in the residual, i.e., the largest element in modulus in the vector $\mathbf{a}_{j_p}$ minus the corresponding elements in the already known rank-1 terms, and set i_p to its location.
3: Extract the row $\mathbf{A}(i_p, :)$ and find the maximal volume submatrix in the residual that is not in the previously chosen column j_p, and set j_{p+1} to its location.
4: Set $\mathcal{J} = \mathcal{J} \bigcup \{j_p\}$, $\mathcal{I} = \mathcal{I} \bigcup \{i_p\}$.
5: If the stopping criterion is not satisfied, set $p = p + 1$ and go to step 2.

3.4.3 Cross approximation for TT

Before we outline a CA-based algorithm for the TT decomposition, we first present a simplified version of a TT algorithm for full tensors based on repeated truncated SVD in Algorithm 3.2. The truncation step 3 determines the compression rank R_n. In a scientific computing context, the compression ranks R_n are chosen such that the decomposition approximates the (noise free) tensor with a user-defined accuracy ϵ [211]. In signal processing, the tensor is often perturbed by noise. Therefore, the compression ranks R_n can be determined by using a procedure estimating the noise level.

Algorithm 3.2: Tensor train decomposition using SVD [217].

1: Set $R_0 = 1$ and $\mathbf{M} = \text{reshape}\left(\mathcal{T}, \left[I_1, \prod_{k=2}^{N} I_k\right]\right)$.
2: **for** $n = 1, \ldots, N-1$ **do**
3: Calculate the (truncated) SVD: $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}}$.
4: Set $\mathcal{A}^{(n)} = \text{reshape}\left(\mathbf{U}, [R_{n-1}, I_n, R_n]\right)$.
5: Set $\mathbf{M} = \text{reshape}(\mathbf{\Sigma}\mathbf{V}^{\mathrm{H}}, [R_n I_{n+1}, \prod_{k=n+2}^{N} I_k])$ if $n < N-1$.
6: **end for**
7: $\mathbf{A}^{(N)} = \mathbf{\Sigma}\mathbf{V}^{\mathrm{H}}$.

The use of the SVD in Line 3 has two disadvantages: all the elements in the tensor need to be known and when this tensor is large, calculating the SVD is expensive. In [217] a CA type method is suggested: the SVD can be replaced by a pseudoskeleton approximation as described above by replacing $\mathbf{U}$ with $\mathbf{CG}$ and $\mathbf{\Sigma V}^{\mathrm{H}}$ with $\mathbf{R}$. The matrix $\mathbf{R}$ does not require extra calculations and working with $\mathbf{R}$ does not require additional memory as it can be handled implicitly by selecting the proper indices. This algorithm requires the compression ranks to be known in advance. By using a compression or *rounding* algorithm on the resulting TT decomposition, the compression rank can be overestimated safely; see, e.g., [211] for a compression method based on the truncated SVD. In a signal processing context, this rounding algorithm can be adapted to use a noise level estimation procedure instead of using a user-defined accuracy ϵ. For practical implementation details we refer to [217].

The positions of the elements needed by the CA algorithm are unknown a priori, but are generated based on information in the mode-n vectors that already have been extracted. In a scientific computing context, where the tensor often is given as a multivariate function, this is not a problem as sampling an entry is evaluating this function. In a signal processing context, this means that either the elements are sampled while running the CA algorithm, or the full tensor has to be known a priori. This last condition can be relaxed, however, by *imputing* unknown elements in selected mode-n vectors by an estimate of the value of these elements, e.g., the mean value over the mode-n vector. This, however, only works well if the imputed value effectively is a good estimator of the unknown value. Only $\mathcal{O}(2KNR)$ columns of length $R_{n-1}I_n$ are investigated during the CA algorithm, where K is the number of iterations in the CA algorithm and assuming $R_n = R$, $n = 1, \ldots, N-1$. If the compression ranks and the number of iterations are low, very few elements need to be sampled.

3.4.4 Cross approximation for LMLRA

The CA method for TT essentially only replaced the SVD with a pseudoskeleton approximation. In case of the LMLRA we look at another generalization of the pseudoskeleton approximation method: $\mathcal{T}$ can be approximated by $\hat{\mathcal{T}} = \left[\!\left[\mathcal{G}; \mathbf{C}^{(1)}\mathbf{G}^{\dagger}_{(1)}, \ldots, \mathbf{C}^{(N)}\mathbf{G}^{\dagger}_{(N)}\right]\!\right]$ where $\mathbf{C}^{(n)}$ contains $\prod_{m\neq n} R_m$ mode-n vectors for $n = 1, \ldots, N$ and where the size of the core tensor $\mathcal{G}$ is $R_1 \times \cdots \times R_N$. The core tensor is the subtensor of $\mathcal{T}$ containing the intersection of the selected mode-n vectors. More concretely, we define the index sets $\mathcal{I}^{(n)} \subset \{1, \ldots, I_n\}$, $n = 1, \ldots, N$. Each column of $\mathbf{C}^{(n)}$ contains a selected mode-n vector defined by an index set in $\bigtimes_{m\neq n}\mathcal{I}^{(m)}$, i.e., $\mathcal{T}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N)$ with $(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) \in \bigtimes_{m\neq n}\mathcal{I}^{(m)}$. $\mathbf{C}^{(n)}$ thus is a matricized $(R_1 \times \cdots \times R_{n-1} \times I_n \times R_{n+1} \times \cdots \times R_N)$ subtensor

of $\mathcal{T}$. The intersection core tensor then is defined as $\mathcal{G} = \mathcal{T}(\mathcal{I}^{(1)}, \ldots, \mathcal{I}^{(N)})$. To determine the index sets $\mathcal{I}^{(n)}$ an adaptive procedure can be used. Each iteration the index $i^{(n)}$ having the largest modulus of the residual in the mode-n vector through the pivot, is added to $\mathcal{I}^{(n)}$. The residual is defined as $\mathcal{E} = \mathcal{T} - \hat{\mathcal{T}}$ where the matrices $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$ and the core tensor $\mathcal{G}$ are defined by the current index sets $\mathcal{I}^{(n)}$. A simplified version of this *fiber sampling tensor decomposition algorithm* [51] is given in Algorithm 3.3. Each matrix $\mathbf{C}^{(n)}$ contains $|\times_{m \neq n} \mathcal{I}^{(m)}| = \mathcal{O}\left(R^{N-1}\right)$ columns. The total number of mode-n vectors of length I_n that has to be extracted in this algorithm is then $\mathcal{O}\left(NR^{N-1}\right)$. Similarly to the pseudoskeleton approximation, an exact decomposition based on fiber sampling can be attained if $\mathcal{T}$ has an exact LMLRA structure and multilinear rank $(R_1, \ldots, R_N)$, i.e., $\mathcal{T} = [\![\mathcal{G}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$. In this case, it can be proven that only R_n mode-n fibers per mode n and $\prod_{n=1}^{N} R_n$ core elements have to be extracted [51]. In both cases, the computational complexity still has an exponential dependence on the number of dimensions [51]. Even with CA, the representation of a tensor as an LMLRA is limited by the curse of dimensionality to low-dimensional problems. We can make the same remarks for this method as for the TT decomposition concerning the fact that the indices of the required elements are only known at runtime. A variation of this algorithm determines the largest element in slices instead of in mode-n vectors [214]. An alternative to the pseudoskeleton approach, is to sample mode-n vectors after a fast estimation of probability densities [196].

Algorithm 3.3: Fiber sampling tensor decomposition algorithm [51].

1: Choose an initial mode-N vector in $\mathcal{T}$ defined by $i_1^{(n)}$, for $n = 1, \ldots, N-1$ and set $i_1^{(N)}$ to the index containing the maximal modulus in this fiber.
2: Set $\mathcal{I}^{(n)} = \{i_1^{(n)}\}$ for $n = 1, \ldots, N$ and set the pivot to $(i_1^{(1)}, \ldots, i_1^{(N)})$.
3: **for** $r = 2, \ldots, R$ **do**
4: **for** $n = 1, \ldots, N$ **do**
5: Select the index $i_r^{(n)}$ of the maximal modulus of the mode-n vector $\mathbf{e}$ going through the pivot in the residual tensor $\mathcal{E}$ (or in $\mathcal{T}$ for $n = 1$, $r = 2$), i.e., in $\mathbf{e} = \mathcal{E}(i_r^{(1)}, \ldots, i_r^{(n-1)}, :, i_{r-1}^{(n+1)}, \ldots, i_{r-1}^{(N)})$.
6: Set $\mathcal{I}^{(n)} = \mathcal{I}^{(n)} \bigcup \{i_r^{(n)}\}$ and select $(i_r^{(1)}, \ldots, i_r^{(n)}, i_{r-1}^{(n+1)}, \ldots, i_{r-1}^{(N)})$ as new pivot.
7: **end for**
8: **end for**

3.5 Case studies

To illustrate the use of the decompositions and incomplete tensors, two case studies are reported. The first case study shows how the concepts can be

applied in a signal processing context. To compare the results with full tensor methods, moderate size tensors are used. The second case study gives an example from materials science, where a huge tensor is decomposed while using only a very small fraction of the elements.

3.5.1 Multidimensional harmonic retrieval

Multidimensional harmonic retrieval (MHR) problems appear frequently in signal processing, e.g., in radar applications and channel sounding [191]. To model a multipath wireless channel, for example, a broadband wireless channel sounder can be used to measure a (time-varying) channel in the time, frequency and spatial domains. The measurement data can then be transformed into a tensor:

$$y_{i_1 i_2 \cdots i_D k} = \sum_{r=1}^{R} s_r(k) \prod_{d=1}^{D} e^{\mathrm{j}(i_d - 1)\mu_r^{(d)}} + n_{i_1 i_2 \cdots i_D k}, \tag{3.5}$$

where $\mathrm{j}^2 = -1$ and $s_r(k)$ is the kth complex symbol carried by the rth multidimensional harmonic. The parameters $\mu_r^{(d)}$ are, for example, the direction of departure, the direction of arrival, the Doppler shift, the delay, and so on. (For more information we refer the interested reader to [191].) The noise $n_{i_1 i_2 \cdots i_D k}$ is modeled as zero mean i.i.d. additive Gaussian noise. We can rewrite the model as a CPD

$$\mathcal{Y} = \left[\!\!\left[\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(D)}, \mathbf{S} \right]\!\!\right] + \mathcal{N} \tag{3.6}$$

with the Vandermonde structured factor matrices $\mathbf{A}^{(d)} \in \mathbb{C}^{I_d \times R}$, $a_{i_d,r}^{(d)} = e^{\mathrm{j}(i_d - 1)\mu_r^{(d)}}$ and $\mathbf{S} \in \mathbb{C}^{K \times R}$ with $s_{kr} = s_r(k)$. In the noiseless case, the CP rank of this tensor is equal to R. The multilinear ranks and the TT ranks are at most R. Uniqueness properties for the problem (3.6) are given in [263].

To estimate the parameters $\mu_r^{(d)}$, a subspace-based approach is used. First, $\mathcal{Y}$ is decomposed using a CPD, an LMLRA or a TT. Then, in case of the LMLRA and TT, we compute the subspaces $\mathbf{B}^{(d)}$ spanned by the mode-d vectors, $d = 1, \ldots, D$. (The parameters can be estimated directly from the factor vectors in case of CPD.) Finally, we use a standard *total least squares* method to estimate the parameters $\hat{\boldsymbol{\mu}}^{(d)}$ from these subspaces (see [289]). Here, we focus on the first two steps, i.e., the approximation of the full or incomplete tensor $\mathcal{Y}$ by a CPD, an LMLRA or a TT and the computation of the subspaces.

In case of a CPD, we use the `cpd_nls` method from Tensorlab [260], [261] to get an estimate of the factor matrices $\hat{\mathbf{A}}^{(d)}$, $d = 1, \ldots, D+1$. This method works on both full and incomplete tensors. Here, we can estimate the parameters directly as the generators of the noisy Vandermonde vectors $\mathbf{a}_r^{(d)}$,

$d = 1, \ldots, D$, so the computation of the subspaces is not necessary. The number of sampled entries N_{samples} can be chosen by the user (see Table 3.1).

In case of an LMLRA $\left[\!\left[\hat{\mathcal{G}}; \hat{\mathbf{A}}^{(1)}, \ldots, \hat{\mathbf{A}}^{(D+1)}\right]\!\right]$, we first compute the decomposition using `lmlra` from Tensorlab [261], which uses an NLS-based optimization method on the full tensor, and using `lmlra_aca`, which implements a fiber sampling adaptive cross approximation technique. In the latter case, the choice of the core size $R_1 \times \cdots \times R_{D+1}$ controls the number of touched elements, i.e., the number of elements from the tensor that are used during the algorithm (see Table 3.1). Recall that R is the number of multidimensional harmonics in (3.5). The subspaces $\hat{\mathbf{B}}^{(d)}$ are now computed using the first R left singular vectors of the unfolded product $(\hat{\mathcal{G}} \cdot_d \hat{\mathbf{A}}^{(d)})_{(d)}$. (It can be verified that these are the dominant mode-d vectors of $\left[\!\left[\hat{\mathcal{G}}; \hat{\mathbf{A}}^{(1)}, \ldots, \hat{\mathbf{A}}^{(D+1)}\right]\!\right]$ if the factor matrices are normalized to have orthonormal columns.)

Finally, in case of TT, we compute the TT cores $\hat{\mathbf{A}}^{(1)}$, $\hat{\mathcal{A}}^{(d)}$, $\hat{\mathbf{A}}^{(D+1)}$ using `tt_full`, which uses the truncated SVD of the full tensor (cf. supra), and using `dmrg_cross`, which uses cross approximation and touches only a limited number of mode-n vectors (cf. supra). Both methods are available in the TT-toolbox (`http://spring.inm.ras.ru/osel/`). The number of touched elements is controlled by the compression ranks R_n and the number of iterations K (see Table 3.1). The estimates for the subspaces $\hat{\mathbf{B}}^{(1)}$ and $\hat{\mathbf{B}}^{(d)}$ can be computed using the first R left singular vectors of $\hat{\mathbf{A}}^{(1)}$ and of the mode-2 unfolding $\hat{\mathcal{A}}^{(d)}_{(2)}$, $d = 2, \ldots, D$, respectively.

Table 3.1: Number of parameters and touched elements for the three decompositions of incomplete tensors. *The number of touched elements concern the presented algorithmic variants. In case of a CPD and TT, the curse of dimensionality can be overcome.*

	# Parameters	# Touched elements
CPD	$\mathcal{O}(NIR)$	N_{samples}
LMLRA	$\mathcal{O}\left(NIR + R^N\right)$	$\mathcal{O}\left(NIR^{N-1}\right)$
TT	$\mathcal{O}\left(2IR + (N-2)IR^2\right)$	$\mathcal{O}\left(2KNR^2I\right)$

With two experiments, we show how the number of touched elements and noise influence the quality of the retrieved parameters using the three decompositions. We create an $8 \times 8 \times 8 \times 8 \times 20$ tensor $\mathcal{Y}$ with rank $R = 4$ according to (3.6). The $D = 4$ parameter vectors are chosen as follows: $\boldsymbol{\mu}^{(1)} = [1.0, -0.5, 0.1, -0.8]$, $\boldsymbol{\mu}^{(2)} = [-0.5, 1.0, -0.9, 1.0]$, $\boldsymbol{\mu}^{(3)} = [0.2, -0.6, 1.0, 0.4]$, $\boldsymbol{\mu}^{(4)} = [-0.8, 0.4, 0.3, -0.1]$. Each of the $R = 4$ uncorrelated binary phase shift keying (BPSK) sources takes $K = 20$ values. To evaluate the quality of

the estimates, the *root mean square error* (RMSE) is used:

$$E_{\mathrm{RMS}} = \sqrt{\frac{1}{RD}\sum_{r=1}^{R}\sum_{d=1}^{D}\left(\mu_r^{(d)} - \hat{\mu}_r^{(d)}\right)^2}.$$

For each experiment, the median value over 100 Monte Carlo runs is reported.

In the first experiment, the signal-to-noise ratio (SNR) is fixed to 20 dB while the fraction of missing entries varies (see Figure 3.5, left). When there are no missing entries, we use the corresponding algorithms for full tensors. All methods then attain a similar accuracy. When the fraction of missing entries is increased, the error E_{RMS} also increases, except for TT where the error remains almost constant, but is higher than for CPD and LMLRA. An increase in the error is expected, as there are fewer noisy samples to estimate the parameters from. For 99% missing entries, the CPD algorithm no longer finds a solution as the number of known entries (820) is close to the number of free parameters (204). The CPD-based method has the best performance. ($\mathcal{Y}$ has a CPD structure to start from).

In the second experiment, the number of known elements is kept between 8% and 12% (remember that it is difficult to control the accesses in an adaptive algorithm) and the SNR is varied (see Figure 3.5, right). In case of the full tensor methods, E_{RMS} is almost equal for all decompositions, unless for low SNR. In case of the incomplete methods, the CPD-based method performs better, especially in the low SNR cases.

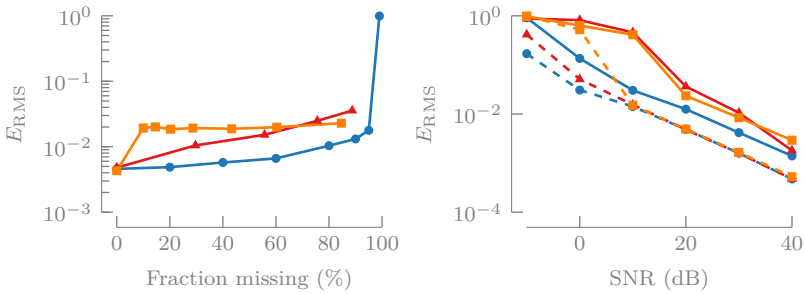


Figure 3.5: Influence of the number of known elements (left) and SNR (right) on E_{RMS}. *Results shown for the CPD (—●—), the LMLRA (—▲—) and TT (—■—). The dashed lines give the results for the full tensor methods.*

3.5.2 Materials science example

When designing new materials, the physical properties of these new materials are key parameters. In the case of alloys, the concentrations of the different constituent materials can be used to model the physical properties. In this particular example, we model the melting point of an alloy, using a dataset kindly provided by InsPyro NV, Belgium. The dataset contains a small set of random measurements of the melting point in function of the concentrations of ten different constituent materials. This dataset can be represented as a ninth-order tensor $\mathcal{T}$. (One concentration is superfluous as concentrations must sum to 100%.) The curse of dimensionality is an important problem for this kind of data, as the number of elements in this tensor is approximately $100^N = 10^{18}$ with $N+1$ the number of constituent materials. Because measuring and computing all these elements is infeasible, only 130 000 elements are sampled.

This case study illustrates how a tensor decomposition algorithm for incomplete tensors can overcome the curse of dimensionality. We use the `cpdi_nls` algorithm [89] because it is suitable for a dataset containing only a small fraction of randomly sampled elements. In particular, we approximate the training tensor $\mathcal{T}_{\text{tr}}$ which contains 70% of the data, by a CPD $\hat{\mathcal{T}} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(9)}]\!]$ and repeat this for several ranks R. To evaluate the quality of a rank-R model the validation error E_{val} of the model is computed using an independent validation tensor $\mathcal{T}_{\text{val}}$ containing the remaining 30% of the data. This error is defined as the weighted relative norm of the error between $\mathcal{T}_{\text{val}}$ and the model:

$$E_{\text{val}} = \frac{||\mathcal{W}_{\text{val}} * (\mathcal{T}_{\text{val}} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(9)}]\!])||}{||\mathcal{W} * \mathcal{T}_{\text{val}}||}.$$

The binary observation tensor $\mathcal{W}_{\text{val}}$ has only ones at the positions of known validation elements. We also report the 99% quantile of the relative residuals between known elements in the validation set and the model as E_{quant}. The timing experiments are performed on a laptop (Intel core i7, quadcore @2.7 GHz, 8 GB RAM, Matlab 2013b).

To compute a CPD from the training tensor, the `cpdi_nls` method [89] is used. This method is an extension of the `cpd_nls` method from Tensorlab [260], [261] for incomplete tensors. When choosing the initial factor matrices, we have to take the high order N into account: from (3.1) we see that every element in $\hat{\mathcal{T}}$ is the sum of R products of $N = 9$ variables. This means that, if most elements in the factor matrices are close to zero, $\mathcal{T} \approx 0$. Here, we have drawn the elements in the initial factor matrices from a uniform distribution in $(0, 1)$, and we have scaled each factor vector $\mathbf{a}_r^{(n)}$, $n = 1, \ldots, N$ by $\sqrt[N]{\lambda_r}$ where λ_r are the minimizers of $\left|\left|\mathcal{W}_{\text{tr}} * (\mathcal{T}_{\text{tr}} - \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(1)})\right|\right|$. Finally, we use a best-out-of-five strategy, which means that we choose five

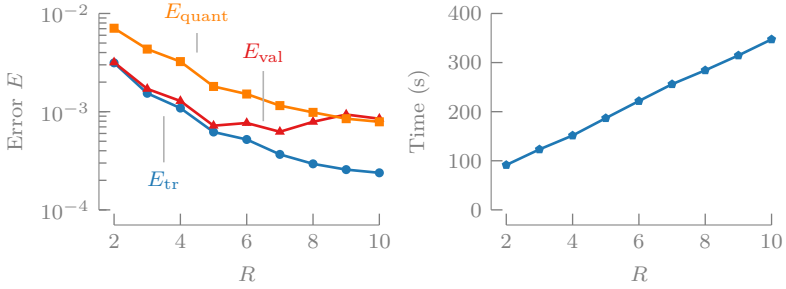


Figure 3.6: Using a rank-5 model is a good trade-off between accuracy and computation time and avoids overmodeling as can be seen for the validation error.

different optimally scaled initial solutions and keep the best result in terms of error on the training tensor E_{tr}. The result is shown in Figure 3.6. Both E_{tr} and E_{quant} keep decreasing as R increases, which indicates that also outliers are modeled when more rank-1 terms are added to the model. Starting from $R = 5$, E_{val} and E_{tr} start diverging, which can indicate that the data is overmodeled for $R > 5$, although E_{quant} keeps decreasing. For the remaining of this case study, we assume $R = 5$ to be a good choice as rank: the relative error E_{quant} is smaller than $1.81 \cdot 10^{-3}$ for 99% of the validation points, while it only took 3 minutes to compute the model. (The time rises linearly in R as can be seen in Figure 3.6.)

To summarize: by using 10^5 elements we have reduced a dataset containing 10^{18} elements to a model having $NIR \approx 4500$ parameters. We can now go one step further by looking at the values in the different factor vectors (see Figure 3.7). We see that the factor vectors have a smooth, low degree polynomial-like behavior, a little perturbed by noise. By fitting smooth spline functions to each factor vector, a continuous model for the physical parameter can be created:

$$\mathcal{T} \approx f(c_1, \ldots, c_N) = \sum_{r=1}^{R} \prod_{n=1}^{N} a_r^{(n)}(c_n),$$

where $a_r^{(n)}$ are continuous functions in the concentrations c_n, $n = 1, \ldots, N$. This has many advantages: the high-dimensional model can be visualized more easily and all elements having a certain melting point can be calculated (see, for example, Figure 3.8). Furthermore, the model can be used in further steps in the design of the material.

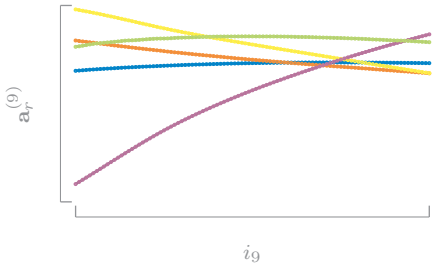


Figure 3.7: The values in the $R = 5$ factor vectors follow a smooth function. *Here the factor vectors for the ninth mode* $\mathbf{a}_r^{(9)}$ *are shown as dots.*

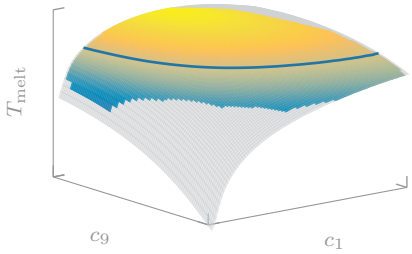


Figure 3.8: Visualization of the continuous surface of melting points when all but two concentrations are fixed. *The blue line links all points having a melting temperature of* $1400\,^\circ C$. *The model is only valid in the colored region.*

3.6 Conclusion

Tensor decompositions open up new possibilities in analysis and computation, as they can alleviate or even break the curse of dimensionality that occurs when working with higher-order tensors. Decompositions such as the tensor train decomposition are often used in other fields such as scientific computing and quantum information theory. These decompositions can easily be ported to a signal processing context. We have addressed some problems when computing decompositions of full tensors. By exploiting the structure of a tensor, compressed sensing type methods can be used to compute these decompositions using incomplete tensors. We have illustrated this with random sampling techniques for the CPD, and with mode-n vector sampling techniques originating from scientific computing for the LMLRA and the TT decomposition.

Part II

Algorithms

Canonical polyadic decomposition of incomplete tensors with linearly constrained factors

4

ABSTRACT | Incomplete datasets are prevalent in domains like machine learning, signal processing and scientific computing. Simple models often allow missing data to be predicted or can be used as representations for full datasets to reduce the computational cost. When data are explicitly or implicitly represented as tensors, which are multiway arrays of numerical values, a low-rank assumption is often appropriate. A nonlinear least squares algorithm to decompose an incomplete tensor into a sum of rank-1 terms is presented here. The combination of second-order information, a novel statistical preconditioner and the careful exploitation of the incompleteness and all available structure are of critical importance to achieve a highly performing algorithm. We show that the algorithm allows faster and more accurate decompositions compared to state-of-the-art methods when few entries are known or when missing entries are structured. Furthermore, the algorithm allows the incorporation of prior knowledge in the form of linearly constrained factor matrices. By imposing these linear constraints, the accuracy is improved further while fewer known entries are required. A specialized variant of the algorithm removes the data dependence from the optimization step by utilizing a compressed representation. When relatively many entries are known, computation time can be reduced significantly using this variant, as illustrated in a novel materials science application.

This chapter is based on N. Vervliet, O. Debals, and L. De Lathauwer, "Canonical polyadic decomposition of incomplete tensors with linearly constrained factors", Technical Report 16–172, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017.

4.1 Introduction

Tensors, or multiway arrays of numerical values, are a higher-order generalization of vectors and matrices and can be found in a variety of applications in, e.g., signal separation, object recognition, pattern detection, numerical simulations, function approximation and uncertainty quantification. Many more examples can be found in numerous overview papers [65], [126], [129], [170], [243]. As the number of entries in a tensor scales exponentially in its order, tensor problems become large-scale quickly. The computational challenges related to this exponential increase are known as the curse of dimensionality. To extract information or to efficiently represent tensors, low rank assumptions are often appropriate. The (canonical) polyadic decomposition (CPD), which is also known as PARAFAC, CANDECOMP, tensor rank decomposition or separated representation, writes an Nth-order tensor as a (minimal) number of R rank-1 terms, each of which is the outer product of N nonzero vectors, with R the tensor rank. Mathematically, we have

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} \stackrel{\text{def}}{=} \left[\!\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right]$$

where $\otimes$ denotes the outer product and $\mathbf{a}_r^{(n)}$ is the rth column of factor matrix $\mathbf{A}^{(n)}$, for $r = 1, \ldots, R$ and $n = 1, \ldots, N$. Many applications benefit from the mild uniqueness properties of a CPD, allowing components to be separated using few restrictions [65], [170], [243]. In contrast to the number of entries, the number of parameters in a CPD scales linearly in the order, allowing a compact representation of higher-order data. To handle large-scale problems, various approaches have been proposed including techniques based on compression [245], randomization [219], [300] and sparsity combined with parallel architectures [150], [251].

Measuring, computing and/or storing all entries is often expensive in terms of time, memory or cost. For example, in high-dimensional integration, the number of points easily exceeds the number of atoms in the universe [217]; in uncertainty quantification, each entry requires the solution of a PDE [172]; or in multivariate regression in many variables it may be impossible to store all possible values of a discretized function [27]. Reducing the number of entries by deliberately taking few samples is therefore important or even necessary to keep the problem manageable [304]. Also, in the case of tensor completion[1], some entries can be missing or unknown due to, e.g., broken sensors, artifacts which have been removed or physically impossible situations. Whether few entries are sampled or some entries are missing, specialized algorithms, e.g., [6], [157], [262], [279], [304], are key to decompose and analyze the resulting incomplete tensors. It is often important that the creation of the full tensor

[1] In this case a Tucker model is often used as well; see, e.g., [110], [188], [246], [247].

is avoided, especially for large-scale tensors.

Prior knowledge is often available, allowing the construction of more accurate and/or interpretable models. Moreover, additional constraints are important to further lower the number of required samples or to improve the accuracy. In this chapter, we focus on prior knowledge that can be represented as linear constraints on the factor matrices of a CPD, i.e., $\mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)}$ in which $\mathbf{B}^{(n)}$ is a known matrix, while $\mathbf{C}^{(n)}$ contains the unknown coefficients. This model is also known as CANDELINC [57] and is ubiquitous although it is not always recognized as a constrained CPD. For example, in multivariate regression with polynomials or sparse grids, $\mathbf{B}^{(n)}$ can be a basis of polynomials or multilevel tent functions [27], [111]. Some applications benefit from other types of bases like Gaussian kernels or trigonometric functions [311]. Another example can be found in compressed sensing (CS) where a compressed measurement $\mathbf{b}$ is represented in a dictionary $\mathbf{D}$ by a sparse coefficient vector $\mathbf{x}$ [54], [101]. If $\mathbf{D}$ is expressed as a Kronecker product and $\mathbf{x}$ is a (possibly sparse) vectorized CPD [52], [244], the CS problem can be written as a linearly constrained CPD.

Computing a linearly constrained CPD of a full tensor is straightforward as the tensor can be compressed using the known matrices $\mathbf{B}^{(n)}$ after which the matrices $\mathbf{C}^{(n)}$ can be found from the unconstrained CPD of the compressed tensor. Because of its efficiency, compression using suitable basis matrices [12] or the Tucker decomposition/MLSVD [46], [166] is a commonly used preprocessing step. For incomplete tensors, only algorithms for unconstrained decompositions are currently available. These algorithms can be divided into imputation or expectation-maximization algorithms which replace missing entries with some value, and direct minimization techniques which take only known entries into account. Examples for the former can be found in [138], [220], [279], while CCD++ [157], CP WOPT [6], INDAFAC [279] and CPD [262] are examples of the latter. The structured data fusion (SDF) framework from [262] can be used to model linear constraints, but it does not fully exploit the inherent multilinear structure. Dedicated algorithms have been presented for specific problems, e.g., in [27] an alternating least squares (ALS) algorithm is presented for multivariate regression problems. Here, we focus on the direct minimization techniques, as the memory requirements often prohibit the construction of the imputed tensor.

The efficient computation of a linearly constrained CPD of an incomplete tensor is our main focus. Two variants are presented in section 4.2: a data-dependent algorithm CPDLI DD which has a per-iteration complexity linear in the number of known entries and a data-independent algorithm CPDLI DI which has a per-iteration complexity depending only on the number of variables in the decomposition and not on the tensor entries. The presented algorithms are of the (inexact) Gauss–Newton type (see subsection 4.1.2); quasi-Newton (qN) or ALS algorithms can be derived easily. After fixing the notation and reviewing the used optimization framework in the remainder of

this section, the algorithms and their complexity are derived in section 4.2. The unconstrained CPD of an incomplete tensor is discussed as a special case in section 4.3. A new statistical preconditioner is discussed in section 4.4. Section 4.5 and section 4.6 illustrate the performance of the presented algorithms on synthetic data and a materials science example, respectively. To keep the main text and idea clear, the derivations of the algorithms and preconditioner are discussed in the supplementary material accompanying this chapter.

4.1.1 Notation

Scalars, vectors, matrices and tensors are denoted by lower case, e.g., a, bold lower case, e.g., $\mathbf{a}$, bold upper case, e.g., $\mathbf{A}$ and calligraphic, e.g., $\mathcal{T}$, letters, respectively. $\mathbb{K}$ denotes either $\mathbb{R}$ or $\mathbb{C}$. Sets are indexed by superscripts within parentheses, e.g., $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. An incomplete tensor $\mathcal{T}$ has N_{ke} known entries at positions $(i_k^{(1)}, i_k^{(2)}, \ldots, i_k^{(N)})$, $k = 1, \ldots, N_{\text{ke}}$. We collect all indices of known entries in index vectors $\mathbf{i}^{(1)}, \ldots, \mathbf{i}^{(N)}$. The extended form of a matrix $\mathbf{A}^{(n)} \in \mathbb{K}^{I_n \times R}$ is denoted by $\mathbf{A}^{[n]} = \mathbf{A}^{(n)}(\mathbf{i}^{(n)}, :)$ using MATLAB style indexing. $\mathbf{A}^{[n]} \in \mathbb{K}^{N_{\text{ke}} \times R}$ contains repeated rows as $1 \leq i_k^{(n)} \leq I_n$ and $N_{\text{ke}} \geq I_n$ and is used to facilitate the derivation of the algorithms. A mode-n vector is the generalization of a column (mode-1) and row (mode-2) vector and is defined by fixing all but the nth index of an Nth-order tensor. The mode-n unfolding of a tensor $\mathcal{T}$ is denoted by $\mathbf{T}_{(n)}$ and has the mode-n vectors as its columns. The vectorization operator $\text{vec}(\mathcal{T})$ stacks all mode-1 vectors into a column vector. Details on the order of the mode-n vectors in unfoldings and vectorizations can be found in [65]. A number of products are needed. The mode-n tensor-matrix product is denoted by $\mathcal{T} \cdot_n \mathbf{A}$. The Kronecker and Hadamard, or element-wise, product are denoted by $\otimes$ and $*$, respectively. The column-wise (row-wise) Khatri–Rao products between two matrices, denoted by $\odot$ ($\odot^{\mathrm{T}}$), is defined as the column-wise (row-wise) Kronecker product. To simplify expressions, the following shorthand notations are used

$$\bigotimes_{n} \equiv \bigotimes_{n=N}^{1} \qquad \bigotimes_{k \neq n} \equiv \bigotimes_{\substack{k=N \\ k \neq n}}^{1},$$

where N is the order of the tensor and indices are traversed in reverse order. Similar definitions are used for $\odot$, $\odot^{\mathrm{T}}$ and $*$. To reduce the number of parentheses needed, we assume the matrix product takes precedence over the Kronecker, (row-wise) Khatri–Rao and Hadamard products, e.g.,

$$\mathbf{AB} \odot \mathbf{CD} \equiv (\mathbf{AB}) \odot (\mathbf{CD}).$$

Similarly, shorthand notations are expanded first, e.g., for $N = 3$, we have

$$\bigodot_{k\neq 2} \mathbf{A}^{(k)} \odot \mathbf{A}^{(2)} \equiv \left(\mathbf{A}^{(3)} \odot \mathbf{A}^{(1)}\right) \odot \mathbf{A}^{(2)},$$
$$\bigodot_{k\neq 2} \mathbf{B}^{(k)}\mathbf{C}^{(k)} \otimes \mathbf{A}^{(2)} \equiv \left((\mathbf{B}^{(3)}\mathbf{C}^{(3)}) \odot (\mathbf{B}^{(1)}\mathbf{C}^{(1)})\right) \otimes \mathbf{A}^{(2)}.$$

The complex conjugate, transpose, conjugated transpose and pseudoinverse are denoted by $\bar{\cdot}$, $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$ and $\cdot^{\dagger}$, respectively. The column-wise concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{x} = [\mathbf{a}; \mathbf{b}]$ and is a shorthand for $\mathbf{x} = \begin{bmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. The inner product between $\mathcal{A}$ and $\mathcal{B}$ is denoted by $\langle \mathcal{A}, \mathcal{B} \rangle = \mathrm{vec}(\mathcal{B})^{\mathrm{H}} \mathrm{vec}(\mathcal{A})$. The weighted inner product between $\mathcal{A}$ and $\mathcal{B}$ is defined as $\langle \mathcal{A}, \mathcal{B} \rangle_{\mathcal{S}} = \mathrm{vec}(\mathcal{B})^{\mathrm{H}} \mathrm{diag}(\mathrm{vec}(\mathcal{S}))\mathrm{vec}(\mathcal{A})$ where weight tensor $\mathcal{S}$ has the same dimensions as $\mathcal{A}$ and $\mathcal{B}$. The Frobenius norm is denoted by $||\cdot||$. The real part of a complex number is written as $\mathrm{Re}(\cdot)$ and the expectation operator as $\mathbb{E}[\cdot]$. The permutation matrix $\mathbf{P}^{(n)}$ maps entry $(i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N)$ of a vectorized tensor to entry $(i_n, i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$. Details on the used notation can be found in [65].

4.1.2 Optimization framework

We discuss the expressions required to implement quasi-Newton (qN) and nonlinear least squares (NLS) algorithms to solve the optimization problem

$$\min_{\mathbf{z}} f = \frac{1}{2} ||F(\mathbf{z})||^2. \tag{4.1}$$

All presented algorithms allow complex tensors and/or factors. We use the complex optimization framework from [258], [259] which implements a large number of qN and NLS algorithms including implementations of various line search, plane search and trust region strategies. This framework requires functions computing the objective function, the gradient and, in the case of NLS algorithms, the Gramian or Gramian-vector product. In the remainder of this subsection, we elaborate on a particular NLS algorithm called Gauss–Newton. A more detailed explanation of other qN and NLS algorithms and complex optimization can be found in [8], [209], [258], [260].

Unconstrained NLS algorithms attempt to find optimal values $\mathbf{z}^* \in \mathbb{K}^M$ using objective function (4.1). In iteration $k+1$, the residual $F(\mathbf{z})$ is linearized using the first-order Taylor expansion at the current iterate $\mathbf{z}_k$:

$$\begin{aligned} F(\mathbf{z}) &\approx F(\mathbf{z}_k) + \left.\frac{\partial F}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_k} (\mathbf{z} - \mathbf{z}_k) \\ &= F(\mathbf{z}_k) + \mathbf{J}\mathbf{p}, \end{aligned}$$

in which $\mathbf{J}$ is the Jacobian of $F(\mathbf{z})$ and $\mathbf{p}$ the step direction. Substituting the

linearization in objective function (4.1) leads to a quadratic model

$$\min_{\mathbf{p}} \frac{1}{2} F(\mathbf{z}_k)^{\mathrm{H}} F(\mathbf{z}_k) + F(\mathbf{z}_k)^{\mathrm{H}} \mathbf{J}\mathbf{p} + \frac{1}{2} \mathbf{p}^{\mathrm{H}} \mathbf{J}^{\mathrm{H}} \mathbf{J}\mathbf{p}.$$

The first-order optimality criterion leads to the minimizer $\mathbf{p}^*$ by solving

$$\mathbf{J}^{\mathrm{H}} \mathbf{J}\mathbf{p}^* = -\mathbf{J}^{\mathrm{H}} F(\mathbf{z}_k). \tag{4.2}$$

Let $\mathbf{H} = \mathbf{J}^{\mathrm{H}}\mathbf{J}$ be the Gramian (of the Jacobian) which approximates the Hessian of f, and $\bar{\mathbf{g}}$ be the conjugated gradient, i.e., $\bar{\mathbf{g}} = \left(\frac{\mathrm{d}f}{\mathrm{d}\mathbf{z}}\right)^{\mathrm{H}} = \mathbf{J}^{\mathrm{H}} F(\mathbf{z}_k)$, then (4.2) becomes

$$\mathbf{H}\mathbf{p}^* = -\bar{\mathbf{g}}. \tag{4.3}$$

Note that (4.3) only requires the derivative w.r.t. $\mathbf{z}$, even for complex variables $\mathbf{z}$ while f is nonanalytic. As shown in [258], the gradient actually consists of the partial derivative w.r.t. $\mathbf{z}$ and $\bar{\mathbf{z}}$, keeping $\bar{\mathbf{z}}$ and $\mathbf{z}$ constant, respectively. Both partial derivatives of f are each other's complex conjugate and the Jacobian $\partial F / \partial \bar{\mathbf{z}} = 0$, as F is analytic in $\mathbf{z}$.

When the number of variables M is large, the computational cost of computing the (pseudo) inverse of $\mathbf{H} \in \mathbb{C}^{M \times M}$ can be a bottleneck. To reduce the complexity, (preconditioned) conjugate gradients (CG) can be used to solve (4.3) [209]. CG iteratively refines an initial guess $\mathbf{p}_0$ using only Gramian-vector products $\mathbf{G}\mathbf{p}_l$, $l = 0, \ldots, M-1$. In the case of CPD computations, [260] showed that preconditioned CG can effectively reduce the computational cost for large-scale systems.

Finally, the variables $\mathbf{z}$ are updated as

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

in which α_{k+1} is computed using a line search algorithm. Other variants use plane search or trust-region methods to compute $\mathbf{z}_{k+1}$. For more information on different updating methods, we refer the interested reader to, e.g., [209]. In this chapter, we use a dogleg trust region (DLTR) approach [209], [260].

It is clear that an NLS-based algorithm requires a way of computing the objective function, the gradient and the Gramian or Gramian-vector products. In the latter case, an effective preconditioner reduces the computational cost of the CG algorithm. In section 4.2, we derive these expressions for the linearly constrained CPD of incomplete tensors. An effective preconditioner is derived in section 4.4.

4.2 CPD with linearly constrained factors for incomplete tensors

The proposed CPDLI algorithms compute the rank-R CPD of an Nth-order incomplete tensor $\mathcal{T} \in \mathbb{K}^{I_1 \times \cdots \times I_N}$ with linear constraints on the factor matrices using an (inexact) Gauss–Newton algorithm with dogleg trust regions. Mathematically, this results in the following optimization problem:

$$\begin{aligned} \min_{\mathbf{C}^{(1)},\ldots,\mathbf{C}^{(N)}} \quad & f = \frac{1}{2} \left|\left| \mathcal{S} * \left(\left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] - \mathcal{T} \right) \right|\right|^2 \\ \text{subject to} \quad & \mathbf{A}^{(n)} = \mathbf{B}^{(n)} \mathbf{C}^{(n)} \quad n = 1, \ldots, N. \end{aligned} \tag{4.4}$$

An entry in the binary sampling tensor $\mathcal{S}$ is one if the corresponding entry in $\mathcal{T}$ is known and zero otherwise. The factor matrices $\mathbf{A}^{(n)} \in \mathbb{K}^{I_n \times R}$ are the product of known matrices $\mathbf{B}^{(n)} \in \mathbb{K}^{I_n \times D_n}$ and unknown coefficient matrices $\mathbf{C}^{(n)} \in \mathbb{K}^{D_n \times R}$, $n = 1, \ldots, N$.

Two variants of CPDLI are derived: the data-dependent algorithm uses the data in every iteration, while the data-independent variant computes a projected representation of the data prior to the optimization and uses this representation in every iteration. The derivations of all expressions are presented in Appendix B.1.

4.2.1 A data-dependent algorithm: CPDLI DD

The CPDLI DD algorithm computes the CPD with linearly constrained factor matrices of an incomplete tensor with a computational complexity linear in the number of known entries. This section summarizes the elements needed for the algorithm outlined in Algorithm 4.1.

Algorithm 4.1: CPDLI DD using Gauss–Newton with dogleg trust region.

1: **Input:** $\mathcal{S}$, $\mathcal{T}$, $\left\{\mathbf{B}^{(n)}\right\}_{n=1}^{N}$ and $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
2: **Output:** $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
3: **while** not converged **do**
4: Compute gradient $\mathbf{g}$ using (4.5) (or (4.6)).
5: Compute step $\mathbf{p}$ using either

- $\mathbf{p} = -\mathbf{H}^{\dagger}\bar{\mathbf{g}}$ with $\mathbf{H}$ from (4.8), or
- PCG of $\mathbf{H}\mathbf{p} = -\bar{\mathbf{g}}$ with Gramian-vector products $\mathbf{H}\mathbf{p}$ computed using (4.9), (4.11) and (4.12) and preconditioner (4.22).

6: Update $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, using DLTR from $\mathbf{p}$, $\mathbf{g}$ and function value (4.4).
7: **end while**

Objective function

Computing the function value is straightforward. After constructing the factor matrices $\mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, the sum of the squared entries of the residual tensor $\mathcal{F} = \mathcal{S} * \left(\llbracket \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \rrbracket - \mathcal{T}\right)$ is computed. For sparsely sampled tensor with $N_{\text{ke}} \ll I_1 I_2 \cdots I_N$, only the nonzero entries of $\mathcal{F}$ are constructed.

Gradient

The gradient $\mathbf{g}$ of (4.4) w.r.t. the (vectorized) optimization variables $\mathbf{C}^{(n)}$ is partitioned as $\mathbf{g} = \left[\text{vec}\left(\mathbf{G}^{(1)}\right); \ldots; \text{vec}\left(\mathbf{G}^{(N)}\right)\right]$ in which each subgradient $\mathbf{G}^{(n)}$ is given by

$$\mathbf{G}^{(n)} = \frac{\partial f}{\partial \mathbf{C}^{(n)}} = \mathbf{B}^{(n)^{\mathrm{T}}} \bar{\mathbf{F}}_{(n)} \left(\bigodot_{k \neq n} \mathbf{A}^{(k)} \right) \tag{4.5}$$

$$= \left(\bar{\mathcal{F}} \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \right), \tag{4.6}$$

which follows from the chain rule and from multilinear identities. Which one of the expressions (4.5) and (4.6) should be used depends on the order, rank and number of known entries in the tensor and D_n. Gradient expression (4.5) is computed using a sparse matricized tensor times Khatri–Rao product (MTKRPROD), which can be computed in $\mathcal{O}\left(2N^2 N_{\text{ke}} R\right)$ operations for the N subgradients $\mathbf{G}^{(n)}$. Gradient expression (4.6) computes a projection onto basis matrices $\mathbf{B}^{(n)}$, $n = 1, \ldots, N$, which is a series of N sparse tensor-matrix products, followed by a dense MTKRPROD. The former operation can also be seen as a matricized tensor times Kronecker product (MTKRONPROD). The overall complexity of (4.6) is therefore $\mathcal{O}\left(2NN_{\text{ke}} \prod_n D_n\right)$ for the (sparse) MTKRONPROD and $\mathcal{O}\left(2NR \prod_n D_n\right)$ for the dense MTKRPROD. Both MTKRPROD and MTKRONPROD are standard operations in tensor computations. Efficient implementations for dense and sparse tensors are described in, e.g., [16], [156], [224], [251], [293].

Gramian of the Jacobian

Finally, we compute the Gramian $\mathbf{H} = \mathbf{J}^{\mathrm{H}}\mathbf{J}$. Similar to the gradient, the Gramian is partitioned into a grid of $N \times N$ blocks $\mathbf{H}^{(m,n)}$, $m, n = 1, \ldots, N$, corresponding to the approximation of the second-order derivative w.r.t. $\mathbf{C}^{(m)}$ and $\mathbf{C}^{(n)}$. Each block is computed from the explicitly constructed Jacobians $\mathbf{J}^{(n)}$:

$$\mathbf{J}^{(n)} = \frac{\partial}{\partial \text{vec}\left(\mathbf{C}^{(n)}\right)} \text{vec}\left(\mathcal{F}\right) = \mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]} \tag{4.7}$$

in which

$$\mathbf{V}^{[n]} = \underset{k \neq n}{*} \mathbf{A}^{[n]}.$$

The Jacobian is therefore a dense matrix of size $N_{\text{ke}} \times D_n R$ and the Gramian blocks $\mathbf{H}^{(m,n)}$ of size $D_m R \times D_n R$ are given by

$$\mathbf{H}^{(m,n)} = \left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right). \tag{4.8}$$

To reduce the cost of computing the pseudoinverse of $\mathbf{H}$, CG is used when dealing with many variables. Each CG iteration requires the matrix-vector product $\mathbf{y} = \mathbf{H}\mathbf{x}$. A partitioning similar to the gradient and Gramian is used for the vectors $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{x} = \left[\operatorname{vec}\left(\mathbf{X}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{X}^{(N)}\right)\right],$$
$$\mathbf{y} = \left[\operatorname{vec}\left(\mathbf{Y}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{Y}^{(N)}\right)\right].$$

The part of the product $\mathbf{y} = \mathbf{H}\mathbf{x}$ corresponding to $\mathbf{Y}^{(m)}$ is then computed as

$$\begin{aligned}\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) &= \mathbf{J}^{(m)\mathrm{H}} \begin{bmatrix}\mathbf{J}^{(1)} & \cdots & \mathbf{J}^{(N)}\end{bmatrix} \left[\operatorname{vec}\left(\mathbf{X}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{X}^{(N)}\right)\right] \\ &= \mathbf{J}^{(m)\mathrm{H}} \sum_{n=1}^{N} \mathbf{J}^{(n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right).\end{aligned}$$

Using multilinear identities, the definition of the Jacobian (4.7) and the auxiliary vector $\mathbf{z}^{(n)}$, we then have

$$\mathbf{z}^{(n)} = \mathbf{J}^{(n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right) = \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]} \mathbf{X}^{(n)}\right) \mathbf{1} \tag{4.9}$$

$$\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) = \left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \sum_{n=1}^{N} \mathbf{z}^{(n)}. \tag{4.10}$$

The matrix $\mathbf{B}^{[n]}\mathbf{X}^{(n)}$ can contain many identical rows as $\mathbf{B}^{[n]}$ is the extended form of $\mathbf{B}^{(n)}$. The computational cost can therefore be reduced by extending the product $\mathbf{B}^{(n)}\mathbf{X}^{(n)}$ instead of $\mathbf{B}^{(n)}$. Using accumulation, the product $\left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \sum_{n=1}^{N} \mathbf{z}^{(n)}$ can be computed efficiently. Let $\mathbf{z} = \sum_{n=1}^{N} \mathbf{z}^{(n)}$ and define the ith row of $\mathbf{Q}^{(m)} \in \mathbb{K}^{I_m \times R}$ by

$$\mathbf{Q}^{(m)}(i,:) = \sum_{k \in \pi^{(m)}} z_k \bar{\mathbf{V}}^{[m]}(k,:) \quad \text{with} \quad \pi^{(m)} = \left\{ k \,|\, \mathbf{i}^{(m)}(k) = i \right\}. \tag{4.11}$$

The cost of the row-wise Khatri–Rao product in (4.10) is avoided by com-

puting

$$\mathbf{Y}^{(m)} = \mathbf{B}^{(m)^{\mathrm{H}}} \mathbf{Q}^{(m)}. \tag{4.12}$$

4.2.2 Removing data dependence: CPDLI DI

Each iteration, the CPDLI DD algorithm has a complexity dependent on the number of known entries N_{ke}. In this section, we show how the structure resulting from the linear constraints allows the per-iteration complexity to be independent of the number of known entries by computing a data-dependent matrix $\mathbf{D}$ prior to the optimization process. This matrix $\mathbf{D}$ is defined as the inner product

$$\mathbf{D} = \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{H}} \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right), \tag{4.13}$$

which only depends on the positions of the known entries and the given matrices $\mathbf{B}^{(n)}$. Blocking is used to lower the intermediate memory requirements for expanding $\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]}$. As the size of $\mathbf{D}$ is $\prod_n D_n \times \prod_n D_n$, it is clear that this approach is limited to tensors of low order N and small D_n as the number of entries in $\mathbf{D}$ scales as D^{2N} assuming $D_n = D$ for $n = 1, \ldots, N$.

In the remainder of this section, we show how $\mathbf{D}$ can be used to reduce the computational cost of the ingredients in the CPDLI algorithm. Algorithm 4.2 gives an overview of CPDLI DI.

Algorithm 4.2: CPDLI DI using Gauss–Newton with dogleg trust region.

1: **Input:** $\mathcal{S}$, $\mathcal{T}$, $\left\{\mathbf{B}^{(n)}\right\}_{n=1}^{N}$ and $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
2: **Output:** $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
3: Compute $\mathbf{D}$ from (4.13), $\mathcal{T}_B$ from (4.15), and $||\mathcal{S} * \mathcal{T}||^2$.
4: **while** not converged **do**
5: Compute gradient $\mathbf{g}$ using (4.17) and (4.16).
6: Compute step $\mathbf{p}$ using either
- $\mathbf{p} = -\mathbf{H}^{\dagger}\bar{\mathbf{g}}$ with $\mathbf{H}$ from (4.18), or
- PCG of $\mathbf{Hp} = -\bar{\mathbf{g}}$ with Gramian-vector products $\mathbf{Hp}$ computed using (4.19) and (4.20), and preconditioner (4.22).

7: Update $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, using DLTR from $\mathbf{p}$, $\mathbf{g}$ and function value (4.14).
8: **end while**

Objective function

The objective function (4.4) can be expanded as follows:

$$f = \frac{1}{2}\left|\left|\mathcal{S} * \mathcal{T}\right|\right|^2 - \mathrm{Re}\left(\left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right], \mathcal{T}\right\rangle_{\mathcal{S}}\right) + \frac{1}{2}\left|\left|\mathcal{S} * \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right]\right|\right|^2.$$

Let $\hat{\mathbf{t}}$ be the vector containing all values of known entries of $\mathcal{T}$. The norm $\left|\left|\mathcal{S} * \mathcal{T}\right|\right|^2$ is then computed prior to the optimization process as $\hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}}$. In Appendix B.1.3 we show that f can be computed as

$$f = \mathrm{Re}\left(\left(\frac{1}{2}\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right)^{\mathrm{H}}\mathbf{D} - \hat{\mathbf{t}}_B^{\mathrm{H}}\right)\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right)\right) + \frac{1}{2}\hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}}. \tag{4.14}$$

The vector $\hat{\mathbf{t}}_B$ is computed a priori and is defined as

$$\hat{\mathbf{t}}_B = \left(\bigodot_n^{\mathrm{T}}\mathbf{B}^{[n]}\right)^{\mathrm{H}}\hat{\mathbf{t}}. \tag{4.15}$$

While using (4.14) is computationally more efficient, it is numerically less accurate. Consider only the known entries $\hat{\mathbf{t}}$ and assume an approximation up to an error vector $\boldsymbol{\delta}$ with $||\boldsymbol{\delta}|| = \epsilon\left|\left|\hat{\mathbf{t}}\right|\right|$ has been found. The data-dependent objective function is then equivalent to

$$2f = \left|\left|(\hat{\mathbf{t}} + \boldsymbol{\delta}) - \hat{\mathbf{t}}\right|\right|^2 = ||\boldsymbol{\delta}||^2 = \mathcal{O}\left(\epsilon^2\right).$$

For the data-independent objective function we have

$$\begin{aligned} 2f &= \left|\left|(\hat{\mathbf{t}} + \boldsymbol{\delta})\right|\right|^2 - 2\mathrm{Re}\left((\hat{\mathbf{t}} + \boldsymbol{\delta})^{\mathrm{H}}\hat{\mathbf{t}}\right) + \left|\left|\hat{\mathbf{t}}\right|\right|^2 \\ &= \mathrm{Re}\left(\left(\left|\left|\hat{\mathbf{t}}\right|\right|^2 + 2\hat{\mathbf{t}}^{\mathrm{H}}\boldsymbol{\delta} + ||\boldsymbol{\delta}||^2\right) - 2\left(\left|\left|\hat{\mathbf{t}}\right|\right|^2 + \hat{\mathbf{t}}^{\mathrm{H}}\boldsymbol{\delta}\right) + \left|\left|\hat{\mathbf{t}}\right|\right|^2\right). \end{aligned}$$

If $\epsilon < \sqrt{\epsilon_{\mathrm{mach}}}$ with ϵ_{mach} the machine precision, $||\boldsymbol{\delta}||^2$ is numerically zero, and $2f = 0$ or $2f = \pm\mathcal{O}\left(\epsilon_{\mathrm{mach}}\right)$ instead of $\mathcal{O}\left(\epsilon_{\mathrm{mach}}^2\right)$. This limits the accuracy to $\mathcal{O}\left(\sqrt{\epsilon_{\mathrm{mach}}}\right)$. In applications, however, other factors often put more stringent restrictions to the accuracy. Due to the signal-to-noise ratio, for example, only a few accurate digits are required in signal processing, or, in machine learning and data analysis, a limited precision is expected because of model errors and regularization terms. In both examples, the limited accuracy caused by using (4.14) is therefore not a problem. If, in a specific application, a solution accurate up to machine precision is required, the solution found

by the data-independent algorithm can be refined using a few iterations of the data-dependent algorithm.

Gradient

Similar to the derivation of the objective function, the residual tensor $\mathcal{F}$ in (4.6) is separated in a data part and a decomposition part. As shown in Appendix B.1.3, the gradient can be written as

$$\mathbf{G}^{(n)} = \left(\bar{\mathcal{Z}} - \bar{\mathcal{T}}_B\right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right), \tag{4.16}$$

which is an MTKRPROD involving the following compressed, dense tensors. The values of compressed data representation $\bar{\mathcal{T}}_B = \left(\mathcal{S} * \bar{\mathcal{T}}\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}$ are $\hat{\mathbf{t}}_B$ from (4.15). The conjugated auxiliary tensor $\bar{\mathcal{Z}}$ has size $D_1 \times \cdots \times D_N$ and is computed as

$$\begin{aligned} \bar{\mathcal{Z}} &= \left(\mathcal{S} * \left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)} \right]\!\right]\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \\ &= \text{reshape}\left(\bar{\mathbf{D}} \text{vec}\left(\left[\!\left[\bar{\mathbf{C}}^{(1)}, \ldots, \bar{\mathbf{C}}^{(N)} \right]\!\right]\right), [D_1, \ldots, D_N]\right). \end{aligned} \tag{4.17}$$

Gramian of the Jacobian

The complexity of computing the data-dependent Gramian in (4.8) can be reduced by removing the dependency on $\mathbf{B}^{[n]}$. In Appendix B.1.3, we show that

$$\mathbf{H}^{(m,n)} = \left(\bigodot_{k \neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \mathbf{D} \mathbf{P}^{(n)^{\mathrm{T}}} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right). \tag{4.18}$$

This is a variation on the MTKRPROD called the Khatri–Rao times matricized tensor times Khatri–Rao product (KRMTKRPROD). While an MTKRPROD can be computed in memory and cache efficient way, i.e., without permuting entries of the tensor [224], [293], permutations are required for KRMTKRPROD. Using similar ideas as in [224], [293], the performance penalty endured by permuting the tensor can be minimized using appropriate left-to-right and right-to-left contractions. The exact description of this algorithm is out of the scope of this chapter. An implementation of KRMTKRPROD will be made available in Tensorlab [305].

Similar to the data-dependent algorithm, CG can be used for large-scale problems, hence, requiring a Gramian-vector product. From (4.18) we have for a single Gramian block $\mathbf{H}^{(m,n)}$

$$\mathbf{H}^{(m,n)} \text{vec}\left(\mathbf{X}^{(n)}\right) = \left(\bigodot_{k \neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \text{vec}\left(\mathcal{Z}^{(n)}\right)$$

in which the vectorized tensor $\mathcal{Z}^{(n)}$ of size $D_1 \times \cdots \times D_N$ is given by

$$\begin{aligned} \operatorname{vec}\left(\mathcal{Z}^{(n)}\right) &= \mathbf{D}\operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right) \\ \operatorname{vec}\left(\mathcal{Z}\right) = \sum_{n=1}^{N} \operatorname{vec}\left(\mathcal{Z}^{(n)}\right) &= \mathbf{D}\sum_{n=1}^{N} \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right). \end{aligned} \tag{4.19}$$

The matricized result $\mathbf{Y}^{(m)}$ is then computed using an MTKRPROD:

$$\mathbf{Y}^{(m)} = \sum_{n=1}^{N} \mathbf{H}^{(m,n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right) = \mathbf{Z}_{(m)}\left(\bigodot_{k\neq m} \bar{\mathbf{C}}^{(k)}\right). \tag{4.20}$$

4.2.3 Complexity

To conclude this section, the per-iteration complexity of both variants is investigated and compared. For simplicity of notation, assume $\mathcal{T}$ is a cubical tensor, i.e., $I_n = I$ and matrices $\mathbf{B}^{(n)}$ have the same number of columns, i.e., $D_n = D$, for $n = 1, \ldots, N$. A summary of the per-iteration complexity for CPDLI DD and CPDLI DI is given in Table 4.1 and Table 4.2, respectively. For large-scale problems, i.e., problems with a large number of variables, the choice between CPDLI DD and CPDLI DI depends on N_{ke} and D: the former's overall complexity per iteration scales with $\mathcal{O}\left(N^2 R D N_{\text{ke}}\right)$ compared to $\mathcal{O}\left(D^{2N}\right)$ for the latter. Therefore, the CPDLI DI algorithm is favored for low-order tensors with D small and relatively many known entries.

Table 4.1: The per-iteration complexity of the CPDLI DD algorithm is dominated by Gramian operations. *The overall complexity depends linearly on the number of known entries N_{ke}. A trust region method is used to determine the update of the variables, requiring it_{TR} additional function evaluations.*

	Calls per iteration	Complexity
Function value	$1 + \text{it}_{\text{TR}}$	$\mathcal{O}\left(N_{\text{ke}} N R\right)$
Gradient (sparse)	1	$\mathcal{O}\left(2N^2 R N_{\text{ke}}\right)$
Gradient (dense)	1	$\mathcal{O}\left(2N D^N (R + N_{\text{ke}})\right)$
Gramian	1	$\mathcal{O}\left(N^2 D^2 R^2 N_{\text{ke}} + \frac{8}{3} N^3 D^3 R^3\right)$
Gramian-vector	it_{CG}	$\mathcal{O}\left(2N^2 D R N_{\text{ke}}\right)$
Total	Small-scale	$\mathcal{O}\left(N^2 D^2 R^2 N_{\text{ke}} + \frac{8}{3} N^3 D^3 R^3\right)$
	Large-scale	$\mathcal{O}\left(2\text{it}_{\text{CG}} N_{\text{ke}} N^2 R D\right)$

Table 4.2: The per-iteration complexity of the CPDLI DI algorithm is dominated by Gramian operations involving $\mathbf{D}$. *The number of known entries N_{ke} appears only in the preparation complexity which is independent of the number of iterations. A trust region method is used to determine the update of the variables, requiring it_{TR} additional function evaluations.*

	Calls per iteration	Complexity
Preparation		$\mathcal{O}\left(2ND^{2N}N_{\text{ke}}\right)$
Function value	$1 + \text{it}_{\text{TR}}$	$\mathcal{O}\left(2D^{2N}\right)$
Gradient	1	$\mathcal{O}\left(2D^{2N} + 3D^{N}RN\right)$
Gramian	1	$\mathcal{O}\left(N^2RD^{2N} + \frac{8}{3}N^3D^3R^3\right)$
Gramian-vector	it_{CG}	$\mathcal{O}\left(2D^{2N} + N^2D^{N}R\right)$
Total	Small-scale	$\mathcal{O}\left(N^2RD^{2N} + \frac{8}{3}N^3D^3R^3\right)$
	Large-scale	$\mathcal{O}\left(2(\text{it}_{\text{CG}} + 2)D^{2N}\right)$

4.3 Unconstrained CPD for incomplete tensors

How to deal with missing entries when computing a CPD has been an important topic for many years and many algorithms have been proposed. We present a new algorithm called CPDI, which is a special case of CPDLI DD. In contrast to imputation-based methods [138], [220], [279], the computational per-iteration complexity is linear in N_{ke}, and in contrast to other direct minimization techniques [6], [157], [262], second-order convergence can be achieved by using the exact Gramian. By combining fast convergence with efficient Gramian-vector products, CPDI outperforms state-of-the-art methods when few entries are known or the position of the missing entries follows a pattern, as shown in the experiments in subsection 4.5.1. These cases are particularly interesting when dealing with large-scale problems as illustrated in [304].

The unconstrained CPD of an incomplete tensor can be seen as a special case of the CPDLI algorithm in which the known matrices are identity matrices, i.e., $\mathbf{B}^{(n)} = \mathbf{I}_{I_n}$, for $n = 1, \ldots, N$. As $D_n = I_n$, the data independent variant CPDLI DI is usually not attractive as I_n is often large. Therefore, the CPDI algorithm is based on CPDLI DD. The expressions for the function value and the gradient can be derived trivially from CPDLI DI and result in the same expressions as in [6] which uses a quasi-Newton approach and [262], [279] which use a Levenberg–Marquardt or Gauss–Newton approach (as in this chapter). The main difference with [279] is that an inexact algorithm is used, meaning CG is used instead of inverting the Gramian. While the CPDI algorithm proposed here uses the exact Gramian, [262] computes the Gramian for a full tensor and then scales the result by the fraction of known entries. As this scaling is countered by a longer step length in the line search or trust region step, the incompleteness is in fact ignored.

In the experiments in subsection 4.5.1, we illustrate that the use of the

exact Gramian is beneficial if the known entries are not scattered uniformly at random across the tensor, or if high accuracy is required while few entries are available. As $\mathbf{B}^{(n)}$ is an identity matrix in the case of CPDI, the Jacobian (Equation (4.7)) is sparse. Exploiting this sparsity using sparse matrix-matrix or matrix-vector products reduces the computational complexity to $\mathcal{O}\left(N^2R^2N_{\text{ke}} + \frac{8}{3}N^3D^3R^3\right)$ flop for the small-scale version (first term is a factor D^2 smaller compared to CPDLI DD) and to $\mathcal{O}\left(2\text{it}_{\text{CG}}N^2RN_{\text{ke}}\right)$ for the large-scale version (a factor D smaller).

4.4 Preconditioner

For relatively large-scale tensors, the cost of inverting the Gramian matrix in (4.3) can be alleviated by using preconditioned CG which only requires matrix-vector products [209]. In PCG, the system

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

is solved instead, in which the preconditioner $\mathbf{M}$ is a symmetric positive definite matrix. As the convergence of CG depends on the clustering of the eigenvalues of $\mathbf{H}$, the number of CG iterations can be reduced if the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered. At the same time, applying the inverse of the preconditioner should be computationally cheap.

For CPD algorithms, [260] proposes a block-Jacobi type preconditioner

$$\mathbf{M}_{\text{FULL}} = \text{blkdiag}\left(\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(N)}\right),$$

where $\mathbf{M}^{(n)} = \mathbf{W}^{(n)} \otimes \mathbf{I}$ with $\mathbf{W}^{(n)} = \mathbf{V}^{(n)^{\text{H}}}\mathbf{V}^{(n)}$ which is computed efficiently as $*_{k\neq n}\,\mathbf{A}^{(n)^{\text{H}}}\mathbf{A}^{(n)}$. In [262], an extension to incomplete tensors uses a scaled preconditioner

$$\mathbf{M}_{\text{INC,SCALED}} = \rho\mathbf{M}_{\text{FULL}}. \tag{4.21}$$

The fraction $\rho = N_{\text{ke}}/\prod_n I_n$ is simply the fraction of known entries. This is a reasonable approximation for tensors with known entries scattered randomly across the tensor and with entries of similar magnitudes.

Here, we propose a similar block-Jacobi type preconditioner based on the expected value of the diagonal Gramian blocks $\mathbf{H}^{(n,n)}$. First, we derive a preconditioner for the CPDI algorithm, which we then extend to the CPDLI algorithm by incorporating linear constraints.

To precondition the linear systems arising when decomposing incomplete tensors using CPDI, we propose the block-Jacobi preconditioner $\mathbf{M}_{\text{INC,STAT}}$ with diagonal blocks $\mathbf{M}^{(n)}_{\text{INC,STAT}}$. The exact expression for the block diagonal of $\mathbf{H}^{(n)} = \mathbf{J}^{(n)^{\text{H}}}\mathbf{J}^{(n)}$ would require the inversion of a $I_nR \times I_nR$ matrix, which

is often too expensive for large-scale problems. Instead, the expected value of the block $\mathbf{H}^{(n,n)}$ is used. More concretely, consider the i_nth slice of order $N-1$ and assume there are $Q_{i_n}^{(n)}$ known entries in this slice. Instead of taking the exact positions, or distribution, of the known entries into account, we assume that all possible distributions of exactly $Q_{i_n}^{(n)}$ known entries in this slice are equally likely. By taking the expected value over all distributions, only the fraction of known entries $f_{i_n}^{(n)} = Q_{i_n}^{(n)} / \prod_{k \neq n} I_k$ is required to approximate the blocks on the diagonal by

$$\mathbf{M}_{\text{INC,STAT}}^{(n)} = \mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \operatorname{diag}\left(\mathbf{f}^{(n)}\right) \tag{4.22}$$

as shown in Appendix B.2. This expression reduces to the result from [262] if every slice contains the same number of known entries.

For both CPDLI algorithms, the preconditioner is extended to the case with linear constraints on the factor matrices, resulting in the expression

$$\mathbf{M}_{\text{LINC,STAT}}^{(n)} = \mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \left(\mathbf{B}^{(n)^{\mathrm{H}}} \operatorname{diag}\left(\mathbf{f}^{(n)}\right) \mathbf{B}^{(n)}\right). \tag{4.23}$$

Even though this preconditioner only uses an approximation of the exact Gramian blocks, which take the exact position of the known entries into account, the experiments in subsection 4.5.2 show that the number of CG iterations is effectively reduced.

4.5 Experiments

The performance of the CPDI method and the effectiveness of the proposed preconditioner are shown with two experiments in this section. Section 4.6 illustrates the benefits of linear constraints and second-order information on a materials science dataset using CPDLI. Details about the parameters and implementation used for the different algorithms are reported in Appendix B.3 to allow reproducibility.

4.5.1 CPD of incomplete tensors

The effectiveness of using second-order information to reduce the computational cost of incomplete tensor decompositions is illustrated for two scenarios: randomly missing entries and structured missing entries. The proposed CPDI algorithm is compared to another Gauss–Newton type algorithm CPD [260], [262], and to two quasi-Newton algorithms based on nonlinear conjugate gradients (NCG): CPD MINF [260], [262] and CP WOPT [6]. The first three algorithms are implemented in Tensorlab [305]. The Tensor Toolbox [17] implementation is used for CP WOPT. For all algorithms, we compute a low accuracy solution ($E_{\text{CPD}} \approx 10^{-5}$) and a high accuracy solution ($E_{\text{CPD}} \leq 10^{-12}$),

in which

$$E_{\text{CPD}} = \max_{n=1,\ldots,N} \frac{\left|\left|\mathbf{A}^{(n)} - \hat{\mathbf{A}}^{(n)}\right|\right|}{\left|\left|\mathbf{A}^{(n)}\right|\right|}$$

with $\mathbf{A}^{(n)}$ the (known) exact factor matrices and $\hat{\mathbf{A}}^{(n)}$ the estimated factor matrices after resolving the scaling and permutation indeterminacies. All experiments are repeated 25 times using different tensors and different initial guesses.

First, let $\mathcal{T} \in \mathbb{R}^{1000\times1000\times1000}$ be a rank-10 tensor generated using random factor matrices $\mathbf{A}^{(n)}$ drawn from a uniform distribution $\mathcal{U}(0,1)$[2]. From this tensor, entries are sampled uniformly and randomly. As determining the rank is an NP-hard problem, prior knowledge, cross validation or trial-and-error are often used in practice to estimate R. Here we always assume the exact rank is known. Figure 4.1 shows the computation time when 300 000 entries and 90 000 entries are sampled, corresponding to 0.03% and 0.009% of all entries, respectively, or alternatively, 10 and 3 entries on average per variable, respectively. Both in time and number of iterations, the Gauss–Newton-based methods CPDI and CPD outperform the NCG-based methods. Hence the higher per-iteration complexity is countered by the reduction in number of iterations. While CPD outperforms CPDI when many entries are known or when a low accuracy is needed, the number of iterations is higher for CPD compared to CPDI. For high accuracy and few known entries, CPDI benefits from its improved convergence properties. CP WOPT fails to find a high-accuracy solution due to its implementation of the objective function.

The second scenario starts from a dense rank-10 tensor $\mathcal{T} \in \mathbb{R}^{100\times100\times100}$ constructed using random factor matrices with entries drawn from $\mathcal{U}(0,1)$. A structured missing data pattern is constructed as follows. From $\mathcal{T}$, 95 mode-1 slices are selected uniformly at random and from each of the selected slices 99% of the entries are discarded at random. The remaining 5 mode-1 slices are dense. While the fraction of known entries is relatively high (5.95%), the pattern prevents CPD from finding a solution. Figure 4.2 shows the computational cost is again reduced significantly because of the improved convergence properties for CPDI.

4.5.2 Preconditioner

CPDI and CPDLI have a relatively high per-iteration complexity while only a few iterations are necessary. Reducing the complexity or the number of iterations has therefore a large impact on the overall computation time. Here, we show the effectiveness of the preconditioners proposed in section 4.4 in

[2] By selecting $\mathcal{U}(0,1)$ the factor matrices are relatively ill-conditioned as the expected angle between the factor vectors is 41 degrees.

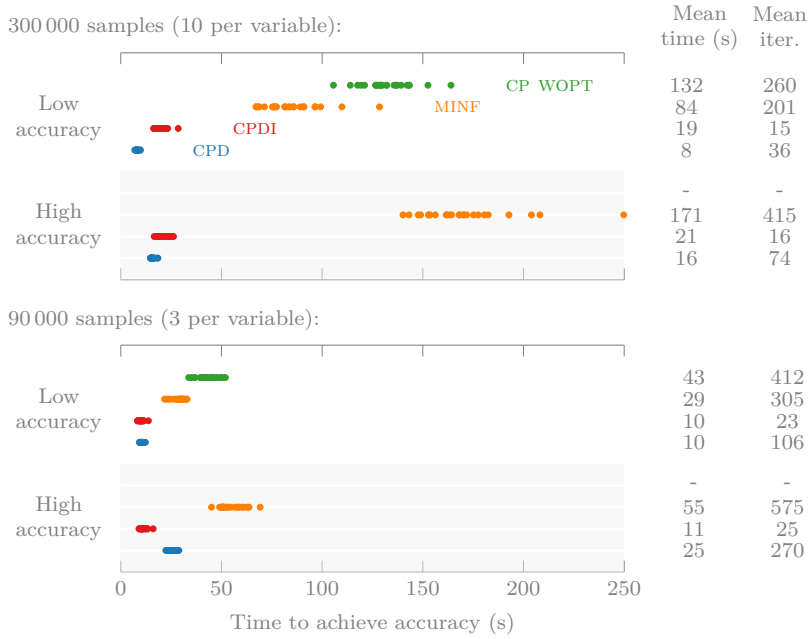


Figure 4.1: The Gauss–Newton type algorithms CPD and CPDI outperform the first-order NCG type algorithms as the higher per-iteration cost is countered by a significantly lower number of required iterations. *When many entries are available, the lower complexity of* CPD *compared to* CPDI *is an advantage. For higher accuracies or fewer entries,* CPDI *is competitive because of its faster convergence rate.* CP WOPT *fails to find a highly accurate solution. The reported timings and number of iterations are averages over 25 experiments.*

reducing the number of CG iterations and therefore in reducing the per-iteration complexity of the algorithm.

Consider the following experiment. A random tensor $\mathcal{T} \in \mathbb{R}^{100\times100\times100}$ is constructed using linearly constrained factor matrices with random basis matrices $\mathbf{B}^{(n)} \in \mathbb{R}^{100\times10}$ and coefficient matrices $\mathbf{C}^{(n)} \in \mathbb{R}^{10\times5}$. The entries of both $\mathbf{B}^{(n)}$ and $\mathbf{C}^{(n)}$ are drawn from a uniform distribution $\mathcal{U}(0,1)$ with $n = 1, 2, 3$. To simulate a typical iteration, the Gramian $\mathbf{H}$ and the gradient $\mathbf{g}$ are computed for random coefficients $\hat{\mathbf{C}}^{(n)}$ and the system $\mathbf{H}\mathbf{x} = -\bar{\mathbf{g}}$ is solved using PCG until convergence, i.e., until the relative residual is smaller than 10^{-6}. The number of CG iterations is monitored when using no preconditioner, the incompleteness agnostic preconditioner (4.21) from [262] and the proposed preconditioner (4.23). As in subsection 4.5.1, two scenarios are considered: 0.1% randomly sampled entries, and structured missing en-

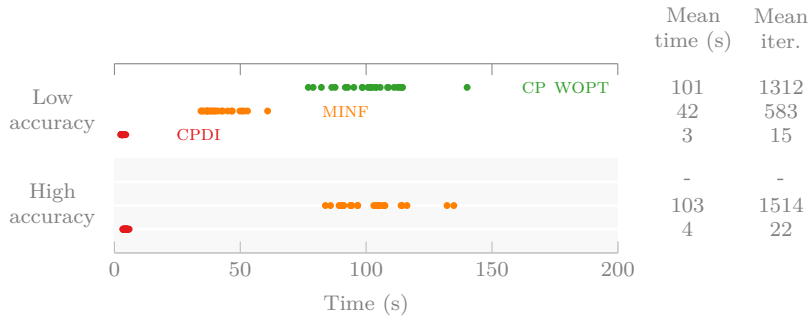


Figure 4.2: When missing entries follow a structured pattern, CPDI needs only few iterations to find a solution. CPD *fails to find a solution in a reasonable amount of time for both accuracy levels and is therefore omitted.* CP WOPT *fails to find a highly accurate solution. The reported timings and number of iterations are averages over 25 experiments.*

tries constructed by removing 99% of the entries of 95 randomly selected mode-1 slices. Table 4.3 shows the results averaged over 25 tensors and 20 random coefficient matrices $\hat{\mathbf{C}}^{(n)}$. While both preconditioners have a similar performance for randomly missing entries, the proposed preconditioner outperforms the scaled preconditioner for structured missing entries. Both preconditioners reduce the number of CG iterations significantly compared to the nonpreconditioned case.

Table 4.3: The proposed PC and the incompleteness agnostic PC from [262] reduce the number of CG iterations significantly compared to the nonpreconditioned case. *When missing entries are scattered randomly across the tensor, both preconditioners have a similar performance, as expected. When the missing entries are structured, the proposed PC reduces the number of iterations by 14 on average. The average (standard deviation) is computed over 25 tensors with dimensions* $100 \times 100 \times 100$ *for 20 sets of variables, each.*

Scenario	No PC	Agnostic PC (4.21)	Proposed PC (4.23)
Uniformly at random	115.8 (19.7)	45.5 (9.3)	45.1 (9.2)
Structured missing	124.6 (18.4)	53.3 (10.0)	39.3 (7.3)

4.6 Materials science application

Designing new materials often requires extensive numerical simulations depending on material parameters. For example, the melting temperature of an alloy depends, among others, on the concentrations of its constituent materials [304]. As a material property can depend on a large number of

variables, e.g., concentrations, temperature and pressure, the corresponding tensors with measured or computed values often have a high order. The curse of dimensionality therefore prohibits measuring all possible combinations. In [304], the CPD of an incomplete tensor allowed modeling a ninth-order tensor using only $10^{-11}\%$ of the data. Here, we show that using incomplete tensors and a CPD with linearly constrained factors allows the Gibbs free energy to be modeled using very few data points, while improving the accuracy w.r.t. the unconstrained problem.

Concretely, the Gibbs free energy for an alloy with silver, copper, nickel and tin as constituent materials is modeled in its liquid phase, at a temperature $T = 1000\,\mathrm{K}$. All parameters other than the concentrations (expressed in mole fraction) of silver (c_1), copper (c_2) and nickel (c_3) are kept constant. The concentration of tin (c_4) is a dependent variable as all concentrations must sum to 1, hence $c_4 = 1 - c_1 - c_2 - c_3$. Following the Calphad model [177], [194], the Gibbs free energy can be modeled as

$$G(c_1, c_2, c_3) = G_p(c_1, c_2, c_3) + G_{\log}(c_1, c_2, c_3)$$
$$G_{\log}(c_1, c_2, c_3) = \sum_{n=1}^{4} UTc_n \log_{10} c_n,$$

in which U is the universal gas constant. Thermo-Calc software [275] with the COST531-v3 database [178] computes G in chosen concentrations c_1, c_2 and c_3. By discretizing the concentrations, an incomplete third-order tensor $\mathcal{G}$ is obtained with the concentrations as its modes. As the logarithmic terms in $\mathcal{G}_{\log}$ depend solely on known values, we subtract these terms from $\mathcal{G}$ and model only $\mathcal{G}_p$ as a multivariate polynomial. Note that $\mathcal{G}$ is necessarily incomplete: if sampled densely, i.e., if all feasible entries are sampled on the chosen discretized grid of concentrations, all entries lie inside a pyramid (Figure 4.3) as $c_1+c_2+c_3 > 1$ is physically unfeasible. Consequently, variables at the vertices of the pyramid are estimated using only a single data point.

Five algorithms are compared in this section:

- the data-dependent (CPDLI DD) and data-independent (CPDLI DI) algorithms proposed in this chapter,
- multivariate regression using alternating least squares (MVR ALS) [27],
- structured data fusion (SDF) using `struct_matvec` to implement the constraints [262] and the `cpdi` factorization which implements CPDI as proposed here [305], and
- CPDI, i.e., without linear constraints.

The precise parameters are outlined in Appendix B.3. All tested algorithms behave differently and achieve a different accuracy, defined as the median

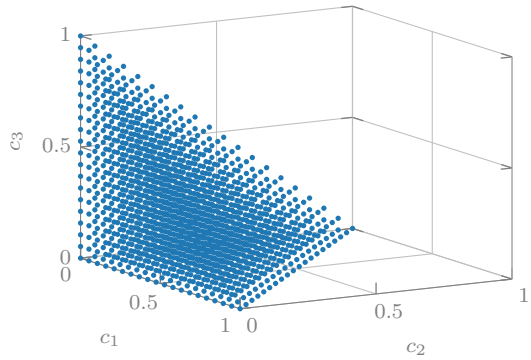


Figure 4.3: A 'dense' sampling scheme for $\mathcal{G}$ results in pyramidal structure as $c_1 + c_2 + c_3 \leq 1$.

relative error over all feasible entries of the discretized Gibbs free energy tensor:

$$E = \text{median} \left| \frac{\mathcal{G}_p(i,j,k) - \sum_{r=1}^{R} a_{ir}^{(1)} a_{jr}^{(2)} a_{kr}^{(3)}}{\mathcal{G}_p(i,j,k)} \right|.$$

As the accuracy of the final solutions computed by the different algorithms can differ, accuracy thresholds are used to allow for a fair comparison of timing results. Five levels with accuracy thresholds are set: $10^{-3}, 5{\cdot}10^{-4}, 10^{-4}, 5{\cdot}10^{-5}, 10^{-5}$. The algorithm proceeds to the next level when the median relative error E is below the current level's threshold and the (cumulative) time for each level is recorded. A maximum computation time of three minutes is set for each level. This is repeated for 50 initial guesses for $\mathbf{C}^{(n)}$, $n = 1, 2, 3$.

In the first experiment, the entries of $\mathcal{G}_p \in \mathbb{R}^{99 \times 99 \times 99}$ are sampled according to a dense, pyramidal grid with $c_n = 0.005, 0.015, 0.025, \ldots, 0.985$ for $n = 1, 2, 3$ which amounts to 166 338 known entries (17% of all entries). A CPD with $R = 5$ terms is computed from $\mathcal{G}_p$ with polynomial constraints on the factor matrices, i.e., each factor vector is a polynomial of maximal degree $d = 4$, hence $D_n = 5$, $n = 1, 2, 3$. A monomial basis is used. Figure 4.4 shows that both CPDLI algorithms find a high accuracy approximation quickly. The computation time for the data-independent algorithm CPDLI DI is almost a factor ten lower, as the number of known entries is high compared to the number of variables. The accuracy of the solution achieved by the other algorithms is lower, while the time needed to find a comparable accuracy is higher. Table 4.4 shows that the CPDLI algorithms result in a more accurate solution proving the usefulness of second-order information. Compared to an unconstrained model, computed by CPDI, the linear constraints improve accuracy more than an order of magnitude. Note that we did not report values

for the CPD method [262], as it failed to reach the first accuracy level, illustrating again that its Gramian approximation does not suffice for structured missing entries.

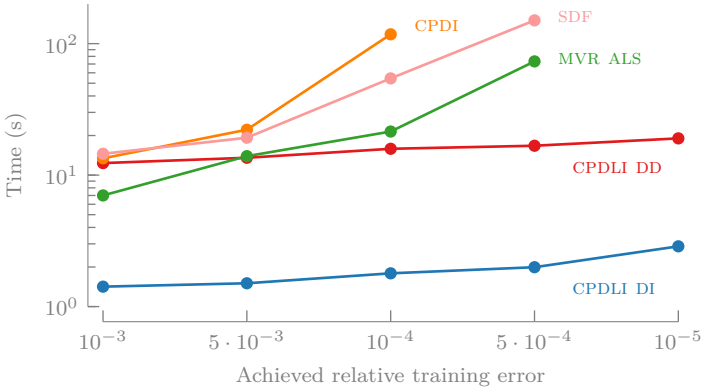


Figure 4.4: Comparison of the time needed to achieve a given accuracy or training error. *Because of the second-order information,* CPDLI *requires few iterations to improve accuracy once a reasonable accurate solution is found which results in a low computation time. As many samples are known, the data-independent algorithm* CPDLI DI *reduces the computation time by a factor 10 compared to* CPDLI DD. MVR ALS *quickly finds a solution, but requires a large number of iterations to improve this solution.* SDF *is a general purpose solver which does not exploit all available structure.* CPDI *does not find a high accuracy solution as it ignores the linear constraints. The median time over 50 experiments is reported.*

In the second experiment, we sample 200 entries at random from $\mathcal{G}_p$ and test CPDLI DD and the MVR ALS algorithm using the same parameters as in the first experiment. All errors are computed for all feasible entries, i.e., E can be seen as the validation error. Again, the CPDLI method achieves a high accuracy as can be seen in Table 4.5. While being slower for low accuracy solutions, CPDLI is more than an order of magnitude faster for high accuracies as can be seen in Figure 4.5. The speedup is caused mainly by the low number of iterations needed.

4.7 Conclusion

Prior knowledge can be incorporated in a CPD using linear constraints, and enables significant reductions in computation time while improving accuracy, as illustrated using experiments. In the case of incomplete tensors, careful exploitation of Kronecker products allows efficient implementations of nonlinear least squares type algorithms CPDLI DD and CPDLI DI. For a large number of known entries and a relatively low number of variables, a projected version of the data removes the dependency on the number of known entries

Table 4.4: The results computed by CPDLI algorithms are almost an order of magnitude more accurate than those computed by MVR ALS and SDF in terms of median error and maximal error. *Using linear constraints improves the achievable accuracy as the* CPDI *algorithm consistently returns the least accurate solution. The point-wise relative error on the training set is reported and is the median over 50 experiments.*

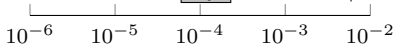
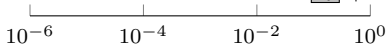
	Rel. error	Median rel. error E
CPDLI DI		$6.91 \cdot 10^{-6}$
CPDLI DD		$7.03 \cdot 10^{-6}$
MVR ALS		$5.29 \cdot 10^{-5}$
SDF		$3.49 \cdot 10^{-5}$
CPDI		$1.28 \cdot 10^{-4}$
		

Table 4.5: CPDLI is more than an order of magnitude more accurate than MVR ALS in terms of relative error on the validation data, when 200 samples are used for training (experiment 2). *The median of the relative error reported here are computed for a rank $R = 6$ and degree $d = 4$ multivariate polynomial model ($D_n = 5$). As a baseline, the accuracy when the mean value is used as predictor is shown as well.*

	Rel. error	Median rel. error E
CPDLI DD		$6.33 \cdot 10^{-6}$
MVR ALS		$2.69 \cdot 10^{-5}$
Mean		$1.84 \cdot 10^{-1}$
		

from the per-iteration computational complexity. For large-scale problems, the overall computational cost is reduced significantly using a cheap yet effective statistical block-Jacobi type preconditioner which uses the expected value of the Gramian. As a special case, the CPDI algorithm for the unconstrained CPD is discussed which benefits from better approximations to the second-order information, especially for difficult problems with structured missing entries. The effectiveness and efficiency of the presented algorithms are shown in a novel materials science application. In this application, the CPDLI algorithms improved the accuracy by almost an order of magnitude, while having a significantly lower computational cost.

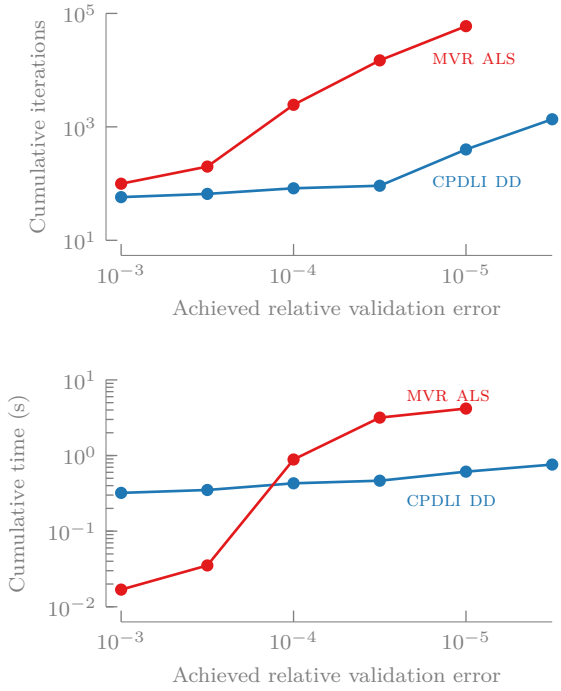


Figure 4.5: CPDLI finds a high accuracy solution quickly even though MVR ALS is faster in achieving a low accuracy solution. *To find a high accuracy solution, the* CPDLI *algorithm benefits from its better convergence properties which translates into a low number of required iterations. The reported results are averages over 50 random initializations.*

A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors

5

ABSTRACT | For the analysis of large-scale datasets one often assumes simple structures. In the case of tensors, a decomposition in a sum of rank-1 terms provides a compact and informative model. Finding this decomposition is intrinsically more difficult than its matrix counterpart. Moreover, for large-scale tensors, computational difficulties arise due to the curse of dimensionality. The randomized block sampling canonical polyadic decomposition method presented here combines increasingly popular ideas from randomization and stochastic optimization to tackle the computational problems. Instead of decomposing the full tensor at once, updates are computed from small random block samples. Using step size restriction the decomposition can be found up to near optimal accuracy, while reducing the computation time and number of data accesses significantly. The scalability is illustrated by the decomposition of a synthetic 8 TB tensor and a real life 12.5 GB tensor in a few minutes on a standard laptop.

This chapter is based on 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, no. 2, pp. 284–295, Mar. 2016. DOI: 10.1109/JSTSP.2015.2503260. The figures have been updated for consistency.

5.1 Introduction

With datasets growing in size and dimensions faster than ever, more efficient algorithms to analyze them are needed. Many datasets can be represented as multiway arrays of numerical values. These so-called tensors can be compressed or analyzed using a variety of decompositions such as a low multilinear rank approximation, a block term decomposition or tensor trains (see e.g., [65], [170]). In this chapter we focus on the decomposition in rank-1 terms. A rank-1 tensor of order N is defined as the outer product, denoted by $\otimes$, of N vectors $\mathbf{a}^{(n)}$. The polyadic decomposition (PD) writes a tensor as a sum of R rank-1 terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} = \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right], \tag{5.1}$$

in which each factor matrix $\mathbf{A}^{(n)}$ has the R factor vectors $\mathbf{a}_r^{(n)}$ as its columns. If R is the minimum number for which the equality holds, R is the rank of the tensor, and the polyadic decomposition (8.1) is called canonical (CPD). Up to trivial scaling and permutation indeterminacies, a CPD is unique under mild conditions [96], [179], [241], which has led to countless applications, e.g., in factor analysis [43], [250] and blind signal separation [65], [71], [242]. Interested readers are kindly referred to [65], [170] for a general background.

Since the introduction of the (C)PD, many algorithms have been developed ranging from direct methods [76], [99], [233], over alternating least squares methods [139], [170], [278] to all-at-once optimization methods [5], [218], [226], [260], [279]. Albeit fairly mature, rank estimation, ill-conditioned decompositions, and ill-posed optimization problems remain important issues [248]. On top of that, most algorithms quickly run into trouble for large-scale tensors, as both the memory and per-iteration computational complexity are linear in the number of entries in the tensor which increases exponentially in the order. These problems are a manifestation of the curse of dimensionality [126], [129], [165], [304].

To overcome or at least reduce the difficulties related to large-scale tensors, many new strategies have emerged recently. A first category involves incomplete tensors where only a fraction of the elements of a tensor is known [5], [262], [276], [279], [304], [307]. The per-iteration complexity of these algorithms is linear in the number of known entries, which can be far lower than the number of entries in the full tensor. To take advantage of this, one can deliberately sample the tensor in only a few entries [304], [307]. A second category uses a similar idea for sparse tensors [16], [156], [219], where the per-iteration complexity is made linear in the number of nonzeros. Third, there are compression-based algorithms. The tensor can, for example, be compressed multiple times using random projections [245] or using random-

ized SVDs [69]. Finally, some algorithms decompose subtensors and then recombine the factor matrices. In the grid PARAFAC method the tensor is subdivided in a grid and each block is decomposed separately. The CPD structured blocks are then used to efficiently compute the CPD of the original tensor [223]. Alternatively, blocks can be sampled and decomposed after which the factor matrices are merged [219].

Rapid developments in parallel and distributed computing have led to a number of new algorithms exploiting the architecture. GigaTensor [156] and PARACOMP [245] use the MapReduce paradigm. In grid PARAFAC all the blocks in the grid are decomposed simultaneously [223]. In [219] the parallelization potential is outlined for the ParCube method. An alternating direction method of multipliers (ADMoM) for a mesh type architecture is shown in [187].

The randomized block sampling CPD algorithm outlined in this paper mainly falls in the fourth, block decomposition, category. Each iteration a new random block of (not necessarily adjacent) entries is sampled, and the affected variables are updated using the previous results as initialization. Then this is repeated for a new block of entries. The main difference with other methods in this category, i.e., grid PARAFAC and ParCube, is the use of ideas from stochastic optimization rather than factorizing blocks (from a grid or randomly sampled) and then recombining the results in the end. After a short overview of stochastic optimization concepts in section 5.2, the actual algorithm is outlined in section 5.3. section 5.4 conceptually discusses the behavior of the algorithm. Two parameters are introduced: the step size restriction strategy and the block size. Their influence on the accuracy and decomposition time is illustrated and discussed in section 5.5.

Notation

Scalars, vectors, matrices and tensors are denoted by lower case (e.g., a), bold lower case (e.g., $\mathbf{a}$), bold upper case (e.g., $\mathbf{A}$) and calligraphic (e.g., $\mathcal{T}$) letters, respectively. Index sets will be denoted by $\mathcal{B}$ and $\mathcal{I}$ for block and tensor level index sets, respectively. The norm $||\cdot||$ is defined as the Frobenius norm. The Kronecker product, the Khatri–Rao product and the Hadamard product are denoted by $\otimes$, $\odot$ and $*$, respectively, and the mode-n tensor unfolding of $\mathcal{T}$ is given by $\mathbf{T}_{(n)}$; see, e.g., [65] for formal definitions. The conjugate of a complex value a is denoted by $\bar{a}$ and the transpose, conjugated transpose, inverse and pseudoinverse by $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$, $\cdot^{-1}$ and $\cdot^{\dagger}$, respectively.

5.2 Stochastic optimization

Randomization is often used to scale up algorithms for big data, e.g., in randomized linear algebra [195] and in stochastic optimization [60]. The methods

discussed in this paper combine ideas from stochastic gradient descent (SGD) and block coordinate descent (BCD). In both cases the objective is to find the parameters $\mathbf{x}$ that minimize a function f:

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

Before developing our randomized block sampling CPD method, some background on both methods is given.

SGD [231] is a simple yet powerful technique widely used in optimization and machine learning. It also appeared as the classical least mean squares (LMS) filter which led to a large number of applications, e.g., in adaptive antenna arrays [308]. Suppose the objective function f can be decomposed in individual contributions f_n from each of the N_s data points:

$$f(\mathbf{x}) = \frac{1}{N_s} \sum_{n=1}^{N_s} f_n(\mathbf{x}).$$

An example of a decomposable function is the Frobenius norm error, e.g., to solve the overdetermined system $\mathbf{A}\mathbf{x} = \mathbf{b}$, the least squares formulation is

$$f(\mathbf{x}) = ||\mathbf{A}\mathbf{x} - \mathbf{b}||^2 = \sum_{n=1}^{N_s} (\mathbf{a}_n^{\mathrm{T}}\mathbf{x} - b_n)^2,$$

in which $\mathbf{a}_n^{\mathrm{T}}$ is the nth row of $\mathbf{A}$. In the SGD method, the gradient ∇f is estimated every iteration from a single random sample point n_k. The estimate of the parameters is then updated from the estimated gradient ∇f_{n_k} using

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{n_k}(\mathbf{x}_k),$$

in which α_k is the learning rate [60]. For matrix factorization, parallel implementations of SGD have been derived [113], [184]. Extensions to the CPD [240] and the (symmetrical) orthogonal decomposition in rank-1 terms [112] appeared recently.

In the case of the coordinate descent method a coordinate i_k is selected in every iteration. The parameter $\mathbf{x}$ is then updated using the exact gradient with respect to x_{i_k}

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla_{i_k} f(\mathbf{x}_k) \mathbf{e}_{i_k},$$

in which α is the learning rate and $\mathbf{e}_{i_k}$ is the i_kth column of an identity matrix [60]. Random selection of the coordinates results in the optimal convergence rate in expectation [207]. The BCD algorithm selects a block of coordinates in every iteration.

Second-order information is often avoided because of the computational

cost [60]. Despite this, the use of second-order information improves the convergence constants [35]. A couple of quasi-Newton approaches have emerged which approximate the Hessian by a diagonal matrix [24], [32] or using online LBFGS [238]. In the case of a CPD, the computation of a good Hessian approximation is relatively cheap, which makes a Gauss–Newton approach feasible.

5.3 CPD by randomized block sampling

The randomized block sampling (RBS) CPD algorithm is both a randomized BCD algorithm and a SGD method. In every iteration a block of variables is updated using an estimate of the gradient and the approximate Hessian based on a randomly sampled subblock of the tensor. The reason for this combination is the 'locality property' of the CPD: a single entry in an Nth order tensor of rank R affects only NR variables, while a block of size $B_1 \times \cdots \times B_N$ affects only $R\sum_{n=1}^{N} B_n$ variables. A CPD can be computed using a least squares approach which leads to the optimization problem

$$\min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} f \tag{5.2}$$

in which f is a decomposable function:

$$\begin{aligned} f &= \frac{1}{2}\left|\left|\mathcal{T} - \left[\!\left[\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\right]\!\right]\right|\right|^2 \\ &= \frac{1}{2}\sum_{i_1=1}^{I_1}\cdots\sum_{i_N=1}^{I_N}\left(t_{i_1\cdots i_N} - \sum_{r=1}^{R} a_{i_1 r}^{(1)} a_{i_2 r}^{(2)}\cdots a_{i_N r}^{(N)}\right)^2. \end{aligned}$$

Let a block be defined by the index sets $\mathcal{B}_1,\ldots,\mathcal{B}_N$, then the gradient ∇f is only nonzero for the variables $a_{i_n r}^{(n)}, i_n \in \mathcal{B}_n$, $n = 1,\ldots,N$, $r = 1,\ldots,R$.

Algorithm 5.1 gives a high level overview of the randomized block sampling CPD method. Every iteration k, a random block is sampled. Only the variables affected by this block are then updated. Contrary to the techniques outlined in section 5.2, a more informed update is computed using a relatively cheap approximation to the Hessian. The parameter Δ_k acts as the learning rate and is decreased in order to achieve convergence and to improve the accuracy as explained in section 5.4 and illustrated in the experiments. The sampling operator, the computation of the update and the learning rate selection are discussed in subsections 5.3.1 to 5.3.3, respectively. subsection 5.3.4 introduces a new stopping criterion based on the Cramér–Rao bound.

Algorithm 5.1: Randomized block sampling CPD.

while not converged **do**
  Randomly generate sample indices $\mathcal{B}_n \subseteq \mathcal{I}_n = \{1, \ldots, I_n\}$, $n = 1, \ldots, N$
  Let $\mathcal{T}_{\text{sub}} = \mathcal{T}(\mathcal{B}_1, \ldots, \mathcal{B}_N)$, and $\mathbf{A}_{\text{sub}}^{(n)} = \mathbf{A}_k^{(n)}(\mathcal{B}_n, :)$, $n = 1, \ldots, N$
  $\{\mathbf{A}_{\text{sub}}^{(n)}\} \leftarrow \text{update}(\mathcal{T}_{\text{sub}}, \{\mathbf{A}_{\text{sub}}^{(n)}\}, \Delta_k)$
  Set $\mathbf{A}_{k+1}^{(n)} = \mathbf{A}_k^{(n)}$ and $\mathbf{A}_{k+1}^{(n)}(\mathcal{B}_n, :) = \mathbf{A}_{\text{sub}}^{(n)}$, $n = 1, \ldots, N$
  $k \leftarrow k + 1$
end while

5.3.1 Sampling operator

Instead of randomly sampling blocks, we use a more involved sampling operator ensuring that every variable is updated at the same rate and allowing blocks to be decomposed in parallel. (A discussion of a parallel implementation is outside the scope of this paper.) The operator is illustrated in Figure 5.1.

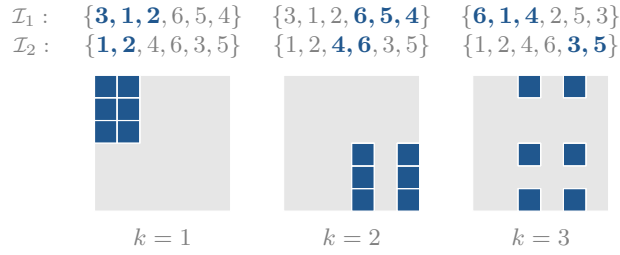


Figure 5.1: Illustration of the block sampling operator for a second-order tensor of size 6×6 and block size 3×2. *In iteration $k = 1$, the row index set $\mathcal{I}_1$ and the column index set $\mathcal{I}_2$ are shuffled, and first blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ are selected (bold). In iteration $k = 2$, the next block is selected. In iteration $k = 3$, the next column block $\mathcal{B}_2$ is sampled. The row index set $\mathcal{I}_1$ is permuted, as it has no blocks left, and $\mathcal{B}_1$ is the first block of the shuffled index set.*

A sample block $\mathcal{T}_{\text{sub}}$ can be generated by selecting a random subset $\mathcal{B}_n$ of B_n indices from $\mathcal{I}_n = \{1, \ldots, I_n\}$. Each iteration N new subsets are generated. Let $Q_n = I_n / B_n$ be the number of blocks per dimension, then after every Q_n iterations, the elements in the index set $\mathcal{I}_n$ are permuted. The indices $\mathcal{B}_n$ are now selected consecutively from $\mathcal{I}_n$, restarting at the first block when $\mathcal{I}_n$ is permuted. More formally, the index sets for $k = 1$ are $\mathcal{B}_n = \mathcal{I}_n(1 : B_n)$, for $k = 2$, $\mathcal{B}_n = \mathcal{I}_n(B_n + 1 : 2B_n)$, and for general $k \leq Q_n$, $\mathcal{B}_n = \mathcal{I}_n((k-1)B_n + 1 : kB_n)$. If $k = Q_n + 1$ no blocks are left and $\mathcal{I}_n$ is shuffled, after which the first indices again define the new index block, i.e., $\mathcal{B}_n = \mathcal{I}_n(1 : B_n)$. In general $\mathcal{B}_n = \mathcal{I}_n((l_n - 1)B_n + 1 : l_n B_n)$, $l_n =$

$\mathrm{mod}(k-1, Q_n)+1$ and $\mathcal{I}_n$ is permuted every time that $\mathrm{mod}(k-1, Q_n) = 0$. When Q_n is not an integer, there are two choices: either the last variables are ignored and $\mathcal{I}_n$ is shuffled when no full block is available, or a smaller block sample is determined by the remaining indices.

The block size and the randomization play important roles in the algorithm as they influence the robustness, the total computation time and the attainable accuracy. Section 5.4 discusses this influence and the effect will be thoroughly tested in the experiments in section 5.5. As randomly sampling blocks can result in suboptimal random accesses to slow memory, preprocessing techniques such as shuffling the data before starting the algorithm, which can be done as blocks do not depend on the updates computed using previous blocks, can be useful. Although these preprocessing techniques are important for practical implementation, a discussion of such techniques is out of the scope of this chapter.

5.3.2 Computing the update

Any CPD algorithm can be used to compute the update in Algorithm 5.1. Two variants are developed here: an alternating least squares (ALS) version and a nonlinear least squares (NLS) version.

ALS with step size restriction

ALS is a well-known optimization technique to solve the least squares problem (5.2); see, e.g., [139], [170], [278]). By fixing all but one factor matrix (say $\mathbf{A}^{(n)}$), Equation (5.2) becomes a linear least squares problem in the factor matrix $\mathbf{A}^{(n)}$:

$$\min_{\mathbf{A}^{(n)}} \frac{1}{2} \left|\left| \mathbf{T}_{(n)} - \mathbf{A}^{(n)} \mathbf{V}_k^{(n)^{\mathrm{T}}} \right|\right|^2, \tag{5.3}$$

in which the matrix $\mathbf{V}_k^{(n)} = \mathbf{A}_k^{(N)} \odot \cdots \mathbf{A}_k^{(n+1)} \odot \mathbf{A}_{k+1}^{(n-1)} \odot \cdots \odot \mathbf{A}_{k+1}^{(1)}$. The least squares problem (5.3) has an exact solution, but in order to introduce the step size parameter α, we compute the update explicitly as $\mathbf{A}_{k+1}^{(n)} = \mathbf{A}_k^{(n)} + \alpha \mathbf{P}_k$. The gradient and Hessian of the cost function in (5.3), evaluated in iteration k, are given by $\mathbf{G}_k = \mathbf{A}_k^{(n)} \overline{\mathbf{W}}_k^{(n)} - \mathbf{T}_{(n)} \overline{\mathbf{V}}_k^{(n)}$ and $\mathbf{H}_k = \mathbf{W}_k^{(n)} \otimes \mathbf{I}_{I_n}$, respectively, in which $\mathbf{W}_k^{(n)} = (\mathbf{V}_k^{(n)})^{\mathrm{H}} \mathbf{V}_k^{(n)}$ is computed efficiently as $\mathbf{W}_k^{(n)} = \mathbf{A}_{k+1}^{(1)^{\mathrm{H}}} \mathbf{A}_{k+1}^{(1)} * \cdots * \mathbf{A}_{k+1}^{(n-1)^{\mathrm{H}}} \mathbf{A}_{k+1}^{(n-1)} * \cdots * \mathbf{A}_k^{(n+1)^{\mathrm{H}}} \mathbf{A}_k^{(n+1)} * \cdots * \mathbf{A}_k^{(N)^{\mathrm{H}}} \mathbf{A}_k^{(N)}$ and $\mathbf{I}_{I_n}$ is the $I_n \times I_n$ identity matrix. The optimal step $\mathrm{vec}(\mathbf{P}_k)$ is given by $\mathbf{H}_k \mathrm{vec}(\mathbf{P}_k) = -\mathrm{vec}(\mathbf{G}_k)$ which is equivalent to $\mathbf{P}_k \mathbf{W}_k^{(n)^{\mathrm{T}}} = \mathbf{P}_k \overline{\mathbf{W}}_k^{(n)} = -\mathbf{G}_k$

using properties of the Kronecker product. Therefore, we find

$$\mathbf{P}_k = \mathbf{T}_{(n)}\overline{\mathbf{V}}_k^{(n)}\left(\overline{\mathbf{W}}_k^{(n)}\right)^{-1} - \mathbf{A}_k.$$

Finally, the updated factor matrix $\mathbf{A}_{k+1}^{(n)}$ is given as

$$\mathbf{A}_{k+1}^{(n)} = (1-\alpha)\mathbf{A}_k^{(n)} + \alpha\mathbf{T}_{(n)}\overline{\mathbf{V}}_k^{(n)}\left(\overline{\mathbf{W}}_k^{(n)}\right)^{-1}.$$

When $\alpha = 1$, the commonly used ALS update is obtained. Here we use $\alpha = \Delta_k$ to control the step lengths.

NLS with step size restriction

An optimum of (5.2) can also be found using all-at-once-optimization, and in the case of NLS problems, the Gauss–Newton algorithm can be used. Let $\mathbf{x}$ be the concatenation of the vectorized factor matrices $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. The Gauss–Newton algorithm solves (5.2) by linearizing $\mathcal{F}(\mathbf{x}) = \mathcal{T} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ in every iteration and solving

$$\min_{\mathbf{p}_k} \frac{1}{2} ||\mathrm{vec}\left(\mathcal{F}(\mathbf{x}_k)\right) - \mathbf{J}_k\mathbf{p}_k||^2$$

in which the step $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{J}_k$ is the Jacobian matrix $\partial \mathrm{vec}\left(\mathcal{F}\right)/\partial \mathbf{x}^{\mathrm{T}}$ evaluated at $\mathbf{x}_k$. The step size can be restricted by adding a constraint on the step size

$$\min_{\mathbf{p}_k} \frac{1}{2} ||\mathrm{vec}\left(\mathcal{F}(\mathbf{x}_k)\right) - \mathbf{J}_k\mathbf{p}_k||^2 \text{ s.t. } ||\mathbf{p}_k|| \leq \Delta_k. \tag{5.4}$$

This is a similar formulation as in trust region algorithms [209]. The main difference is that Δ_k is not updated based on the trustworthiness of the linearized model, but explicitly set by the stochastic algorithm. Similar strategies as for trust regions can be used to solve (5.4) approximately. In our implementation we use the dogleg method, which approximately finds a solution to (5.4) using the Gauss–Newton direction and the steepest descent direction [209], [260]. Usually we assume the step constraint is relative, i.e., $||\mathbf{p}_k|| \leq \Delta_k ||\mathbf{x}_0||$, where $\mathbf{x}_0$ is the initial guess. Note that, contrary to the ALS variant, Δ_k is only an upper bound of the step size.

5.3.3 Step size selection

In stochastic optimization the selection of the step size is an important element in the convergence of the algorithm. If the step size decreases too slowly, a lot of computation time is wasted, whereas one may not reach the

optimum in time if the step size decreases too fast. In a signal processing context, the data is often perturbed by noise, rendering step size selection schemes based on the function value hard. Often a fairly good solution can be attained without restricting the step size [205], which is also what we experience in the experiments. Therefore a search-then-converge strategy can be used [75] in which the step size is large and constant for a number of iterations and then gradually decreased. For simplicity we use a two step strategy:

$$\Delta_k = \begin{cases} \delta_0 & \text{if } k < K_{\text{search}} \\ \delta_{K_{\text{search}}} \cdot \alpha^{(k-K_{\text{search}})/Q} & \text{if } k \geq K_{\text{search}} \end{cases},$$

in which $Q = \max_n I_n/B_n$. The initial δ_0 can be quite large, e.g., 0.8 for the NLS method, and should be 1 for the ALS method. The parameter $\delta_{K_{\text{search}}}$ is generally smaller than δ_0 to speed up convergence. In the case of NLS, the restriction Δ_k is used on the relative step size $||\mathbf{p}_k|| \,/\, ||\mathbf{x}_0||$. The shrinkage factor $\alpha < 1$ and K_{search} have to be determined experimentally. As explained in section 5.4, K_{search} should be set large enough such that the algorithm has converged to the neighborhood of the optimum after K_{search} iterations, and α should be large, e.g., 0.99, to improve the accuracy. Tuning the parameters can shorten the computation time considerably. The parameters can often be found by decomposing a smaller, representative subtensor first [34]. This is illustrated in subsection 5.5.2.

5.3.4 Stopping criterion

We propose a new stopping criterion based on the estimated Cramér–Rao bound. Conventional stopping criteria in CPD algorithms involve either an evaluation of the objective function or depend on the (relative) step size. Both criteria can be used here as well, but some remarks have to be made. The function value is known to decrease only down to the noise level, which causes premature convergence in low SNR cases. Continuing to iterate often improves the solution (as shown in the convergence plots in subsection 5.5.1 and subsection 5.5.2). Moreover, we can only evaluate the function in sub-blocks as computing the function value for the full tensor is unfeasible for large-scale tensors. The function value, which is computed for a block, does not decrease monotonically, as one block may have a better fit than another one. It is also rather difficult to use the step size, as the step length is explicitly restricted by Δ_k.

The Cramér–Rao bound on the other hand takes the noise estimate into account as well as step size. The Cramér–Rao bound $\mathbf{C}$ gives a lower bound on the covariance matrix of the estimators $\hat{\mathbf{A}}^{(n)}$ of the variables $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. In the stopping criterion, we only consider the estimated lower bounds on the variances which can be found on the diagonal of $\mathbf{C}$. (In other

words, we do not use the covariances.) Let $\mathbf{c} = \text{diag}(\mathbf{C})$ be the vector of length $R\sum_{n=1}^{N} I_n$ that contains these variances, then we define $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, as the $I_n \times R$ matrix with the lower bounds on the variances, i.e., $\mathbf{C}^{(n)} = \text{reshape}(\mathbf{c}(J_{n-1}+1 : J_n), I_n, R)$, $J_n = R\sum_{k=1}^{n} I_n$. The Cramér–Rao bound then implies that for a noisy observation of a tensor generated by $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$, the estimator lies with a probability of 99.7% in the interval $\mathbf{A}^{(n)} \pm 3\boldsymbol{\Sigma}^{(n)}$, where $\boldsymbol{\Sigma}^{(n)} \geq \sqrt{\mathbf{C}^{(n)}}$ in which both $\geq$ and the square root are defined element-wise. If the current estimate $\mathbf{A}_k^{(n)}$, $n = 1, \ldots, N$ is close to a stationary point, we argue that it makes little sense to set steps that are small compared to the lower bound on the variance as the noise makes the result unreliable. Let us define the mean absolute difference between the estimated variables in iteration k and $k - K_{\text{CRB}}$ relative to the Cramér–Rao bound as

$$D_{\text{CRB}} = \frac{1}{R\sum_n I_n} \sum_{n=1}^{N} \sum_{i=1}^{I_n} \sum_{r=1}^{R} \frac{\left| \mathbf{A}_k^{(n)}(i,r) - \mathbf{A}_{k-K_{\text{CRB}}}^{(n)}(i,r) \right|}{\sqrt{\mathbf{C}^{(n)}(i,r)}}.$$

The algorithm is said to have converged if

$$D_{\text{CRB}} < \gamma \tag{5.5}$$

and γ is a constant, e.g., 0.5. The window K_{CRB} over which the change is computed, is introduced for robustness. Namely, if the step size is very small, but the steps proceed in the same direction, the overall difference over multiple steps can be large. On the other hand, if the algorithm repeatedly jumps over the optimum, the net difference will be small. Note that the scaling indeterminacies are taken care of: when the norm of a factor vector $\mathbf{a}_r^{(n)}$ increases, the corresponding entries in $\mathbf{C}^{(n)}$ increase appropriately.

The Cramér–Rao bound for a CPD is given by

$$\mathbf{C} = \sigma^2 (\mathbf{J}^{\text{H}}\mathbf{J})^{-1},$$

where $\mathbf{J}^{\text{H}}\mathbf{J}$ is the Gramian of the Jacobian of (5.2) [190], [276]. The Gramian is not invertible, however, as it has at least $(N-1)R$ singular values equal to zero due to scaling indeterminacies. These indeterminacies can be resolved by fixing $(N-1)R$ entries [190] [276], or the pseudoinverse can be used instead, as is illustrated here. The observation that $\mathbf{J}^{\text{H}}\mathbf{J}$ can be factored as $\mathbf{G}+\mathbf{Z}\mathbf{K}\mathbf{Z}^{\text{H}}$ with $\mathbf{G}$ and $\mathbf{Z}$ block diagonal [226], has led to an efficient inversion algorithm [276]. Using a similar derivation as in [276], the Cramér–Rao bound can be

computed using the pseudoinverse:

$$\begin{aligned} \mathbf{B} &= \mathbf{K}\left(\mathbf{I}_{NR^2} + \mathbf{Z}^{\mathrm{H}}\mathbf{G}^{-1}\mathbf{Z}\mathbf{K}\right)^{\dagger} \\ \frac{1}{\sigma^2}\mathbf{C} &= \mathbf{G}^{-1} - \mathbf{G}^{-1}\mathbf{Z}\mathbf{B}\mathbf{Z}^{\mathrm{H}}\mathbf{G}^{-1}. \end{aligned} \tag{5.6}$$

By exploiting that $\mathbf{G}$ and $\mathbf{Z}$ are block diagonal and that only the estimated lower bounds on the variances $\mathbf{C}^{(n)}$ are used, this bound can be computed very efficiently as long as R is low. (The computation cost is governed by the pseudoinverse of the $NR^2 \times NR^2$ matrix in Equation (5.6).) If the computational cost of evaluating $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$ becomes large compared to the amount of useful work, a computationally (much) cheaper variant can be used. As shown in [31], the inverse of the diagonal of the covariance matrix is larger than the diagonal elements of the inverse of the covariance matrix, i.e., let $\mathbf{F} = \mathbf{C}^{-1}$ then $\mathbf{C}(i,i) \geq (\mathbf{F}(i,i))^{-1}$, $i = 1, \ldots, R\sum_{n=1}^{N} I_n$. This means that only the diagonal of $\mathbf{J}^{\mathrm{H}}\mathbf{J}$ needs to be inverted, which contains only NR unique entries [260], [276] . This new bound is a, possible loose, lower bound for the Cramér–Rao bound [31], leading to a more severe stopping criterion. This can be compensated for by adjusting γ. In our experience, the entry-wise bound is usually a factor 1 to 5 smaller than the exact bound.

In practice the true underlying variables are unknown, as is the noise level σ. Instead of the true variables, the estimated variables are used as they are expected to converge to the real variables. The noise variance can be estimated from the previous K_{noise} function values f_k using

$$\sigma^2 = \frac{2}{K_{\text{noise}}(\prod_n B_n) - 1} \sum_{l=k-K_{\text{noise}}+1}^{k} f_l. \tag{5.7}$$

Recall that $f_k = 0.5 \left|\left| \mathcal{T}_{\text{sub}} - \left[\!\left[\mathbf{A}_{\text{sub}}^{(1)}, \ldots, \mathbf{A}_{\text{sub}}^{(N)} \right]\!\right] \right|\right|^2$ for the sample block used in iteration k. By dividing $2f_k$ by the number of elements $\prod_n B_n$ in a sample block, we obtain the average squared error which indeed defines the estimator of the variance. The minus one in (5.7) ensures that the estimator is unbiased. The noise window K_{noise} should be at least $\max_n Q_n$ such that all variables are accounted for in the noise computation. A large K_{noise} provides a better estimate of σ^2. Initially, when the solution is far off the optimum, the corresponding f_k do not provide a good estimate for the noise, hence K_{noise} should not be too large in order to ignore these initial values.

The Cramér–Rao bound used here is derived under the assumption of exact rank and Gaussian i.i.d. noise on the tensor. If these conditions do not hold, the Cramér–Rao bound may no longer be a lower bound for the variances. The estimated bound can suggest a good stopping criterion nonetheless.

5.4 Conceptual discussion

We now discuss the behavior of the algorithm as a whole. The goal of this section is to give some intuition on why the algorithm works rather than a rigorous proof. To keep ideas simple, we restrict our discussion to well behaved problems involving tensors with an exact CPD structure of a known rank R which is low compared to the dimensions, and possibly perturbed by zero mean i.i.d. noise such that the SNR is high enough. For these tensors, blocks with most dimensions larger than the rank can be selected. Note that we do not make claims for difficult decompositions, e.g., tensors that only just satisfy uniqueness conditions, tensors with very high rank or tensors with a low SNR. For these tensors identifiability can potentially be lost, or the estimation accuracy can be decreased when randomized block sampling is used.

The optimal estimator for a particular noise realization of a rank-R tensor $\mathcal{T} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ is denoted by $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. Note that the minimizers $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$ of optimization problem (5.2) are perturbed w.r.t. $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$ because of the perturbation of the tensor by that particular noise term. The progress of the algorithm can be split in two phases: the search phase during which the step size is unrestricted, and the convergence phase with step restriction. During the unrestricted phase, the algorithm converges to a neighborhood of a (local) optimum in which the iterates eventually jump around. The restricted phase can be seen as a variance reduction strategy which improves the accuracy of the solution and leads to an estimate of $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. Proving convergence to an optimum is still an open problem for general rank-R tensors. For truly big tensors, we often have to make compromises, however, and we therefore aim at finding a reasonably good solution. The experiments indicate that the RBS CPD algorithm succeeds in finding such a good approximation. For the remainder of this section, we focus on the NLS variant. The reasoning can be extended to the ALS variant.

5.4.1 Unrestricted phase

During this first phase, the algorithm tries to improve the initial guess $\mathbf{x}_0$ by moving the current iterate to a neighborhood of an optimum. We will now show this by leveraging uniqueness of blocks to uniqueness of the full tensor. We explain that near the end of this phase, the iterates $\mathbf{x}_k$ are jumping around in the neighborhood of an optimum because of the block sampling and the convergence properties of NLS type algorithms close to optima. The parameter K_{search} is chosen to mark the end of this phase.

We only consider blocks with most dimensions larger than the rank of the full tensor. First, consider the noiseless case. In this case, the CPD of a block with dimensions larger than the rank is likely to be unique; the solution can

actually be computed using a generalized eigenvalue decomposition [183]. The CPD of the full tensor inherits its uniqueness from the uniqueness of the sub-CPDs. If we add noise to the tensor such that the SNR remains high enough, we generally do not expect major problems to find the optimum for a block. We expect that the optimal CPD for the full tensor can now again be found from the sub-CPDs.

Using the reasoning above, a step computed from a block is generally expected to be well-conditioned in the sense that the optimum of the sub-CPDs will lead to the optimum for the full tensor. However, particular blocks can result in ill-conditioned steps. On the other hand, as only one step is computed for each block, the effect of this ill-conditioned block on the optimization process is limited. Due to the randomization, the probability of sampling many ill-conditioned blocks one after another is small as the total number of blocks is $\prod_{n=1}^{N} \binom{I_n}{B_n}$ and ill-conditioned blocks are often local phenomena, e.g., a part of a tensor that has lower rank (see subsection 5.5.3 for an example).

The NLS variant of the RBS CPD algorithm is a second-order algorithm. Each iteration, $\mathbf{x}_k$ is updated as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \mathbf{B}_k \mathbf{g}_k,$$

in which $\mathbf{B}_k = (\mathbf{J}_k^{\mathrm{H}} \mathbf{J}_k)^{\dagger}$ is a computationally cheap positive semidefinite approximation of the inverse Hessian. Because the restriction is not effective in this phase, the step size γ_k is chosen such that the quadratic model is minimized in each step, i.e.,

$$\gamma_k = \min \left(\frac{\mathbf{g}_k^{\mathrm{H}} \mathbf{g}_k}{\mathbf{g}_k^{\mathrm{H}} (\mathbf{J}_k^{\mathrm{H}} \mathbf{J}_k) \mathbf{g}_k}, 1 \right).$$

A property of second-order algorithms is their fast convergence near optima. At the end of the unrestricted phase, the algorithm is assumed to have converged to a neighborhood of an optimum. Therefore, the NLS algorithm (approximately) finds the optimum for the next block $k+1$ in one step regardless of the previous block. By consequence, $\mathbf{x}_{k+1}$ depends only on block $k+1$ and not on $\mathbf{x}_k$. This causes the algorithm to jump around in this neighborhood as each block has it own optimum because of the noise. The uncertainty on the estimates (which is determined by their variances) is larger for a block than the uncertainty that can be expected for the full tensor, as the dimensions of a block are (far) smaller than the dimensions of the full tensor. (This can be verified by comparing the Cramér–Rao bound for both the block and the full tensor, as this gives a lower bound on their variances.) This limits the accuracy of the solution in the unrestricted phase. In the following section, we show that step restriction reduces this loss in accuracy. Note that the next phase is only relevant for noisy tensors, as the uncertainty for exact

tensors is zero.

5.4.2 Restricted phase

At the end of the first phase, the iterates $\mathbf{x}_k$ are jumping around in a neighborhood of an optimum. By applying a step length restriction, we now show that the uncertainty can be reduced. Next, we illustrate how a good choice for the step restriction schedule reduces the variance quickly. Finally, we show how the Cramér–Rao bound stopping criterion interacts with the variance reduction.

We focus, without loss of generality, on one variable, say $a_k = \mathbf{A}_k^{(1)}(1,1)$. The discussion can be generalized to updates of all variables $\mathbf{x}_k$, but for the clarity of the presentation we only present the results for a single variable. We assume the permutation and scaling indeterminacies are resolved. (We also ignore the fact that the variable is only updated every Q_1 iterations.) Without step restriction, the variance of a_k is given by σ_b^2. As explained in the previous section, the NLS algorithm approximately finds the optimal value $\tilde{a}_{k+1}$ for the unrestricted case for block $k+1$ in one step, i.e., $\tilde{a}_{k+1} = a_k + p_k$, where p_k denotes the unrestricted step. As $\tilde{a}_{k+1}$ is the optimal value for the next unrestricted block problem, its variance is again σ_b^2. Now, we impose the step restriction γ_k

$$a_{k+1} = a_k + \gamma_k p_k = (1-\gamma_k)a_k + \gamma_k \tilde{a}_{k+1}.$$

The estimate a_{k+1} can be seen as a running average over all $k+1$ estimates $\tilde{a}_l$, $l = 1, \ldots, k+1$. Blocks k and $k+1$ are selected independently, and are both perturbed by mutually independent noise terms. As $\tilde{a}_{k+1}$ is the optimum for block $k+1$, $\tilde{a}_{k+1}$ is independent from a_k. Therefore, the variance of a_{k+1} is

$$\begin{aligned}\operatorname{Var}(a_{k+1}) &= (1-\gamma_k)^2 \operatorname{Var}(a_k) + \gamma_k^2 \operatorname{Var}(\tilde{a}_{k+1}) \\ &= (1-\gamma_k)^2 \operatorname{Var}(a_k) + \gamma_k^2 \sigma_b^2,\end{aligned}$$

Lemma 1. *Setting $\gamma_k \in (0,1)$ reduces the variance $Var(a_{k+1})$. For constant $\gamma_k = \gamma$, the variance asymptotically goes to zero as $\gamma \to 0$.*

Proof. In this proof, we keep $\gamma_k = \gamma$ constant. Define $\beta = (1-\gamma)^2$. Let a_0 be the guess at the end of the unrestricted phase, and $k = 1$ the first iteration in the restricted phase, then

$$\begin{aligned}\operatorname{Var}(a_1) &= \beta\sigma_b^2 + \gamma^2\sigma_b^2 && k = 1 \\ \sigma_b^{-2}\operatorname{Var}(a_k) &= \beta^k + \gamma^2 \sum_{l=0}^{k-1} \beta^l && k \geq 1 \\ \sigma_b^{-2}\operatorname{Var}(a_k) &= \frac{1}{1-\beta}\gamma^2 = \frac{\gamma}{2-\gamma} && k \to \infty.\end{aligned}$$

For $\gamma \in (0,1)$, we have $\gamma/(2-\gamma) < 1$, thus the variance is reduced. The derivative w.r.t. γ is $2/(2-\gamma)^2 > 0$ for $\gamma \in (0,1)$, hence decreasing γ reduces the asymptotic variance, and, for $\gamma \to 0$, $\mathrm{Var}(a_k) \to 0$. □

The proof above assumes constant γ_k. For decreasing step sizes γ_k, the expressions get more involved. This enables us to reduce the variance faster, as explained in the next paragraph.

Step restriction schedules

We now move on to decreasing step size γ_k. Figure 5.2 shows the evolution of the variance for different step restriction schedules. In subsection 5.3.3 we defined exponentially decreasing step schedules of the form $\delta_{K_{\text{search}}}\alpha^k$. Setting α closer to 1 reduces the variance at a slower rate, but has a larger variance reduction effect.

Figure 5.2 also shows the more conventional $1/k$ restriction scheme which reduces the variance to zero and may therefore seem more interesting. A key factor in the NLS algorithm is the iteration where the restriction becomes active. (Remember that Δ_k is an upper bound on the step size.) Suppose this occurs K_{active} iterations after the start of the restricted phase, then the $1/k$-type restriction behaves as $K_{\text{active}}/(k + K_{\text{active}})$, which reduces the variance far slower. The proposed exponentially decreasing step sizes are less sensitive to the choice of K_{search} (and hence K_{active}) as the variance is reduced at a more constant rate.

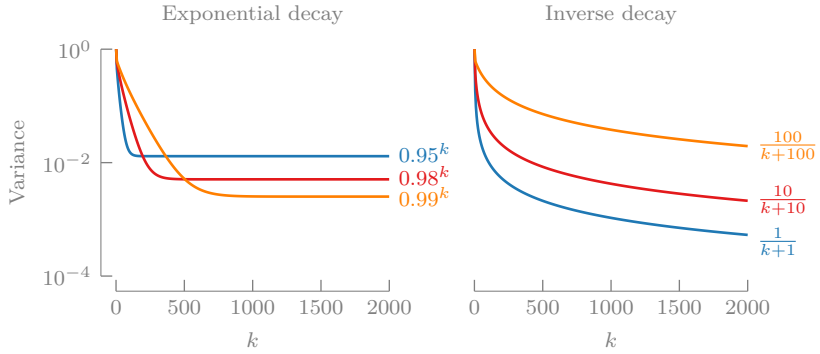


Figure 5.2: Decreasing the step size exponentially, reduces the variance quickly but levels off, while using inverse decays continue reducing the variance.

Stopping criterion

We have showed that the variance of the estimate $\mathrm{Var}(a_{k+1})$ can asymptotically be reduced to zero for certain choices of step restrictions γ_k. As

indicated in the beginning of this section, the estimates $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$ are not identical to the true $\mathbf{A}^{(n)}$ because of perturbations caused by noise. After a number of iterations, the step restriction can reduce the 'algorithmic' uncertainty of a_{k+1} below the 'intrinsic' uncertainty due to noise on the tensor. The Cramér–Rao bound stopping criterion takes this into account.

5.5 Analysis and experiments

The step size restriction strategy and the selection of the block size are important parameters in the algorithm. The following experiments illustrate their effect on the number of data accesses, computation time and accuracy. Except for the last experiment where hazardous gasses are analyzed, we focus on synthetically generated data. All tensors are generated using random factor matrices in which the elements are drawn from either the standard normal distribution $\mathcal{N}(0, 1)$ or the uniform distribution $\mathcal{U}(0, 1)$. The former distribution results in well-conditioned factor matrices, as the expected angle between the factor vectors is 90°. When using the latter distribution, the expected angle is 42° which results in less well-conditioned factor matrices. If noisy tensors are used, the elements are perturbed by i.i.d. Gaussian noise which is scaled to obtain a given signal-to-noise ratio (SNR). To initialize the algorithm, random factor matrices are generated from the same distribution as the original factor matrices, and the exact rank is used. Finding the rank of a tensor can be a hard problem in practice and the rank is often determined by prior knowledge or multiple attempts with different ranks. These techniques can be used here as well, and will not be discussed further. In all experiments, two random initializations are used for each Monte Carlo experiment and the best result is retained. The algorithms are implemented in Matlab and make use of (adapted versions of) existing Tensorlab routines for the update step, more in particular `cpd_als` and `cpd_nls` with the Gauss–Newton solver with dogleg trust regions [260], [261]. Unless explicitly stated otherwise, the algorithm is said to have converged if the relative step size is smaller than 10^{-15} or the Cramér–Rao bound criterion (5.5) is met for $\gamma = 0.1$ and $K_{\mathrm{CRB}} = 2$. To measure the accuracy of the estimates $\hat{\mathbf{A}}^{(n)}$ the CPD error E_{CPD} compared to the original factor matrices $\mathbf{A}^{(n)}$ is used, where

$$E_{\mathrm{CPD}} = \max_n \frac{\left|\left|\hat{\mathbf{A}}^{(n)} - \mathbf{A}^{(n)}\right|\right|}{\left|\left|\mathbf{A}^{(n)}\right|\right|},$$

in which the scaling and permutation ambiguities have been resolved (using `cpderr` [261]). The timing experiments are performed on a standard laptop (quad core i7-4800MQ @ 2.70 GHz, 16 GB RAM, 256 GB Toshiba THNSNH25 SSD) running Matlab 2014b on Ubuntu 14.04.

5.5.1 Influence of the step size adaptation

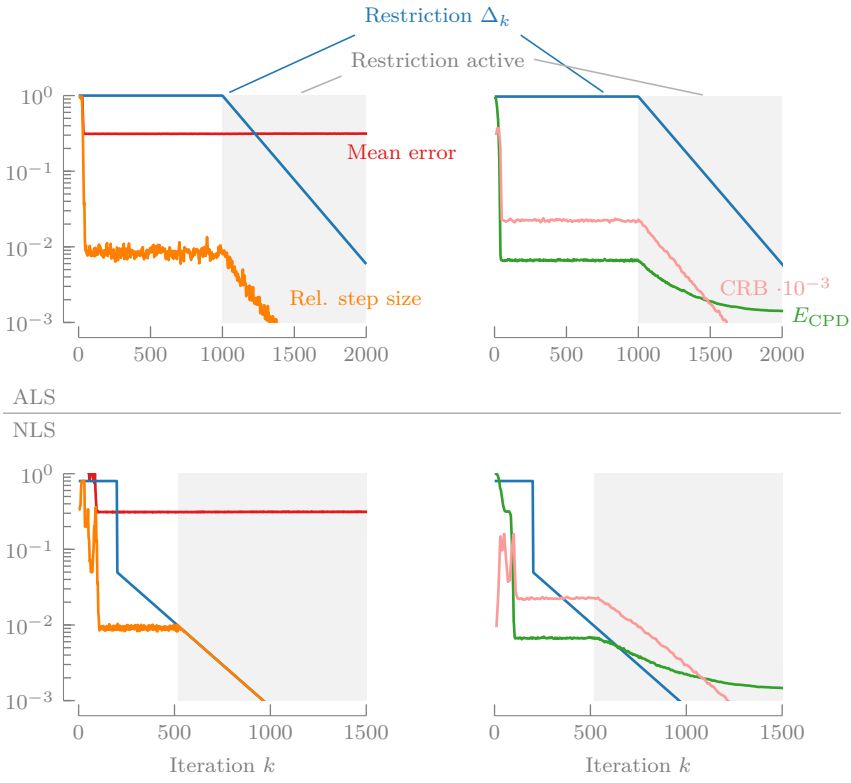


Figure 5.3: The CPD error E_{CPD} is decreased further when the step restriction becomes active. *Typical convergence plots for ALS (top) and NLS (bottom) are shown. In case of ALS $\alpha = \Delta_k$ influences the step size, but does not impose a direct constraint which explains why the curves do not touch.*

First we analyze the effects of step size restriction. As an example a rank $R = 10$ tensor of size $250 \times 250 \times 250$ is generated using well-conditioned factor matrices with entries drawn from the standard normal distribution $\mathcal{N}(0, 1)$. The SNR is set to 20 dB. Blocks of size $50 \times 50 \times 50$ are sampled. The step restriction schedule first keeps Δ_k constant for $K_{\mathrm{search}} = 200$ iterations in the NLS case and $K_{\mathrm{search}} = 1000$ iterations in the ALS case. After the search phase, the step size is decreased exponentially using $0.95^{(k-K_{\mathrm{search}})/10}$. (For the NLS variant, there is also a jump from 0.8 to 0.05 such that the restriction becomes effective sooner.) The convergence plots for both the NLS and the ALS variants are shown in Figure 5.3. When $k < K_{\mathrm{search}}$ both variants converge to a reasonably accurate solution, but the relative step size and the

change in variables compared to the Cramér–Rao bound remain relatively high. This indicates that the algorithm is jumping around the optimum. In the NLS case, there is no effective restriction as the relative step size is always smaller than $\Delta_k = 0.8$. In the ALS case $\Delta_k = 1$, which means common unrestricted ALS iterations are performed. As the exact solution is known, we can monitor the CPD error E_{CPD}, and it has a similar behavior as the step size. When Δ_k starts restricting the step size (around $k = 550$ for NLS and $k = 1000$ for ALS), the step size, the change relative to the Cramér–Rao bound and E_{CPD} start decreasing again. Note that the function value (mean error in Figure 5.3) appears to be constant after few iterations at a level determined by the noise. However, a lot of progress can still be made in terms of accuracy as shown by the E_{CPD} curve. Figure 5.4 shows the effect of restricting the step size more clearly for uniformly distributed factor matrices and the NLS variant. By carefully choosing the step restriction schedule a solution as accurate as the full tensor solution can be attained in little time if the SNR is high enough. This is illustrated further in subsection 5.5.2.

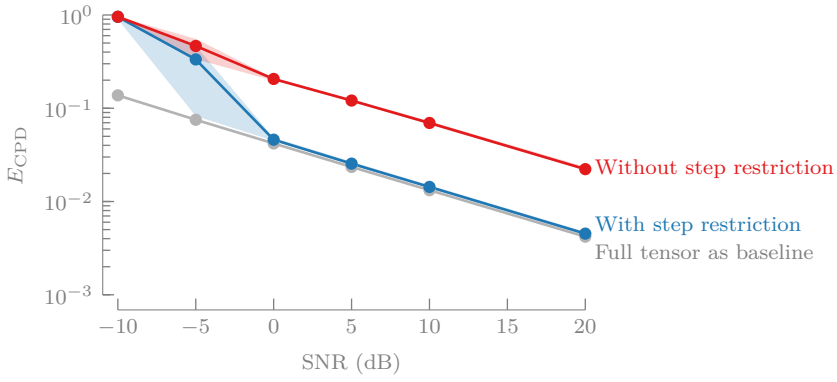


Figure 5.4: Thanks to step restriction, the error E_{CPD} is as small as if the full tensor is used, given a large enough SNR. *The shadings indicate the minimum and maximum accuracy achieved over all experiments over 50 Monte Carlo experiments.*

Typically, ALS performs well on well-conditioned data like the normally distributed factor matrices used in the previous experiment. This is illustrated in Figure 5.5. Here both normally distributed and uniformly distributed factor matrices are used. (All other parameters are the same as in the previous experiment.) For normally distributed factor matrices, ALS achieves the same median accuracy as NLS, but sometimes it does not converge within the maximum of 2000 iterations, hence the large spread. NLS on the other hand always converges. When using uniformly distributed random factor matrices, the problem is more difficult. ALS now only converges to

a good solution if the SNR is high, and even then ALS often fails to find a good solution. The NLS variant always finds a good solution if the SNR is not too low. Table 5.1 shows the results in more detail for a SNR of 10 dB. If ALS converges, a good solution is found quickly. For uniformly distributed factor matrices, the difference in computation time between ALS and NLS becomes small. ALS consumes much more data, however: while NLS often converges before all entries are accessed, ALS processes every element multiple times. The choice between the ALS and NLS version hence depends on the condition of the data and the cost of accessing or generating data.

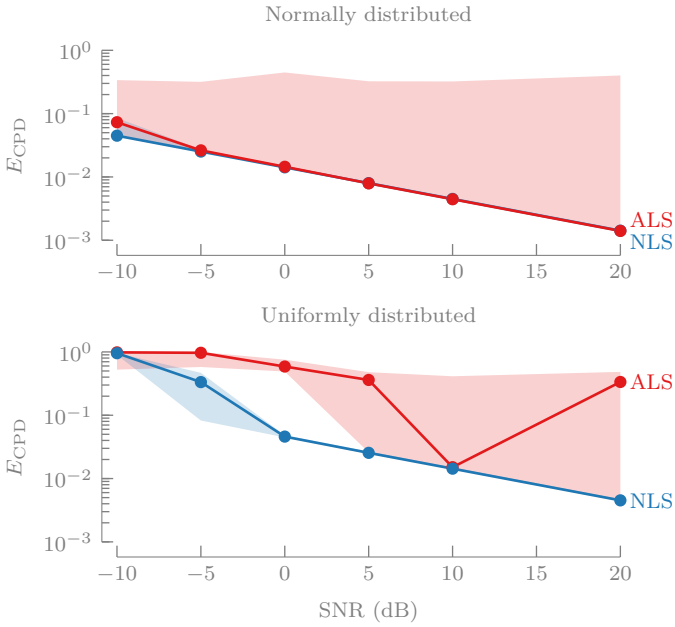


Figure 5.5: The NLS variant consistently performs as well as or better than the ALS variant, especially for more difficult problems involving entries drawn from a uniform distribution. *Moreover, ALS often fails to find an accurate solution. Medians over 50 Monte Carlo experiments are shown. The shadings indicate the minimum and maximum accuracy achieved over all experiments.*

5.5.2 Step size selection for an 8 TB tensor

The previous analysis showed the importance of the step selection strategy. Now we illustrate how to choose this strategy for a $1000 \times 1000 \times 1000 \times 1000$ tensor, which would consume 8 TB of memory if generated. The rank $R = 20$ tensor is generated from uniformly distributed random factor matrices. In the first factor matrix, we set the top half of the first 19 columns to zero,

Table 5.1: When ALS converges, it is usually fast, but needs a lot of samples; NLS always converges and often does not access the full tensor. *The percentage of experiments in which the algorithm converged, the median percentage of data entries sampled and the median run time are given for the SNR* $= 10\,\mathrm{dB}$ *case and for normally and uniformly distributed factor matrices.*

Distribution	Method	Conv. (%)	Data (%)	Time (s)
Normal	ALS	86	748.0	14.45
	NLS	100	56.6	30.26
Uniform	ALS	76	253.0	23.10
	NLS	100	42.4	26.57

which means the top half of the tensor has only rank 1. The tensor is then perturbed by Gaussian i.i.d. noise such that the SNR is 20 dB. Blocks of size $40 \times 40 \times 40 \times 40$ are used and are generated when needed. To determine the step size, a small random subtensor of size $80 \times 80 \times 80 \times 80$ is sampled from the tensor, hence $Q = 2$. The NLS variant is used in combination with the following initial step size strategy:

$$\Delta_k = \begin{cases} 0.8 & \text{if } k < 100 \\ 0.1 \cdot (0.85)^{(k-100)/2} & \text{if } k \geq 100 \end{cases}$$

Figure 5.6 shows the resulting convergence behavior. The step size restriction becomes effective only after 140 iterations, while the convergence stagnates after 25 iterations, which is confirmed by the CPD error. The computation time can be reduced by restricting the step size earlier, e.g., at $k = 30$. Additionally, we will reset Δ_k to 0.01 at $k = 30$ because the step size drops rather slowly, resulting in only small decreases in the change relative to the Cramér–Rao bound.

To convert the step restriction strategy to the full tensor, we take into account that instead of 2 iterations, now $Q = 25$ iterations are needed to update all variables. To compensate for this we take $K_{\text{search}} = 12.5 \cdot 30 \approx 400$. This results in the following strategy:

$$\Delta_k = \begin{cases} 0.8 & \text{if } k < 400 \\ 0.01 \cdot (0.85)^{(k-400)/25} & \text{if } k \geq 400 \end{cases}$$

The result for the full tensor is shown in Figure 5.7. The convergence profile is indeed very similar to the $80 \times 80 \times 80 \times 80$ case. The total decomposition time for this tensor is 308 s: the decomposition of the small sample to determine the step restriction schedule takes 24 s and the decomposition of the full tensor 284 s. Note that the shuffling operation is important here. If the first index set $\mathcal{I}_1$ is not shuffled every Q_1 iterations, the rank-1 blocks will cause the algorithm to fail.

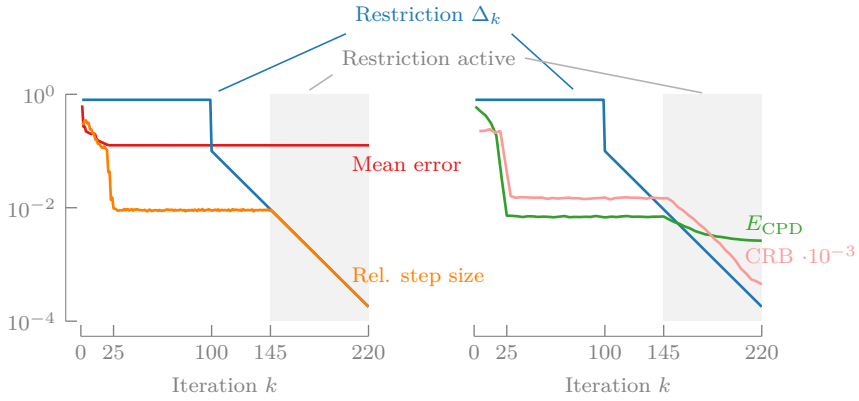


Figure 5.6: To determine good step restriction parameters, a smaller random sample of size $80 \times 80 \times 80 \times 80$ is decomposed first. *The parameters can be improved as the restriction becomes active only after 145 iterations, while the step size stagnates after 25 iterations. (E_{CPD} is usually unknown.) Sample blocks of* $40 \times 40 \times 40 \times 40$ *are used. The SNR w.r.t. the* $1000 \times 1000 \times 1000 \times 1000$ *full tensor is SNR of* 20 dB.

5.5.3 Influence of the block size

Another important parameter is the size of the sampled blocks. In this section, the influence of the block size on the time needed to converge, the number of data accesses and the accuracy is illustrated.

First the computation time and number of data accesses are investigated for a noiseless tensor of size $800 \times 800 \times 400$ (1.9 GB). The rank $R = 20$ and the original factor matrices are generated from a uniform distribution $\mathcal{U}(0, 1)$. No step size restriction schedule is needed in this noiseless exact case. The parameter ν is used to vary the sample size $[4 \quad 4 \quad 2] \cdot \nu$. The `cpd_nls` method [260], [261] is also applied to the full tensor for comparison. The algorithm is stopped when the CPD error $E_{\text{CPD}} < 10^{-8}$. Figure 5.8 shows timing results for this experiment. As shown in Figure 5.9, the computation cost per block is low when the block size is small, but many iterations are needed in order to converge. The reverse is true for large blocks. In this case, this leads to an optimal block size for $\nu = 20$ (Figure 5.8). However, if generating data is expensive, smaller block sizes may be preferable as the number of data accesses is lower, as can be seen in Figure 5.10. For small blocks, the algorithm finds the exact solution without accessing all elements in the tensor.

In the next experiment, Gaussian i.i.d. noise is added to the tensor such that the SNR is 20 dB. All other parameters are the same as in the previous experiment, except that the Cramér–Rao bound stopping criterion is now used with $\gamma = 0.01$. The resulting CPD errors are shown in Figure 5.11. For



Figure 5.7: Decomposition of the full $1000 \times 1000 \times 1000 \times 1000$ tensor with a SNR of 20 dB and sample blocks of size $40 \times 40 \times 40 \times 40$.

small block sizes, the algorithm does not find an accurate solution. Above a critical block size, the accuracy can be improved by using larger blocks. When using the step size restriction schedule the improvement no longer seems proportional as the accuracy curve flattens. Again we see that restricting the step size improves the accuracy significantly if the block sizes are large enough.

5.5.4 Classifying hazardous gasses

We conclude the experiments with a chemo-sensing application. The goal is to predict which chemical analyte, in casu a hazardous gas, is detected by an array of sensors using time series. The dataset consists of ten different analytes, measured at six positions in a wind tunnel under different (turbulent) wind conditions and temperatures. In total 18000 time series of 260 seconds sampled at 100 Hz were collected for each of the 72 sensors [299].

Here, we only consider the measurements for CO, acetaldehyde and ammonia at the first position (L1). We perform minimal preprocessing: missing values are linearly interpolated, the 50 first and 83 last samples of each time series are trimmed, each time series is filtered using a simple moving average of length 10 and each time series is centered by subtracting its mean. One experiment is dropped because it contained too many missing values. Here, we only use spatiotemporal patterns to classify the analytes and ignore scale, wind and temperature information. Therefore, we normalize all time series for each experiment by dividing each time-sensor slice by its Frobenius norm. The resulting tensor has dimensions $25900 \times 72 \times 899$. Loading this 12.5 GB tensor into RAM takes about 10 minutes from an SSD. This is close to the

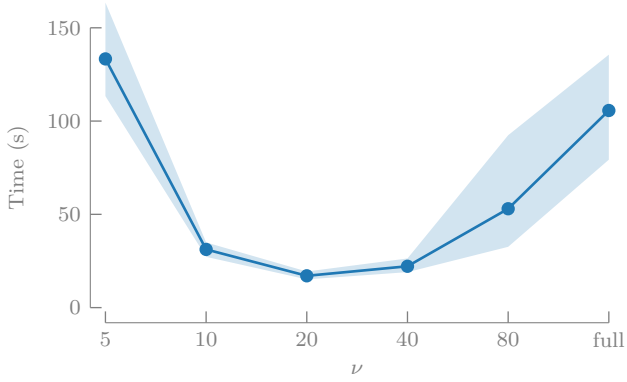


Figure 5.8: Choosing the optimal block size in terms of computation time involves a trade-off between the number of iterations and the cost for decomposing a sampled block. *The median computation time over 50 Monte Carlo experiments for blocks of size* $[4 \quad 4 \quad 2] \cdot \nu$ *to obtain a CPD error* $E_{CPD} < 10^{-8}$ *is shown. The noiseless tensor has size* $800 \times 800 \times 400$ *and is generated using uniformly distributed factor matrices.*

largest tensor we can load on a system with 16 GB of RAM. If the tensor does not fit into RAM, the data can be split across multiple files. Each time a block is needed, the appropriate files need to be read from disk resulting in a high IO cost.

To classify the analytes, we cluster the coefficients of the spatiotemporal features, i.e., time-sensor patterns. The patterns are determined using a rank $R = 5$ CPD. Each row in the experiment mode factor matrix hence contains five coefficients, one for each of the patterns. To compute this CPD, the NLS variant with blocks of size $100 \times 36 \times 100$ is used. We do two experiments. In the first experiment no step size restriction is imposed by setting $\delta_0 = 1.5$ for

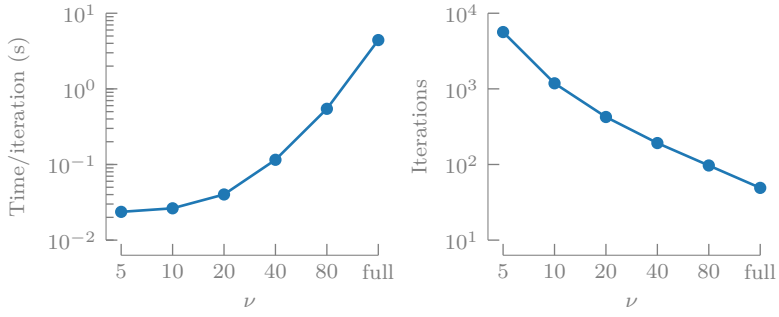


Figure 5.9: Detail of trade-off between cost per block and number of iterations. *The median over 50 Monte Carlo experiments is shown for blocks of size* $[4 \quad 4 \quad 2] \cdot \nu$.

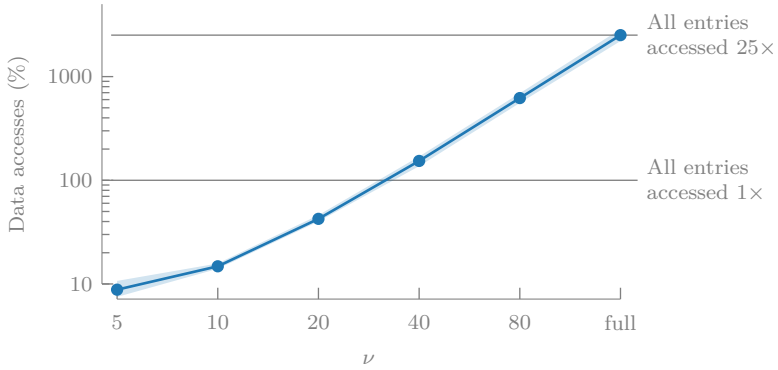


Figure 5.10: When using smaller blocks, not all tensor entries are required to recover the CPD using RBS. *The number of data accesses compared to the number of entries in the tensor for blocks of size* $[4 \quad 4 \quad 2]\cdot\nu$ *is shown. The noiseless tensor has size* $800\times800\times400$ *and is generated using uniformly distributed factor matrices. The algorithm is stopped when* $E_{CPD} < 10^{-8}$*. The median over 50 Monte Carlo experiments is shown.*

all iterations. In the second experiment the step size is restricted after 2000 iterations using $1.5 \cdot 0.93^{(k-2000)}$. As initialization we use factor matrices drawn from a normal distribution. The resulting factor matrices for the second experiment are shown in Figure 5.12. As the experiments are grouped per analyte, we see that the top three factor vectors in the experiment mode can distinguish between the three different analytes that we consider. The noise on these vectors is due to the different wind and temperature conditions, as well as optimization noise because the variance has not been fully reduced. The distinctive coefficients are clustered using $k$means, which is repeated 100 times after which we use the most frequently occurring cluster for each experiment as the predicted cluster. The performance for both experiments can be seen in Table 5.2. Imposing the step size restriction clearly improves the prediction accuracy, at the cost of a longer computation time. The time needed with restriction is still less than 3 minutes, however.

Using only three spatiotemporal features, a good classification accuracy can be achieved. The task will become more difficult if more analytes are used. In that case other features such as scale and wind and temperature measurements can be used to complement the pattern coefficients.

5.6 Conclusion

The randomized block sampling CPD algorithm presented here enables the decomposition of large-scale tensors using only small random blocks from the tensor. The advantage of using smaller blocks is twofold: small blocks

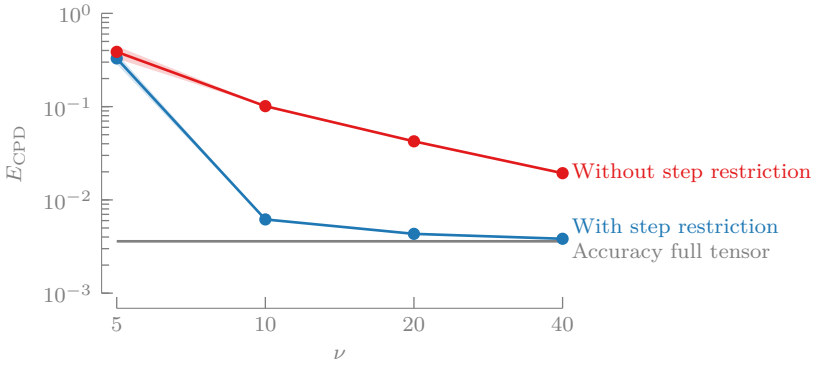


Figure 5.11: The accuracy can be improved greatly by using step restriction. *Median accuracy over 50 Monte Carlo experiment for varying block sizes* $[4 \quad 4 \quad 2] \cdot \nu$ *with and without step restriction strategy. The rank 20 tensor of size* $800 \times 800 \times 400$ *is constructed using uniformly distributed factor matrices and the SNR is* 20 dB. *The NLS variant is used.*

Table 5.2: Performance on the chemical analytes dataset without and with step restriction. *Step restriction clearly improves the prediction accuracy, the price being a somewhat longer computation time.*

	Iterations	Time (s)	Error (%)
No restriction	3000	60	5.0
Restriction	9000	170	0.3–0.8

can be handled more efficiently than the full tensor, and the number of data accesses needed to converge is far lower as the algorithm often needs only a fraction of all entries. We have developed two variants of our method. The ALS version is fast but needs many iterations, and thus data accesses, and may not converge for ill-conditioned datasets. The NLS variant on the other hand is robust and needs only very little data, but may be slower. We have shown that a good choice for the step size restriction schedule and block size allows our algorithm to find the CPD with an accuracy close to the accuracy achieved by state-of-the-art algorithms using the full tensor. Finally, we have introduced a new stopping criterion based on the Cramér–Rao bound that compares the actual change in variables to the expected uncertainty on the solution.

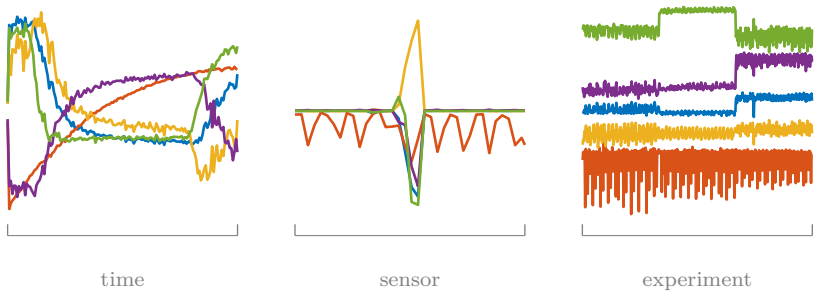


Figure 5.12: Recovered factor matrices for a rank $R = 5$ CPD of the chemical analytes dataset. *The vectors in the experiment mode have been shifted vertically for illustrative reasons. The top three vectors in the experiment mode appear the most distinctive and are used for classification.*

Exploiting efficient representations in large-scale tensor decompositions

6

ABSTRACT | Decomposing tensors into simple terms is often an essential step to discover and understand underlying processes or to compress data. However, storing the tensor and computing its decomposition is challenging in a large-scale setting. Though, in many cases a tensor is structured, i.e., it can be represented using few parameters: a sparse tensor is determined by the positions and values of its nonzeros, a polyadic decomposition by its factor matrices, a tensor train by its core tensors, a Hankel tensor by its generating vector, etc. We show that the complexity of tensor decomposition algorithms is reduced significantly in terms of time and memory if these efficient representations are exploited directly, thereby avoiding the explicit construction of multiway arrays. Key to this performance increase is rewriting the least squares cost function, which exposes core operations such as norms and inner products. Moreover, as the optimization variables do not change, constraints and coupling can be handled trivially. To illustrate this, large-scale nonnegative tensor factorization is performed using Tucker and tensor train compression. We show how vector and matrix data can be analyzed using tensorization while keeping a vector or matrix complexity through the new concept of implicit tensorization, as illustrated for Hankelization and Löwnerization. The concepts and numerical properties are extensively investigated with experiments.

This chapter is based on N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in tensor decompositions", Technical Report 16–174, ESAT-STADIUS, KU Leuven, Belgium, Oct. 2017.

6.1 Introduction

In signal processing and data analysis, tensors are often given as multiway arrays of numerical values. Tensor decompositions are then used to discover patterns, separate signals, model behavior, cluster similar phenomena, detect anomalies, predict missing data, etc. More concrete examples include intrusion detection in computer networks [219], analysis of food samples [45], crop classification using hyperspectral imaging [309], direction of arrival estimation using a grid of antennas [242] and modeling melting temperatures of alloys [304]. In these contexts, the canonical polyadic decomposition (CPD), the low multilinear rank approximation (LMLRA), or Tucker decomposition, and the block term decomposition (BTD) are common. To improve interpretability, constraints such as nonnegativity and smoothness can be imposed. When multiple datasets describing (partially) the same phenomenon are available, coupling or data fusion allows more meaningful patterns to be extracted. More examples can be found in several overview papers; see, e.g., [65], [170], [243].

Given as dense arrays of numerical values, the cost of storing and processing tensors tends to increase quickly, especially for high orders: for an Nth-order tensor of size $I \times I \times \cdots \times I$ the number of entries is I^N. To deal with these large-scale tensors, various techniques emerged to handle their decompositions, e.g., by deliberately sampling entries [302], [304], through randomization [300], through compression and/or using parallelism, e.g., [64], [156], [187], [219], [245], [251]. Here we consider tensors that are structured and can be represented efficiently using fewer parameters than the number of entries in the dense array. Moreover, we assume that structure is exploitable to reduce the computational and memory complexity from the $\mathcal{O}$(tensor entries) to $\mathcal{O}$(parameters). We identify three important types of efficient representation that allow such exploitation: compact, exact representation, compression-based representation and (implicit) tensorization.

For the first type, the tensor has an *exact and known structure* that can be represented compactly. A sparse tensor, for instance, is represented by the indices and values of the nonzeros, or more efficiently using compressed formats [16], [252]. For sparse tensors, the compact representation has been exploited extensively; see, e.g., [16], [64], [149], [151], [156], [158], [159], [171], [219], [251]–[253]. A system may be modeled mathematically as a polyadic decomposition, with a possible non-minimal number of rank-1 terms. In this case, the factor matrices form the compact representation [203], [229].

Second, in the case of *compression-based representations*, the tensor is first approximated using a decomposition that is easier or cheaper to compute and this approximation is then used to speed up subsequent computations. Common examples are a truncated MLSVD [79], a hierarchical Tucker approximation [124], [132] or a tensor train (TT) approximation [216]. Tucker compression is used extensively as a preprocessing step to speed up the com-

putation of an unconstrained CPD, as it suffices to decompose the core tensor [46], which follows from the CANDELINC model [57], [166]. To design effective updating algorithms, a previously computed compact representation of the old data combined with a newly arrived slice is used in [291]. In scientific computing, recompression or rounding are often used to lower the multilinear rank of a PD or a Tucker decomposition [163], [164], [215], [235], [236], or to lower the TT rank of a tensor given as a PD [216] or a TT approximation [143], [211]. Other examples include the approximation of a matrix or tensor by a greedy orthogonal rank-1 decomposition [169], a hierarchical Kronecker tensor product decomposition [130] or a two-level rank-$(r_1, \ldots, r_d)$ decomposition [161].

The third type is *implicit tensorization*. Tensorization involves mapping vectors or matrices to higher-order tensors [65], [85]. By decomposing the obtained tensors, mild uniqueness conditions of tensor decompositions can be leveraged. However, for some types of tensorization, the number of data points increases drastically. For example, third-order Hankelization of a vector of length M results in a tensor with $\mathcal{O}\left(M^3\right)$ entries [242], while Löwnerization [90] of K observations of mixed rational functions, sampled in M points, may result in a tensor of dimensions $M/2 \times M/2 \times K$. In both examples, the signal length M is limited by the respective cubic and quadratic dependence of the number of entries on M. To avoid this data explosion, we propose the concept of implicit tensorization: by operating on the underlying data directly, no explicit construction of the tensor is required. Implicit tensorization alleviates computation and memory cost if the data underlying the tensor has few parameters, and the structure of this representation can be exploited easily, e.g., through fast Fourier transforms (FFT); see section 6.3. Examples are Hankelization, Löwnerization, Toeplitzization, segmentation with overlap and outer product structures [29], [85]. In this way, longer signals can be handled, as illustrated in subsection 6.4.3.

6.1.1 Contributions

We explain how efficient representations originating from each of the three types of efficient representations can be exploited in optimization-based algorithms for the computation of a CPD, a decomposition into multilinear rank-$(L_r, L_r, 1)$ terms (LL1), an LMLRA or a general BTD. By rewriting the least squares loss function, core operations are exposed, each of which can be implemented such that the structure underlying the efficient representation is exploited. While a number of papers propose similar ideas for the computation of a CPD or LMLRA in the specific cases of sparse tensors [16], [64], [149], [151], [156], [158], [159], [171], [219], [251]–[253], or CPD and LMLRA approximations [317]–[319], we show that these ideas fit in a broader framework for structured tensor decompositions, including tensorization methods. Moreover, we show that, in contrast to previous results, both

first and second-order optimization algorithms can be used and that constraints and joint factorizations are handled trivially. We also pay attention to often neglected numerical issues. Finally, we show that our approach enables enormous speedups for large-scale problems even when constraints are imposed or when multiple datasets are jointly factorized, in contrast to the traditional compression approach using the CANDELINC model [46] which cannot be used in these cases. We demonstrate this experimentally for large-scale nonnegative tensor factorization using Tucker and TT compression and for implicit Hankelization of signals with up to 500 000 samples. Matlab implementations for the framework are available in Tensorlab [305].

6.1.2 Outline

After discussing the notation in the remainder of this section, section 6.2 explains how efficient representations of structured tensors can be exploited by rewriting the Frobenius norm objective function and gradient for the CPD, LL1, LMLRA and BTD. Implementations of the four core operations — norm, inner product, matricized tensor times Khatri–Rao product and matricized tensor times Kronecker product — are given in section 6.3 for structured tensors given as a CPD, LMLRA, TT, Hankel tensor or Löwner tensor. Finally, the experiments in section 6.4 discuss the numerical properties of the presented framework and illustrate the performance for compression-based nonnegative CPD and blind separation of exponential polynomials through implicit Hankelization.

6.1.3 Notation

Scalars, vectors, matrices and tensors are denoted by lower case, e.g., a, bold lower case, e.g., $\mathbf{a}$, bold upper case, e.g., $\mathbf{A}$ and calligraphic, e.g., $\mathcal{T}$, letters, respectively. $\mathbb{K}$ denotes either $\mathbb{R}$ or $\mathbb{C}$. Sets are indexed by superscripts within parentheses, e.g., $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. A mode-n vector is the generalization of a column (mode-1) and row (mode-2) vector and is defined by fixing all but the nth index of an Nth-order tensor. The mode-n unfolding of a tensor $\mathcal{T}$ is denoted by $\mathbf{T}_{(n)}$ and has the mode-n vectors as its columns. The mode-(m, n) unfolding is defined similarly and is denoted by $\mathbf{T}_{(m,n)}$. The vectorization operator $\operatorname{vec}(\mathcal{T})$ stacks all mode-1 vectors into a column vector. A number of products are needed. The mode-n tensor-matrix product is denoted by $\mathcal{T} \cdot_n \mathbf{A}$. The Kronecker and Hadamard, or element-wise, product are denoted by $\otimes$ and $*$, respectively. The column-wise (row-wise) Khatri–Rao products between two matrices, denoted by $\odot$ and $\odot^{\mathrm{T}}$, respectively, are defined as the column-wise (row-wise) Kronecker product. To simplify expressions, the

following shorthand notations are used

$$\bigotimes_{n} \equiv \bigotimes_{n=N}^{1} \qquad \bigotimes_{k\neq n} \equiv \bigotimes_{\substack{k=N \\ k\neq n}}^{1},$$

where N is the order of the tensor and indices are traversed in reverse order. Similar definitions are used for $\odot$, $\odot^{\mathrm{T}}$ and $*$. To reduce the number of parentheses, we assume the matrix product takes precedence over $\otimes$, $\odot$, $\odot^{\mathrm{T}}$ and $*$, e.g., $\mathbf{AB} \odot \mathbf{CD} \equiv (\mathbf{AB}) \odot (\mathbf{CD})$. The complex conjugate, transpose and conjugated transpose are denoted by $\bar{\cdot}$, $\cdot^{\mathrm{T}}$ and $\cdot^{\mathrm{H}}$, respectively. The column-wise concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{x} = [\mathbf{a}; \mathbf{b}]$ and is a shorthand for $\mathbf{x} = \begin{bmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. The inner product between $\mathcal{A}$ and $\mathcal{B}$ is denoted by $\langle \mathcal{A}, \mathcal{B} \rangle = \mathrm{vec}\,(\mathcal{B})^{\mathrm{H}} \mathrm{vec}\,(\mathcal{A})$. The Frobenius norm is denoted by $||\cdot||$. $\mathrm{Re}\,(\cdot)$ returns the real part of a complex number. $\mathbf{I}_I$ is the $I \times I$ identity matrix and $\mathbf{1}_I$ is a length I column vector with ones. Finally, $\mathbf{V}^f$ is the row-wise 'flipped' version of $\mathbf{V} \in \mathbb{K}^{I \times J}$, i.e., $\mathbf{V}^f(i,:) = \mathbf{V}(I - i + 1, :)$ for $i = 1, \ldots, I$. See [65] for more details.

6.2 Exploiting efficient representations

Many tensor decompositions used in signal processing and data analysis can be computed by minimizing the least squares (LS) error between the approximation and the given tensor $\mathcal{T}$

$$\min_{\mathbf{z}} f(\mathbf{z}) \quad \text{with} \quad f(\mathbf{z}) = \frac{1}{2} ||\mathcal{M}(\mathbf{z}) - \mathcal{T}||^2, \tag{6.1}$$

in which $\mathcal{M}(\mathbf{z})$ is a tensor decomposition with variables $\mathbf{z}$. Various algorithms have been proposed to solve (6.1), including alternating least squares (ALS) [44], [56], [82], [142], [176], nonlinear conjugate gradient (NCG) [5], quasi-Newton (qN) [260] and Gauss–Newton (GN) [260], [262], [278] algorithms. We first review the notation for common tensor decompositions in subsection 6.2.1. In subsection 6.2.2 we give an overview of LS optimization theory for algorithms using gradients (first-order information) and Hessian approximations (second-order information) and show that many approximations to the Hessian do not depend on the data $\mathcal{T}$. Hence, second-order algorithms can be used without additional changes in implementation compared to first-order algorithms. Finally, the core operations required to reduce the computational complexity are introduced by rewriting $f(\mathbf{z})$ in subsection 6.2.3.

6.2.1 Overview of tensor decompositions

The techniques derived in this chapter focus on four main decompositions, which can be grouped into two families from an optimization point-of-view:

CPD-based and BTD-based algorithms. Note that we use different families compared to the usual division between decompositions computed using numerical linear algebra techniques such as singular value decompositions (SVD), e.g., MLSVD, TT and hierarchical Tucker, and decompositions usually computed through optimization (CPD, LL1, BTD). Here, we focus mainly on notation; for pointers to uniqueness results and applications, we refer to the overview papers [65], [126], [129], [170], [243].

CPD-based algorithms

The (canonical) polyadic decomposition (CPD) writes an Nth-order tensor as a (minimal) sum of R rank-1 terms, each of which is an outer product, denoted by $\otimes$, of N nonzero factor vectors $\mathbf{a}_r^{(n)}$:

$$\mathcal{M}_{\mathrm{CPD}}(\mathbf{z}) = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} \stackrel{\mathrm{def}}{=} \left[\!\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right].$$

Each factor matrix $\mathbf{A}^{(n)}$ contains $\mathbf{a}_r^{(n)}$, $r = 1, \ldots, R$, as its columns and the variables $\mathbf{z}$ are the vectorized factor matrices $\mathbf{A}^{(n)}$, i.e.,

$$\mathbf{z} = \left[\mathrm{vec}\left(\mathbf{A}^{(1)}\right); \ldots; \mathrm{vec}\left(\mathbf{A}^{(N)}\right) \right].$$

Depending on the field, the CPD is also known as PARAFAC, CANDECOMP or tensor/separation rank decomposition.

The decomposition into multilinear rank-$(L_r, L_r, 1)$ terms (LL1) is accommodated within this family as it is often computed as a constrained CPD [44], [82], [260]:

$$\mathcal{M}_{\mathrm{LL1}}(\mathbf{z}) = \sum_{r=1}^{R} (\mathbf{A}_r \mathbf{B}_r^{\mathrm{T}}) \otimes \mathbf{c}_r = [\![\mathbf{A}, \mathbf{B}, \mathbf{CP}]\!],$$

in which $\mathbf{A}_r \in \mathbb{K}^{I_1 \times L_r}$, $\mathbf{B}_r \in \mathbb{K}^{I_2 \times L_r}$ for $r = 1, \ldots, R$, $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \ldots & \mathbf{A}_R \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \ldots & \mathbf{B}_R \end{bmatrix}$, $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \ldots & \mathbf{c}_R \end{bmatrix}$ and $\mathbf{P}$ is a binary matrix replicating column $\mathbf{c}_r$ L_r times, i.e., $\mathbf{P} = \mathrm{blockdiag}(\mathbf{1}_{L_1}^{\mathrm{T}}, \ldots, \mathbf{1}_{L_R}^{\mathrm{T}})$. Similar to the CPD, $\mathbf{z} = \left[\mathrm{vec}\left(\mathbf{A}\right); \mathrm{vec}\left(\mathbf{B}\right); \mathrm{vec}\left(\mathbf{C}\right) \right]$. The LL1 decomposition is similar to PARALIND [44]. CONFAC generalizes this decomposition by allowing an arbitrary full row rank matrix $\mathbf{P}$ in every mode [11]. In [260], the decomposition into (rank-L_r $\otimes$ rank-1) terms is discussed as one generalization of $\mathcal{M}_{\mathrm{LL1}}$ to higher-order tensors.

BTD-based algorithms

The low-multilinear rank approximation (LMLRA) or Tucker decomposition writes a tensor as a multilinear transformation of a core tensor $\mathcal{S} \in \mathbb{K}^{J_1 \times \cdots \times J_N}$

by the factor matrices $\mathbf{U}^{(n)} \in \mathbb{K}^{I_n \times J_n}$, $n = 1, \ldots, N$:

$$\mathcal{M}_{\text{LMLRA}}(\mathbf{z}) = \mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdots \cdot_N \mathbf{U}^{(N)}.$$

An LMLRA can be computed using SVDs of tensor unfoldings, i.e., as a (truncated) multilinear singular value decomposition (MLSVD) [78], [294], by higher-order orthogonal iteration [79] or through optimization [147], [148], [260], [262].

The more general block term decomposition writes a tensor as a sum of R multilinear rank-$(J_{r1}, J_{r2}, \ldots, J_{rN})$ terms [77]:

$$\mathcal{M}_{\text{BTD}}(\mathbf{z}) = \sum_{r=1}^{R} \mathcal{S}^{(r)} \cdot_1 \mathbf{U}^{(r,1)} \cdots \cdot_N \mathbf{U}^{(r,N)}.$$

From an optimization point-of-view, the difference between $\mathcal{M}_{\text{LMLRA}}$ and $\mathcal{M}_{\text{BTD}}$ is the additional summation over $r = 1, \ldots, R$. For simplicity of notation, we only use $\mathcal{M}_{\text{LMLRA}}$ in the derivations in this chapter.

6.2.2 Optimization for least squares problems

In gradient-based optimization, an initial guess for the variables $\mathbf{z}_0$ is iteratively refined as

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

with the step direction $\mathbf{p}_k$ found by solving

$$\mathbf{H}_k \mathbf{p}_k = -\overline{\mathbf{g}}_k, \tag{6.2}$$

in which $\mathbf{g}_k$ and $\mathbf{H}_k$ are the gradient and the Hessian (or an approximation) of $f(\mathbf{z})$ w.r.t. $\mathbf{z}$ in (6.1), respectively, assuming $\mathcal{M}(\mathbf{z})$ is analytic in $\mathbf{z}$ [258]. The step size α_k ensures sufficient decrease in the objective function value, e.g., through line search. Alternatively, $\mathbf{z}$ can be updated using trust-region approaches or using plane search [209], [257], [260]. The linear system (6.2) can be solved using direct pseudoinversion or using (preconditioned) conjugate gradients [209].

For the LS objective function (6.1), various Hessian approximations are used depending on the algorithm, e.g.,

- $\mathbf{H}_k = \mathbf{I}$ for gradient descent,
- $\mathbf{H}_k = \mathbf{H}_{k-1} + \mathbf{U}_{k-1} + \mathbf{V}_{k-1}$ in which $\mathbf{U}_{k-1}$ and $\mathbf{V}_{k-1}$ are symmetric rank-1 matrices constructed using previous gradients and updates of $\mathbf{z}$ for BFGS,

- $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{H}}\mathbf{J}_k$ with $\mathbf{J}_k = \frac{\partial}{\partial \mathbf{z}}\mathrm{vec}\left(\mathcal{M}(\mathbf{z}) - \mathcal{T}\right)\big|_{\mathbf{z}=\mathbf{z}_k} = \frac{\partial}{\partial \mathbf{z}}\mathrm{vec}\left(\mathcal{M}(\mathbf{z})\right)\big|_{\mathbf{z}=\mathbf{z}_k}$ for Gauss–Newton,
- $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{H}}\mathbf{J}_k + \lambda_k \mathbf{I}$ for Levenberg–Marquardt.

None of these approximations of the Hessian depend on the tensor $\mathcal{T}$. Therefore, only the objective function evaluation and the gradient computation require adaptation to exploit structure in $\mathcal{T}$. Note that this also holds for ALS algorithms.

6.2.3 Exploiting efficient representations

Due to the explicit construction of the residual tensor $\mathcal{F} = \mathcal{M}(\mathbf{z}) - \mathcal{T}$ in (6.1) and the computation of the gradient, the per-iteration complexity is governed by the number of entries in the tensor, i.e., $\mathcal{O}\left(\prod_n I_n\right)$. To reduce the complexity, we assume $\mathcal{T}$ is a function of a limited number of parameters. The parameters can, for instance, be the set of positions and values of nonzero entries in a sparse tensor, the factor matrices of a CPD or the underlying signal in a Hankelized tensor. To exploit the structure in $\mathcal{T}$ and to avoid the creation of the residual tensor, we rewrite the objective function and gradients which reveals the core operations. More concretely, we have

$$f(\mathbf{z}) = \frac{1}{2}\left|\left|\mathcal{M}(\mathbf{z})\right|\right|^2 - \mathrm{Re}\left(\langle \mathcal{M}(\mathbf{z}), \mathcal{T}\rangle\right) + \frac{1}{2}\left|\left|\mathcal{T}\right|\right|^2, \tag{6.3}$$

which requires the efficient computation of the norms of the model and the tensor, and the inner product between the model and the tensor. Similarly, the gradient which is given by $\bar{\mathbf{g}} = \mathbf{J}^{\mathrm{H}}\mathrm{vec}\left(\mathcal{F}\right)$ with $\mathbf{J} = \frac{\partial \mathrm{vec}(\mathcal{F})}{\partial \mathbf{z}}$ the Jacobian matrix, is rewritten by grouping the model and data dependent terms as

$$\begin{aligned}\bar{\mathbf{g}} &= \underbrace{\mathbf{J}^{\mathrm{H}}\mathrm{vec}\left(\mathcal{M}(\mathbf{z})\right)} - \underbrace{\mathbf{J}^{\mathrm{H}}\mathrm{vec}\left(\mathcal{T}\right)} \\ &= \qquad \bar{\mathbf{g}}_{\mathcal{M}} \qquad - \qquad \bar{\mathbf{g}}_{\mathcal{T}}.\end{aligned} \tag{6.4}$$

(In the derivations, we work with $\mathbf{g}$ rather than $\bar{\mathbf{g}}$.) The core operations are the MTKRPROD for the CPD-based algorithms and the MTKRONPROD for the BTD-based algorithms as shown below. Extensions to constrained and coupled constraints conclude this section.

CPD-based algorithms

Let us partition $\mathbf{g}$ according to the variables $\mathbf{z}$, i.e., in the case $\mathcal{M}(\mathbf{z})$ is a CPD, we have

$$\mathbf{g} = \left[\mathrm{vec}\left(\mathbf{G}^{(1)}\right); \ldots; \mathrm{vec}\left(\mathbf{G}^{(N)}\right)\right]$$

with

$$\operatorname{vec}\left(\mathbf{G}^{(n)}\right) = \frac{\partial f}{\partial \operatorname{vec}\left(\mathbf{A}^{(n)}\right)} \qquad n = 1, \ldots, N.$$

By grouping the model and data dependent terms as in (6.4), the gradient terms become

$$\begin{aligned} \mathbf{G}_{\mathcal{M}}^{(n)} &= \bar{\mathbf{A}}^{(n)} \left(\mathop{\ast}_{k \neq n} \mathbf{A}^{(k)^{\mathrm{H}}} \mathbf{A}^{(k)} \right) \\ \mathbf{G}_{\mathcal{T}}^{(n)} &= \bar{\mathbf{T}}_{(n)} \left(\mathop{\odot}_{k \neq n} \mathbf{A}^{(k)} \right). \end{aligned} \tag{6.5}$$

It is clear that $\mathbf{G}_{\mathcal{M}}^{(n)}$ can be computed efficiently after forming the inner products $\mathbf{A}^{(k)^{\mathrm{H}}} \mathbf{A}^{(k)}$, $k = 1, \ldots, N$. We therefore focus on the operation in (6.5) which involves the common matricized tensor times Khatri–Rao product (MTKRPROD).

In the case $\mathcal{M}(\mathbf{z})$ is an LL1 decomposition, $\mathbf{A}^{(3)} = \mathbf{CP}$ is used to form the inner products and to compute the MTKRPROD. From the chain rule follows that for $n = 3$, $\mathbf{G}^{(n)}$ is given by

$$\begin{aligned} \mathbf{G}_{\mathcal{M}}^{(3)} &= \bar{\mathbf{A}}^{(3)} \left(\mathop{\ast}_{k \neq 3} \mathbf{A}^{(k)^{\mathrm{H}}} \mathbf{A}^{(k)} \right) \mathbf{P}^{\mathrm{T}} \\ \mathbf{G}_{\mathcal{T}}^{(3)} &= \bar{\mathbf{T}}_{(3)} \left(\mathop{\odot}_{k \neq 3} \mathbf{A}^{(k)} \right) \mathbf{P}^{\mathrm{T}}. \end{aligned}$$

BTD-based algorithms

Similarly, in the case $\mathcal{M}(\mathbf{z})$ is an LMLRA, the gradient is partitioned as

$$\mathbf{g} = \left[\operatorname{vec}\left(\mathcal{G}^{(0)}\right); \operatorname{vec}\left(\mathbf{G}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{G}^{(N)}\right) \right]$$

with

$$\begin{aligned} \operatorname{vec}\left(\mathcal{G}^{(0)}\right) &= \frac{\partial f}{\partial \operatorname{vec}\left(\mathcal{S}\right)} \\ \operatorname{vec}\left(\mathbf{G}^{(n)}\right) &= \frac{\partial f}{\partial \operatorname{vec}\left(\mathbf{U}^{(n)}\right)}, \qquad n = 1, \ldots, N. \end{aligned}$$

More specifically, the gradient expressions for the core tensors are given by

$$\begin{aligned} \mathcal{G}_{\mathcal{M}}^{(0)} &= \bar{\mathcal{S}} \cdot_1 \left(\mathbf{U}^{(1)^{\mathrm{T}}} \bar{\mathbf{U}}^{(1)} \right) \cdots \cdot_N \left(\mathbf{U}^{(N)^{\mathrm{T}}} \bar{\mathbf{U}}^{(N)} \right) \\ \mathcal{G}_{\mathcal{T}}^{(0)} &= \bar{\mathcal{T}} \cdot_1 \mathbf{U}^{(1)^{\mathrm{T}}} \cdots \cdot_N \mathbf{U}^{(N)^{\mathrm{T}}}, \end{aligned} \tag{6.6}$$

and the expressions for the factor matrices by

$$\mathbf{G}_{\mathcal{M}}^{(n)} = \bar{\mathbf{U}}^{(n)} \bar{\mathbf{S}}_{(n)} \left(\bigotimes_{k \neq n} \mathbf{U}^{(k)^{\mathrm{H}}} \mathbf{U}^{(k)} \right) \mathbf{S}_{(n)}^{\mathrm{T}}$$
$$\mathbf{G}_{\mathcal{T}}^{(n)} = \bar{\mathbf{T}}_{(n)} \left(\bigotimes_{k \neq n} \mathbf{U}^{(k)} \right) \mathbf{S}_{(n)}^{\mathrm{T}} \tag{6.7}$$

for $n = 1, \ldots, N$. In the case of more general BTD with R terms, an additional summation is introduced; see, e.g., [262]. As (6.6) can be written as $\operatorname{vec}\left(\mathcal{G}_{\mathcal{T}}^{(0)}\right)^{\mathrm{T}} = \operatorname{vec}(\mathcal{T})^{\mathrm{H}} \left(\otimes_n \mathbf{U}^{(n)}\right)$, both (6.6) and (6.7) require a vectorized or matricized tensor times Kronecker product (MTKRONPROD). Note that

$$\operatorname{vec}\left(\mathcal{G}_{\mathcal{T}}^{(0)}\right)^{\mathrm{T}} = \operatorname{vec}\left(\mathbf{U}^{(1)^{\mathrm{T}}} \bar{\mathbf{T}}_{(1)} \left(\bigotimes_{k \neq 1} \mathbf{U}^{(k)} \right) \right),$$

hence the MTKRONPROD from (6.7) can be reused, although this may lead to a higher complexity.

Core operations, constraints and coupling

In order to reduce the complexity from $\mathcal{O}$ (tensor entries) to $\mathcal{O}$ (parameters), structure exploiting implementations of the Frobenius norm and inner products with CPD and BTD models are required to evaluate the objective function (6.3) as well as implementations for the MTKRPROD and the MTKRONPROD for CPD and BTD gradients, respectively. Examples of such implementations are discussed in section 6.3. In contrast to approaches taken in, e.g., [46], [69], the original factor matrices and core tensors remain the optimization variables. Therefore, algorithms for coupling datasets, e.g., [3], [262], only require new implementations of the core operations. Similarly, parametric constraints such as nonnegativity, orthogonality or Vandermonde structure can be handled using the chain rule as in [262], as well as other types of constraints involving projecting onto a feasible set; see, e.g., [145], [160]. Concrete examples are given in the experiments in section 6.4.

6.3 Operations on efficient representations

In the previous section, the computation of the norm, inner product, MTKRPROD and MTKRONPROD have been identified as the four key operations to allow tensor decomposition algorithms to work on structured tensors. In this section, we show for a number of efficiently represented tensors that the explicit construction of the full tensor is not required and that the complexity is governed by the number of parameters in the efficient representation, rather than the number of entries in the tensor. First, the polyadic and Tucker for-

mat are discussed as expressions involving one of these formats occur in all of the discussed decompositions. Then, efficient implementations for the TT format and two implicit tensorizations — Hankelization and Löwnerization — are derived. These and other implementations are available in Tensorlab [305].

Unless stated otherwise, only third-order tensors $\mathcal{T} \in \mathbb{K}^{I \times I \times I}$ are considered to simplify notations and complexity expressions for the two computational families: the rank-R CPD $\mathcal{M}_{\text{CPD}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and the multilinear rank-(R, R, R) LMLRA $\mathcal{M}_{\text{LMLRA}} = \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$. The extension to a BTD introduces additional summations and is omitted here; see [262]. For the MTKRPROD and MTKRONPROD operations, only the case $n = 1$ is shown, unless the expressions do not generalize trivially for $n = 2, \ldots, N$. Table 6.1 gives a summary of the computational per-iteration complexity when computing a rank-R CPD and clearly illustrates independence on the total number of entries I^N.

Table 6.1: Computational per-iteration complexity when computing a rank-R CPD of an Nth-order $I \times \cdots \times I$ tensor given in its efficient representation. *For Hankelization and Löwnerization, second-order tensorization of K signals is assumed, resulting in a tensor of size $I \times I \times K$. The number of samples is $M = 2I - 1$ for Hankel and $M = 2I$ for Löwner.*

Structure	Function	Complexity
Dense	Objective	$\mathcal{O}\left(RI^N\right)$
	Gradient	$\mathcal{O}\left(NRI^N\right)$
CPD	Parameters	NIF
	Objective	$\mathcal{O}\left(NIFR + NIR^2\right)$
	Gradient	$\mathcal{O}\left(NIFR + NIR^2\right)$
LMLRA	Parameters	$\mathcal{O}\left(NIJ + J^N\right)$
	Objective	$\mathcal{O}\left(NIJR + J^N R + NIR^2\right)$
	Gradient	$\mathcal{O}\left(NIJR + J^N NR + NIR^2\right)$
TT	Parameters	$\mathcal{O}\left(NIr^2\right)$
	Objective	$\mathcal{O}\left(NIr^2R + NIR^2\right)$
	Gradient	$\mathcal{O}\left(NIr^2R + NIR^2\right)$
Hankel	Parameters	MK
	Objective	$\mathcal{O}\left(M(K + \log_2 M)R + (M + K)R^2\right)$
	Gradient	$\mathcal{O}\left(M(K + \log_2 M)R + (M + K)R^2\right)$
Löwner	Parameters	$M(K + 1)$
	Objective	$\mathcal{O}\left(M(K + \log_2 M)R + (M + K)R^2\right)$
	Gradient	$\mathcal{O}\left(M(K + \log_2 M)R + (M + K)R^2\right)$

6.3.1 Polyadic format

In the polyadic format, a third-order tensor $\mathcal{T}$ is given by a set of factor matrices $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ with F columns, i.e., $\mathcal{T} = [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!]$. The number of variables is therefore $F \sum_n I_n$ instead of $\prod_n I_n$. Efficient implementations in the case $\mathcal{T}$ admits a PD, with F not necessary equal to R, have been presented in [16] (see Kruskal tensor) and are here extended to complex data:

$$\begin{aligned} ||\mathcal{T}||^2 &= \mathbf{1}^{\mathrm{T}} \left(\mathbf{X}^{\mathrm{H}}\mathbf{X} * \mathbf{Y}^{\mathrm{H}}\mathbf{Y} * \mathbf{Z}^{\mathrm{H}}\mathbf{Z}\right) \mathbf{1} \\ \langle \mathcal{M}_{\mathrm{CPD}}, \mathcal{T} \rangle &= \mathbf{1}^{\mathrm{T}} \left(\mathbf{X}^{\mathrm{H}}\mathbf{A} * \mathbf{Y}^{\mathrm{H}}\mathbf{B} * \mathbf{Z}^{\mathrm{H}}\mathbf{C}\right) \mathbf{1} \\ \bar{\mathbf{T}}_{(1)}(\mathbf{C} \odot \mathbf{B}) &= \bar{\mathbf{X}} \left(\mathbf{Y}^{\mathrm{H}}\mathbf{B} * \mathbf{Z}^{\mathrm{H}}\mathbf{C}\right). \end{aligned} \tag{6.8}$$

The complexity of these operations is governed by the construction of the inner products which require $\mathcal{O}\left(NIF^2\right)$ and $\mathcal{O}\left(NIFR\right)$ flop. Each MTKRPROD in (6.8) required for $\mathbf{G}_{\mathcal{T}}^{(n)}$ can then be computed using $\mathcal{O}\left(IFR\right)$ flop.

To compute an LMLRA, the inner product with an LMLRA

$$\langle \mathcal{M}_{\mathrm{LMLRA}}, \mathcal{T} \rangle = \mathrm{trace}\left(\mathcal{S} \cdot_1 \mathbf{X}^{\mathrm{H}}\mathbf{U} \cdot_2 \mathbf{Y}^{\mathrm{H}}\mathbf{V} \cdot_3 \mathbf{Z}^{\mathrm{H}}\mathbf{W}\right) \tag{6.9}$$

and the MTKRONPROD operations are required:

$$\bar{\mathbf{T}}_{(1)}(\mathbf{W} \otimes \mathbf{V}) = \bar{\mathbf{X}} \left(\mathbf{W}^{\mathrm{H}}\mathbf{Z} \odot \mathbf{V}^{\mathrm{H}}\mathbf{Y}\right)^{\mathrm{H}} \tag{6.10}$$

$$\mathrm{vec}\left(\mathcal{T}\right)^{\mathrm{H}} \left(\mathbf{W} \otimes \mathbf{V} \otimes \mathbf{U}\right) = \mathrm{vec}\left([\![\mathbf{U}^{\mathrm{H}}\mathbf{X}, \mathbf{V}^{\mathrm{H}}\mathbf{Y}, \mathbf{W}^{\mathrm{H}}\mathbf{Z}]\!]\right)^{\mathrm{H}}. \tag{6.11}$$

The computation of the trace in (6.9) requires only the diagonal entries of the LMLRA and takes $\mathcal{O}\left(NIFR + FR^N\right)$ flop, including the construction of the inner products. To compute the actual gradient $\mathbf{G}_{\mathcal{T}}^{(1)}$, the result of (6.10) is multiplied by $\mathbf{S}_{(1)}^{\mathrm{T}}$, requiring the product $\left(\mathbf{W}^{\mathrm{H}}\mathbf{Z} \odot \mathbf{V}^{\mathrm{H}}\mathbf{Y}\right)^{\mathrm{H}} \mathbf{S}_{(1)}^{\mathrm{T}}$ which is a transposed MTKRPROD involving usually small matrices. This product can be computed efficiently using, e.g., [224], [293] and requires $\mathcal{O}\left(NFR^N + NIFR\right)$ flop. The complexity of (6.11) is the same.

6.3.2 Tucker format

In the Tucker format a third-order tensor $\mathcal{T} = \mathcal{Q} \cdot_1 \mathbf{X} \cdot_2 \mathbf{Y} \cdot_3 \mathbf{Z}$ is defined by a third-order core tensor $\mathcal{Q} \in \mathbb{K}^{J_1 \times J_2 \times J_3}$ and the factor matrices $\mathbf{X} \in \mathbb{K}^{I \times J_1}$, $\mathbf{Y} \in \mathbb{K}^{I \times J_2}$, $\mathbf{Z} \in \mathbb{K}^{I \times J_3}$. The Tucker format often follows from the computation of an LMLRA or an MLSVD, e.g., as the result of a compression step [46]. For the complexity analysis, we assume $J_n = J$ for $n = 1, \ldots, N$. We assume that the factor matrices have orthonormal columns, which is always possible through normalization.

The norm of $\mathcal{T}$ is computed in $\mathcal{O}\left(J^N\right)$ operations as

$$||\mathcal{T}||^2 = ||\mathcal{Q}||^2 .$$

The inner product with $\mathcal{M}_{\text{CPD}}$ and MTKRPROD is similar to (6.9) and (6.10):

$$\begin{aligned}\langle \mathcal{M}_{\text{CPD}}, \mathcal{T} \rangle &= \text{trace}\left(\bar{\mathcal{Q}} \cdot_1 \mathbf{A}^{\text{T}}\bar{\mathbf{X}} \cdot_2 \mathbf{B}^{\text{T}}\bar{\mathbf{Y}} \cdot_3 \mathbf{C}^{\text{T}}\bar{\mathbf{Z}}\right) \\ \bar{\mathbf{T}}_{(1)}(\mathbf{C} \odot \mathbf{B}) &= \bar{\mathbf{X}}\bar{\mathbf{Q}}_{(1)}\left(\mathbf{Z}^{\text{H}}\mathbf{C} \odot \mathbf{Y}^{\text{H}}\mathbf{B}\right)\end{aligned}$$

and both require $\mathcal{O}\left(NIJR + J^N R\right)$ flop.

To compute an LMLRA, the following expressions for the inner product and the MTKRONPROD in (6.7) can be used

$$\begin{aligned}\langle \mathcal{M}_{\text{LMLRA}}, \mathcal{T} \rangle &= \langle \mathcal{S}, \mathcal{Q} \cdot_1 \mathbf{U}^{\text{H}}\mathbf{X} \cdot_2 \mathbf{V}^{\text{H}}\mathbf{Y} \cdot_3 \mathbf{W}^{\text{H}}\mathbf{Z} \rangle \\ \bar{\mathbf{T}}_{(1)}(\mathbf{W} \otimes \mathbf{V}) &= \bar{\mathbf{X}}\bar{\mathbf{Q}}_{(1)}\left(\mathbf{Z}^{\text{H}}\mathbf{W} \otimes \mathbf{Y}^{\text{H}}\mathbf{V}\right).\end{aligned}$$

Similarly, the MTKRONPROD in (6.6) is computed as

$$\text{vec}(\mathcal{T})^{\text{H}}(\mathbf{W} \otimes \mathbf{V} \otimes \mathbf{U}) = \text{vec}\left(\mathcal{Q} \cdot_1 \mathbf{U}^{\text{H}}\mathbf{X} \cdot_2 \mathbf{V}^{\text{H}}\mathbf{Y} \cdot_3 \mathbf{W}^{\text{H}}\mathbf{Z}\right)^{\text{H}} .$$

All computations involving the LMLRA require $\mathcal{O}\left(NIJR + J^N R + JR^N\right)$ flop.

6.3.3 Tensor train format

An Nth-order tensor $\mathcal{T} \in \mathbb{K}^{I_1 \times \cdots \times I_N}$ can be represented by N core matrices or tensors $\mathcal{Q}^{(n)} \in \mathbb{K}^{r_{n-1} \times I_n \times r_n}$ serially linked by $N-1$ indices such that

$$\begin{aligned}t_{i_1,\ldots,i_N} &= \sum_{s_1=1}^{r_1} \cdots \sum_{s_{N-1}=1}^{r_{N-1}} q^{(1)}_{1,i_1,s_1} q^{(2)}_{s_1,i_2,s_2} \cdots q^{(N)}_{s_{N-1},i_N,1}, \quad \text{or} \\ \mathcal{T} &= \text{tt}\left(\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}\right).\end{aligned}$$

The parameters r_n, $n = 0, \ldots, N$, are the compression ranks with $r_0 = r_N = 1$. This formulation is known as matrix product states [210] in computational chemistry and as tensor trains (TT) [211] in scientific computing and can be computed using SVDs, cross approximation or optimization [211]–[213], [217]. Assuming all dimensions and compression ranks are equal, i.e., $r_n = r$ for $n = 1, \ldots, N-1$, the total number of parameters is $\mathcal{O}\left(NIr^2\right)$.

The squared norm of $\mathcal{T}$ is given by $||\mathcal{T}||^2 = |\mathbf{F}^{(N)}|$, in which the scalar

$\mathbf{F}^{(N)}$ is constructed using the recursive formula

$$\begin{aligned}\mathbf{F}^{(0)} &= 1\\ \mathbf{F}^{(n)} &= \left(\mathbf{Q}^{(n)} \bullet_1 \mathbf{F}^{(n-1)}\right)^{\mathrm{H}}_{(1,2)} \mathbf{Q}^{(n)}_{(1,2)} \qquad n = 1, \ldots, N.\end{aligned}$$

This way $||\mathcal{T}||^2$ can be computed in $\mathcal{O}\left(NIr^3\right)$ flop. While this recursive formula is essentially identical to the formula derived in [217], it reveals the inner products more clearly, allowing efficient implementations.

To compute inner products with $\mathcal{M}_{\mathrm{CPD}}$ and $\mathcal{M}_{\mathrm{LMLRA}}$, we first introduce the auxiliary core tensors $\hat{\mathcal{Q}}^{(n)}$. Let $\mathbf{A}^{(1)} = \mathbf{A}$, $\mathbf{A}^{(2)} = \mathbf{B}$ and $\mathbf{A}^{(3)} = \mathbf{C}$ for $\mathcal{M}_{\mathrm{CPD}}$, and $\mathbf{A}^{(1)} = \mathbf{U}$, $\mathbf{A}^{(2)} = \mathbf{V}$ and $\mathbf{A}^{(3)} = \mathbf{W}$ for $\mathcal{M}_{\mathrm{LMLRA}}$, then $\hat{\mathcal{Q}}^{(n)}$ is computed using $\mathcal{O}\left(Ir^2R\right)$ flop as

$$\hat{\mathcal{Q}}^{(n)} = \bar{\mathcal{Q}}^{(n)} \bullet_2 \mathbf{A}^{(n)^{\mathrm{T}}}.$$

The inner products are then computed as

$$\langle \mathcal{M}_{\mathrm{CPD}}, \mathcal{T} \rangle = \mathrm{trace}\left(\mathrm{tt}(\hat{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})\right) \tag{6.12}$$

$$\langle \mathcal{M}_{\mathrm{LMLRA}}, \mathcal{T} \rangle = \left\langle \mathrm{tt}(\hat{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)}), \bar{\mathcal{S}} \right\rangle. \tag{6.13}$$

As only the diagonal entries of $\mathrm{tt}(\hat{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})$ are required to compute the trace in (6.12), the construction of the $R \times \cdots \times R$ tensor can be avoided. The construction of the diagonal entries in (6.12) and the inner product in (6.13) require $\mathcal{O}\left(NRr^2\right)$ and $\mathcal{O}\left(R^N r^2\right)$ flop, respectively.

To compute an MTKRPROD, the auxiliary cores $\hat{\mathcal{Q}}^{(n)}$, $n = 1, \ldots, N$ are used again. Define $\mathcal{P}^{(1)} = \mathrm{tt}(\bar{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})$ and $\mathbf{p}_r$ as the mode-1 vectors $\mathcal{P}^{(1)}(:, r, r)$, $r = 1, \ldots, R$, then

$$\bar{\mathbf{T}}_{(1)}(\mathbf{C} \odot \mathbf{B}) = \begin{bmatrix}\mathbf{p}_1^{(1)} & \mathbf{p}_2^{(1)} & \cdots & \mathbf{p}_R^{(1)}\end{bmatrix}.$$

The complexity of computing one MTKRPROD is therefore $\mathcal{O}\left(Ir^2R\right)$, assuming auxiliary cores have been constructed.

Similarly, the MTKRONPROD can be formed efficiently as

$$\bar{\mathbf{T}}_{(1)}(\mathbf{W} \otimes \mathbf{V}) = \left(\mathrm{tt}(\bar{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})\right)_{(1)}.$$

In order to compute the gradient terms $\mathbf{G}^{(n)}_{\mathcal{T}}$ in (6.7) the matrix unfolding $\left(\mathrm{tt}(\bar{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})\right)_{(1)}$ can be constructed and multiplied with $\mathbf{S}^{\mathrm{T}}_{(1)}$, which requires $\mathcal{O}\left(IR^N + IR^{N-1}r^2\right)$ flop. However, by exploiting the TT structure, the product can be computed directly in $\mathcal{O}\left(rR^N + r^2R^{N-1} + Ir^2R\right)$ flop.

The precise implementation is out of the scope of this chapter. Finally, to compute $\mathcal{G}_{\mathcal{T}}^{(0)}$ in $\mathcal{O}\left(r^2 R^N\right)$ flop, we can use

$$\operatorname{vec}(\mathcal{T})^{\mathrm{H}}(\mathbf{W} \otimes \mathbf{V} \otimes \mathbf{U}) = \operatorname{vec}\left(\operatorname{tt}(\hat{\mathcal{Q}}^{(1)}, \hat{\mathcal{Q}}^{(2)}, \hat{\mathcal{Q}}^{(3)})\right)^{\mathrm{T}}.$$

6.3.4 Implicit Hankelization

Hankelization of a (possibly complex) signal vector $\mathbf{m} \in \mathbb{K}^M$ of length $M = \sum_{n=1}^{N} I_n - N + 1$ yields an $I_1 \times I_2$ matrix or $I_1 \times \cdots \times I_N$ tensor with constant anti-diagonals or constant anti-diagonal hyperplanes for $N = 2$ and $N > 2$, respectively [85], [221]. In blind source separation, a set of K vectors $\mathbf{m}_k \in \mathbb{K}^M$ is often given [71], [85]. Each vector $\mathbf{m}_k$ can be Hankelized separately and the resulting matrices or tensors can be concatenated along the $(N+1)$th mode. To simplify the expressions[1], we only consider second-order Hankelization and assume $I_1 = I_2 = I$. Let us stack the vectors $\mathbf{m}_k$ as the columns of $\mathbf{M}$. Each entry of the $I \times I \times K$ tensor $\mathcal{T}$ is then given by $t_{ijk} = m_{i+j-1,k}$. As multiplying a Hankel matrix by a vector corresponds to a convolution, fast Fourier transforms (FFT) can be used to speed up computations [15], [93]. Let $\mathfrak{F}$ denote the M-point FFT and $\mathfrak{F}^{-1}$ the inverse M-point FFT. Only the last I rows of the inverse M-point FFT are retained when the operator $\underline{\mathfrak{F}}^{-1}$ is used.

The squared norm can be computed in $\mathcal{O}(MK)$ flop as

$$||\mathcal{T}||^2 = \mathbf{w}^{\mathrm{T}}(\mathbf{M} * \overline{\mathbf{M}})\mathbf{1}_K,$$

in which $\mathbf{w}$ is a vector that contains the number of occurrences of each entry of $\mathbf{M}$ in $\mathcal{T}$ and can be computed as $\mathbf{w} = \operatorname{convolution}(\mathbf{1}_I, \mathbf{1}_I)$. The inner products can be computed as

$$\begin{aligned}\langle \mathcal{M}_{\mathrm{CPD}}, \mathcal{T} \rangle &= \left\langle \mathfrak{F}^{-1}(\mathfrak{F}\mathbf{A} * \mathfrak{F}\mathbf{B})\mathbf{C}^{\mathrm{T}}, \mathbf{M} \right\rangle \\ \langle \mathcal{M}_{\mathrm{LMLRA}}, \mathcal{T} \rangle &= \left\langle \mathfrak{F}^{-1}\left((\mathfrak{F}\mathbf{V} \odot^{\mathrm{T}} \mathfrak{F}\mathbf{U})\mathbf{S}_{(1,2)}\mathbf{W}^{\mathrm{T}}\right), \mathbf{M} \right\rangle.\end{aligned}$$

The computational cost for the inner products is $\mathcal{O}(RM(K + \log_2 M))$ and $\mathcal{O}\left((R+K)M \log_2 M + MR^3\right)$ flop in the case of $\mathcal{M}_{\mathrm{CPD}}$ and $\mathcal{M}_{\mathrm{LMLRA}}$, respectively.

To compute the MTKRPROD, the Hankel structure can be exploited as

$$\begin{aligned}\bar{\mathbf{T}}_{(1)}(\mathbf{C} \odot \mathbf{B}) &= \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\bar{\mathbf{M}}\mathbf{C} * \mathfrak{F}\mathbf{B}^f\right) \\ \bar{\mathbf{T}}_{(2)}(\mathbf{C} \odot \mathbf{A}) &= \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\bar{\mathbf{M}}\mathbf{C} * \mathfrak{F}\mathbf{A}^f\right) \\ \bar{\mathbf{T}}_{(3)}(\mathbf{B} \odot \mathbf{A}) &= \mathbf{M}^{\mathrm{H}}\mathfrak{F}^{-1}\left(\mathfrak{F}\mathbf{A} * \mathfrak{F}\mathbf{B}\right)\end{aligned}$$

[1] More general implementations are available in Tensorlab 3.0 [305].

requiring $\mathcal{O}\left(RM(K+\log_2 M)\right)$ flop each. ($\mathbf{A}^f$ and $\mathbf{B}^f$ are the flipped versions of $\mathbf{A}$ and $\mathbf{B}$; see subsection 6.1.3.)

The MTKRONPROD in $\mathbf{G}_{\mathcal{T}}^{(n)}$ in (6.7) can be computed as

$$\begin{aligned}\bar{\mathbf{T}}_{(1)}(\mathbf{W}\otimes\mathbf{V}) &= \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\bar{\mathbf{M}}\mathbf{W}\odot^{\mathrm{T}}\mathfrak{F}\mathbf{V}^f\right)\\ \bar{\mathbf{T}}_{(2)}(\mathbf{W}\otimes\mathbf{U}) &= \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\bar{\mathbf{M}}\mathbf{W}\odot^{\mathrm{T}}\mathfrak{F}\mathbf{U}^f\right)\\ \bar{\mathbf{T}}_{(3)}(\mathbf{V}\otimes\mathbf{U}) &= \mathbf{M}^{\mathrm{H}}\mathfrak{F}^{-1}\left(\mathfrak{F}\mathbf{V}\odot^{\mathrm{T}}\mathfrak{F}\mathbf{U}\right)\end{aligned}$$

using $\mathcal{O}\left(R^2M\log_2 M + RMK\right)$ flop. The multiplication with $\mathbf{S}_{(n)}^{\mathrm{T}}$ in (6.7) additionally takes $\mathcal{O}\left(IR^3\right)$ flop for $n=1,2$ and $\mathcal{O}\left(KR^3\right)$ flop for $n=3$. However, by first computing the multiplication of the row-wise Khatri–Rao product with $\mathbf{S}_{(n)}^{\mathrm{T}}$, i.e., before performing the inverse FFT, the total cost can be reduced to $\mathcal{O}\left(RM(K+\log_2 M)+MR^3\right)$. Finally, the MTKRONPROD in all modes can be computed using

$$\mathrm{vec}(\mathcal{T})^{\mathrm{H}}(\mathbf{W}\otimes\mathbf{V}\otimes\mathbf{U}) = \mathrm{vec}\left((\mathfrak{F}^{-1}(\mathfrak{F}\mathbf{V}\odot^{\mathrm{T}}\mathfrak{F}\mathbf{U}))^{\mathrm{T}}\bar{\mathbf{M}}\mathbf{W}\right)^{\mathrm{T}}$$

in $\mathcal{O}\left(R^2M\log_2 M + R^3M + RMK\right)$ flop.

6.3.5 Implicit Löwnerization

Löwner matrices and tensors have attractive properties for applications involving rational functions [85], [90]. Given a function $h:\mathbb{K}\to\mathbb{K}$ evaluated at M points $t_i\in T=\{t_1,\ldots,t_M\}$, we partition T in two disjoint point sets $X=\{x_1,\ldots,x_I\}$ and $Y=\{y_1,\ldots,y_J\}$ such that $T=X\cup Y$ and $M=I+J$. The entries in a Löwner matrix $\mathbf{L}\in\mathbb{K}^{I\times J}$ are then given by

$$l_{ij}=\frac{h(x_i)-h(y_j)}{x_i-y_j},\quad i=1,\ldots,I,\ \text{and}\ j=1,\ldots,J.$$

While higher-order generalizations of the Löwner transformation exist [85], [86], we focus on third-order tensors in which each kth frontal slice is a Löwner matrix constructed from a function h_k evaluated in the points in X and Y. Let $\mathbf{P}\in\mathbb{K}^{I\times K}$ and $\mathbf{Q}\in\mathbb{K}^{J\times K}$ contain sampled function values at the points in X and Y, respectively, i.e.,

$$\begin{aligned}p_{ik}&=h_k(x_i),\quad i=1,\ldots,I,\ \text{and}\ k=1,\ldots,K\\ q_{jk}&=h_k(y_j),\quad j=1,\ldots,J,\ \text{and}\ k=1,\ldots,K.\end{aligned}$$

The tensor $\mathcal{T}\in\mathbb{K}^{I\times J\times K}$ is then fully determined by $\mathbf{P}$, $\mathbf{Q}$, X and Y. To simplify complexity expressions, we take $I=J$. Each frontal slice $\mathbf{T}_k$ can be

written as

$$\mathbf{T}_k = \mathrm{Diag}(\mathbf{p}_k)\mathbf{M} - \mathbf{M}\mathrm{Diag}(\mathbf{q}_k), \quad k = 1, \ldots, K,$$

in which $\mathbf{M}$ is a Cauchy matrix with

$$m_{ij} = \frac{1}{x_i - y_j}, \quad 1 \leq i, j \leq I.$$

By assuming that all points in X and Y are equidistant, $\mathbf{M}$ is also a Toeplitz matrix. As multiplying a Toeplitz matrix with a vector $\mathbf{x}$ can be seen as a convolution of the generating vector $\mathbf{v} \in \mathbb{K}^{2I-1}$ with $\mathbf{x}$, in which

$$\mathbf{v} = \left[m_{1,I}; m_{1,I-1}; \ldots; m_{1,1}; m_{2,1}; \ldots; m_{I,1}\right],$$

it is not necessary to construct $\mathbf{M}$ and the multiplication can be performed using FFTs. Let $\mathfrak{F}$ be the $(2I-1)$-point FFT with zero padding for shorter vectors, $\mathfrak{F}^{-1}$ the inverse $(2I-1)$-point FFT and define $\underline{\mathfrak{F}}^{-1}$ to be the last I rows of the result of the inverse FFT. For a matrix $\mathbf{X} \in \mathbb{K}^{I \times K}$, $\mathbf{MX}$ and $\mathbf{M}^{\mathrm{T}}\mathbf{X}$ can then be computed as

$$\mathbf{MX} = \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\mathbf{v}\mathbf{1}_K^{\mathrm{T}} * \mathfrak{F}\mathbf{X}\right) \tag{6.14}$$

$$\mathbf{M}^{\mathrm{T}}\mathbf{X} = \underline{\mathfrak{F}}^{-1}\left(\mathfrak{F}\mathbf{v}^f\mathbf{1}_K^{\mathrm{T}} * \mathfrak{F}\mathbf{X}\right). \tag{6.15}$$

If $\mathfrak{F}\mathbf{v}$ is precomputed, the cost of this multiplication is $\mathcal{O}\left(2KI + 4KI\log_2 2I\right)$ flop.

We now define the different operations for Löwner tensors. As the unfoldings $\mathbf{T}_{(n)}$, $n = 1, 2, 3$, are given by

$$\begin{aligned}
\mathbf{T}_{(1)} &= \mathbf{P} \odot^{\mathrm{T}} \mathbf{M} - \mathbf{M}\left(\mathbf{Q} \odot^{\mathrm{T}} \mathbf{I}_J\right) \\
\mathbf{T}_{(2)} &= \mathbf{M}^{\mathrm{T}}\left(\mathbf{P} \odot^{\mathrm{T}} \mathbf{I}_I\right) - \left(\mathbf{Q} \odot^{\mathrm{T}} \mathbf{M}^{\mathrm{T}}\right) \\
\mathbf{T}_{(3)} &= \mathbf{P}^{\mathrm{T}}\left(\mathbf{M} \odot^{\mathrm{T}} \mathbf{I}_I\right) - \mathbf{Q}^{\mathrm{T}}\left(\mathbf{I}_J \odot^{\mathrm{T}} \mathbf{M}^{\mathrm{T}}\right),
\end{aligned}$$

the expressions for the MTKRPROD and MTKRONPROD can be derived easily using multilinear algebra identities. The MTKRPROD can be computed efficiently as

$$\begin{aligned}
\bar{\mathbf{T}}_{(1)}\left(\mathbf{C} \odot \mathbf{B}\right) &= \bar{\mathbf{P}}\mathbf{C} * \bar{\mathbf{M}}\mathbf{B} - \bar{\mathbf{M}}\left(\bar{\mathbf{Q}}\mathbf{C} * \mathbf{B}\right) \\
\bar{\mathbf{T}}_{(2)}\left(\mathbf{C} \odot \mathbf{A}\right) &= \mathbf{M}^{\mathrm{H}}\left(\bar{\mathbf{P}}\mathbf{C} * \mathbf{A}\right) - \bar{\mathbf{Q}}\mathbf{C} * \mathbf{M}^{\mathrm{H}}\mathbf{A} \\
\bar{\mathbf{T}}_{(3)}\left(\mathbf{B} \odot \mathbf{A}\right) &= \mathbf{P}^{\mathrm{H}}\left(\bar{\mathbf{M}}\mathbf{B} * \mathbf{A}\right) - \mathbf{Q}^{\mathrm{H}}\left(\mathbf{B} * \mathbf{M}^{\mathrm{H}}\mathbf{A}\right).
\end{aligned}$$

As each MTKRPROD requires two FFT-based multiplications with $\mathbf{M}$ and two regular matrix-matrix multiplications with $\mathbf{P}$ and $\mathbf{Q}$, the complexity is $\mathcal{O}\left(RI(K + \log_2 I)\right)$. Similarly, the MTKRONPROD in (6.7) can be computed

as

$$\begin{aligned}\bar{\mathbf{T}}_{(1)}\left(\mathbf{W}\otimes\mathbf{V}\right) &= \bar{\mathbf{P}}\mathbf{W}\odot^{\mathrm{T}}\bar{\mathbf{M}}\mathbf{V}-\bar{\mathbf{M}}\left(\bar{\mathbf{Q}}\mathbf{W}\odot^{\mathrm{T}}\mathbf{V}\right)\\ \bar{\mathbf{T}}_{(2)}\left(\mathbf{W}\otimes\mathbf{U}\right) &= \mathbf{M}^{\mathrm{H}}\left(\bar{\mathbf{P}}\mathbf{W}\odot^{\mathrm{T}}\mathbf{U}\right)-\bar{\mathbf{Q}}\mathbf{W}\odot^{\mathrm{T}}\mathbf{M}^{\mathrm{H}}\mathbf{U}\\ \bar{\mathbf{T}}_{(3)}\left(\mathbf{V}\otimes\mathbf{U}\right) &= \mathbf{P}^{\mathrm{H}}\left(\bar{\mathbf{M}}\mathbf{V}\odot^{\mathrm{T}}\mathbf{U}\right)-\mathbf{Q}^{\mathrm{H}}\left(\mathbf{V}\odot^{\mathrm{T}}\mathbf{M}^{\mathrm{H}}\mathbf{U}\right),\end{aligned}$$

in $\mathcal{O}\left(R^2I+R^2I\log_2 I+RKI\right)$ flop for $n=1,2$ and $\mathcal{O}\left(RI\log_2 I+R^2IK\right)$ flop for $n=3$. This can be reduced further to $\mathcal{O}\left(RI\log_2 I+RKI\right)$ flop for $n=1,2$, by exploiting the row-wise Khatri–Rao products when computing the subsequent multiplication with $\mathbf{S}_{(n)}^{\mathrm{T}}$, which costs $\mathcal{O}\left(IR^3\right)$. A similar improvement can be made for $n=3$. The MTKRONPROD in (6.6) is computed as

$$\mathrm{vec}\left(\mathcal{T}\right)^{\mathrm{H}}\left(\mathbf{W}\otimes\mathbf{V}\otimes\mathbf{U}\right)=\mathrm{vec}\left(\mathbf{U}^{\mathrm{T}}\left(\bar{\mathbf{P}}\mathbf{W}\odot^{\mathrm{T}}\bar{\mathbf{M}}\mathbf{V}\right)-\left(\mathbf{M}^{\mathrm{H}}\mathbf{U}\right)^{\mathrm{T}}\left(\bar{\mathbf{Q}}\mathbf{W}\odot^{\mathrm{T}}\mathbf{V}\right)\right)^{\mathrm{T}}$$

and requires $\mathcal{O}\left(RI\log_2 I+R^3I\right)$ flop.

The squared Frobenius norm is computed as

$$\begin{aligned}\left|\left|\mathcal{T}\right|\right|^2 = {}& \mathbf{1}_I^{\mathrm{T}}(\mathbf{M}*\overline{\mathbf{M}})^{\mathrm{T}}\left((\mathbf{P}*\overline{\mathbf{P}})\mathbf{1}_K\right)\\ &+\mathbf{1}_I^{\mathrm{T}}(\mathbf{M}*\overline{\mathbf{M}})\left((\mathbf{Q}*\overline{\mathbf{Q}})\mathbf{1}_K\right)\\ &-2\mathrm{Re}\left(\mathbf{1}_K^{\mathrm{T}}((\mathbf{M}*\overline{\mathbf{M}})^{\mathrm{T}}\mathbf{P}*\overline{\mathbf{Q}})\mathbf{1}_K\right).\end{aligned}$$

As only $K+2$ FFTs are required, the total cost is $\mathcal{O}\left(4IK\log_2 2I\right)$ flop. To compute the inner products with $\mathcal{M}_{\mathrm{CPD}}$ and $\mathcal{M}_{\mathrm{LMLRA}}$, the results from MTKRPROD and MTKRONPROD are reused:

$$\begin{aligned}\left\langle\mathcal{M}_{\mathrm{CPD}},\mathcal{T}\right\rangle &= \left\langle\bar{\mathbf{T}}_{(3)}\left(\mathbf{B}\odot\mathbf{A}\right),\bar{\mathbf{C}}\right\rangle\\ \left\langle\mathcal{M}_{\mathrm{LMLRA}},\mathcal{T}\right\rangle &= \left\langle\mathbf{X}^{\mathrm{T}}\bar{\mathbf{T}}_{(1)}\left(\mathbf{W}\otimes\mathbf{V}\right),\bar{\mathbf{S}}_{(1)}\right\rangle.\end{aligned}$$

Apart from the cost of the MTKRPROD and the MTKRONPROD, an additional cost of $\mathcal{O}\left(KR\right)$ and $\mathcal{O}\left(R^N\right)$ flop is incurred when computing $\left\langle\mathcal{M}_{\mathrm{CPD}},\mathcal{T}\right\rangle$ and $\left\langle\mathcal{M}_{\mathrm{LMLRA}},\mathcal{T}\right\rangle$, respectively.

In the case the points sets X or Y do not contain equidistant points, $\mathbf{M}$ no longer admits a Toeplitz structure. By exploiting the low displacement rank of Cauchy matrices, it is again not necessary to explicitly construct the tensor. The computational complexity of the multiplications with $\mathbf{M}$ in (6.14) and (6.15) is slightly increased to $\mathcal{O}\left(4KI\log_2^2 2I\right)$ flop [117], [118].

6.4 Experiments

The following experiments illustrate the scalability of constrained CPD, LL1 and BTD algorithms that exploit efficient representations of tensors and the

effect on the accuracy of the results. The latter is discussed in subsection 6.4.1 for ill-conditioned problems and in subsection 6.4.3 for Hankelized signals. The former is illustrated by computing the nonnegative CPD of a compressed tensor in subsection 6.4.2 and for the unconstrained LL1 decomposition and the constrained BTD in subsection 6.4.3. The CPD error E_{CPD} is defined as

$$E_{\text{CPD}} = \max_n \frac{\left|\left|\mathbf{A}^{(n)} - \hat{\mathbf{A}}^{(n)}\right|\right|}{\left|\left|\mathbf{A}^{(n)}\right|\right|},$$

in which $\mathbf{A}^{(n)}$ ($\hat{\mathbf{A}}^{(n)}$) are the exact (estimated) factor matrices, $n = 1, \ldots, N$, and scaling and permutation indeterminacies are assumed to be resolved. All timing results are total computation times for a complete run of the optimization algorithm, in contrast to the derived per-iteration complexities in section 6.3. Tensorlab 3.0 [305] is used for all experiments in combination with Matlab 2016b running on a dual socket 20 core Intel Xeon E5-2660 v2 machine with 128 GiB of RAM running CentOS 7.

6.4.1 Accuracy and conditioning

In this first experiment, we study the accuracy of recovered factor matrices for the full tensor and its efficient representation. More specifically, the CPD of a tensor with an exact rank-R structure is computed while varying the relative condition number κ, which is defined as in [292]. Concretely, a rank-5 tensor of size $25 \times 25 \times 25$ is constructed using random factor matrices $\mathbf{A}^{(n)}$, $n = 1, 2, 3$. Each random factor vector $\mathbf{a}_r^{(n)}$ has norm one and a fixed angle α w.r.t. the other factor vectors in the same factor matrix, i.e., for $n = 1, 2, 3$, $\left|\left|\mathbf{a}_r^{(n)}\right|\right| = 1$ and $\cos\alpha = {\mathbf{a}_r^{(n)}}^{\mathrm{T}} \mathbf{a}_s^{(n)}$, for $r, s = 1, \ldots, R$, $r \neq s$. As α decreases, the rank-1 terms become more collinear and the condition number κ increases. A rank-5 CPD is computed from the full tensor and from the structured tensor in the polyadic format using `cpd_nls`, starting from a perturbed exact solution. Figure 6.1 shows the error E_{CPD} on the recovered factor matrices for $\alpha = \pi/2, \ldots, \pi/180$ (using a logarithmic scale). As expected, the error increases when the condition number κ worsens. For well-conditioned problems, E_{CPD} is in the order of the machine precision $\epsilon \approx 10^{-16}$ if the full tensor is used, while E_{CPD} is higher for the structured tensor. This can be explained by the fact that changes smaller than $\sqrt{\epsilon} \approx 10^{-8}$ cannot be distinguished when using the structured tensor, as the computation of the objective function (6.3) requires the difference of squared, almost equal numbers. For very ill-conditioned problems, E_{CPD} may be undesirably high when using the structured tensor. If the tensor admits an exact decomposition *and* if this exact decomposition is required up to machine precision, using the full tensor may be necessary. However, the structured tensor can still be used as an initialization to reduce the overall computational cost.

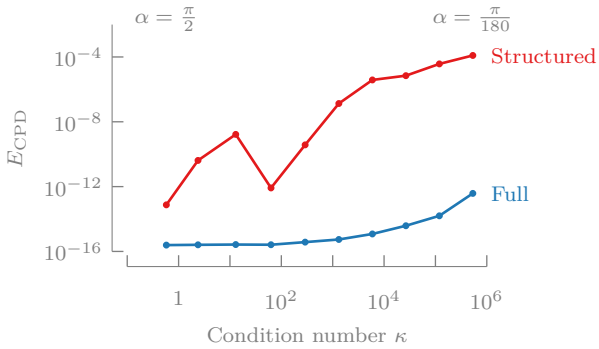


Figure 6.1: By reducing the angle α between the factor vectors, the condition of the CPD worsens, resulting in a higher error E_{CPD}. *For very ill-conditioned problems, using the structured tensor approach may result in an undesirably large error and a few additional iterations using the full tensor may be required to improve the accuracy. Results are medians over 100 experiments.*

6.4.2 Compression for nonnegative CPD

The following experiments illustrate how compression can be used to reduce the complexity of constrained tensor decompositions. Here, we focus on the widely used nonnegative CPD, which is sped up by first computing a truncated MLSVD or a TT approximation. Two approaches to enforce the constraints are used: a projected Gauss–Newton approach generating updates such that the variables are always positive using active sets [160], and using parameter-based constraints, i.e., each entry is the square of an underlying variable [232]. The former approach is implemented using a boundary constrained Gauss–Newton algorithm `nlsb_gndl`; the latter is implemented using the structured data fusion framework [262] (`sdf_nls` and `struct_nonneg`).

For the first experiment, let $\mathcal{T}$ be a rank-10 tensor of size $I \times I \times I$ constructed using random factor matrices $\mathbf{A}^{(n)}$ with entries drawn from a uniform distribution $\mathcal{U}(0, 1)$. Gaussian i.i.d. noise is added such that the signal-to-noise ratio (SNR) is 20 dB. Using a randomized MLSVD (`mlsvd_rsi` [306]) the tensor is compressed such that the core $\mathcal{G}$ has size $10 \times 10 \times 10$, i.e., $\mathcal{T} \approx [\![\mathcal{G}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]\!]$. The core tensor $\mathcal{G}$ and the factors $\mathbf{U}^{(n)}$ are then used as structured format to compute a rank-10 CPD. Random initial factor matrices are drawn from the same distribution as $\mathbf{A}^{(n)}$. Figure 6.2 shows the time required to compute an unconstrained CPD (hence relying on CPD uniqueness), a constrained CPD using projected GN (`cpd_nls` with `nlsb_gndl` solver) and a constrained CPD using parametrization (`sdf_nls`). In the three cases, the computation time for the original tensor $\mathcal{T}$ rises cubically in the dimension I, for $I = 2^6, 2^7, \ldots, 2^{10}$, while the time rises linearly

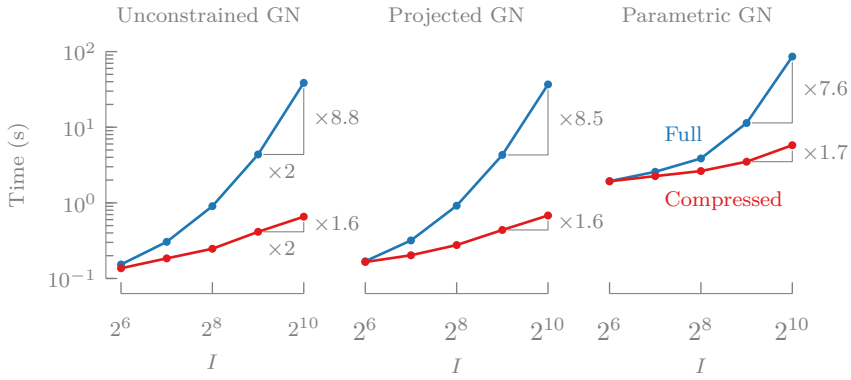


Figure 6.2: When using a compressed tensor instead of the full tensor, the computational cost scales linearly in the tensor dimensions instead of cubically for a rank-10 nonnegative tensor of size $I \times I \times I$. *MLSVD compression with a core size of* $10 \times 10 \times 10$ *is used. The increase for the structured tensor is actually less than linear* ($\times 1.6 - \times 1.7$ *instead of* $\times 2$*), which is caused by an improved multicore usage. The time required to compute the MLSVD increases cubically from* 20 ms *for* $I = 2^6$ *to* 8 s *for* $I = 2^{10}$ *and is not included. Medians over 100 experiments for each parameter are reported.*

using the structured approach. Hence, existing CPD algorithms can be scaled to handle large problems, provided that the MLSVD approximation can be computed and the rank is modest. Note that the compression time, which increases cubically from 20 ms for $I = 2^6$ to 8 s for $I = 2^{10}$, is not included as it can be amortized: in practice, one often uses multiple initializations or experiments with different parameters or constraints.

In a second experiment, the complexity in function of the order N of the tensor is investigated. We compare the required time to compute a nonnegative CPD of an Nth order rank-5 tensor with dimensions $10 \times \cdots \times 10$ using the full tensor and its TT approximation. The data is generated by constructing N random factor matrices with entries drawn from the uniform distribution $\mathcal{U}(0, 1)$. Random Gaussian i.i.d. noise is added to the generated tensor such that the SNR is 20 dB. The core tensors $\mathcal{G}^{(n)}$ corresponding to the TT approximation are computed using `tt_tensor` from the TT Toolbox [213], which we slightly adapted such that all compression ranks $r_n = 5$, $n = 1, \ldots, N-1$[2]. Random factor matrices using the same distribution are used as initialization. Nonnegativity is enforced using the projected GN approach (`cpd_nls` with `nlsb_gndl` solver). The timing results in Figure 6.3 clearly show that using the TT approximation avoids the curse of dimen-

[2]The compression time in not included and increases exponentially from 20 ms for $N = 5$ to 114 s for $N = 9$. An SVD based algorithm is used here; the timings can be improved using, e.g., cross approximation techniques (see Chapter 3).

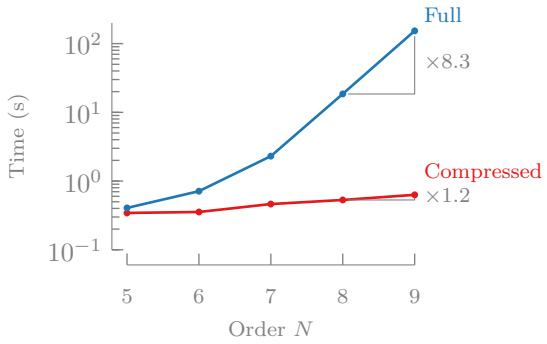


Figure 6.3: Using a TT approximation instead of the full $10\times10\times\cdots\times10$ tensor removes the exponential dependence on the order N when computing a rank-5 nonnegative CPD. *From* $N = 8$ *to* $N = 9$ *the time increases with a factor* $1.2 \approx 9/8$ *using the TT approximation, while the time increase is* $8.3 \approx 10$ *using the full tensor. The median TT compresssion time is not included and increases exponentially from* 20 ms *for* $N = 5$ *to* 114 s *for* $N = 9$. *The results are medians over 100 experiments.*

sionality as the time increases linearly in N in contrast to the time required for the decomposition of the full tensor. Note that we expect the time to increase with a factor 10 if the order increases by one, but the actual time increase is lower as seen in Figure 6.3. This is due to the fact that the decomposition problem becomes easier as relatively more data is available per variable. This is also reflected in the accuracy: for $N = 5$ the median accuracy $E_{\text{CPD}} = 4.7 \cdot 10^{-3}$, while $E_{\text{CPD}} = 3.8 \cdot 10^{-5}$ for $N = 9$.

6.4.3 Signal separation through Hankelization

When source signals are sums of exponential polynomials, the source signals $\mathbf{S} \in \mathbb{R}^{M\times R}$ can be recovered from a noisy mixture $\mathbf{X} \in \mathbb{R}^{M\times K}$ using Hankelization [80]. We use the setup

$$\mathbf{X} = \mathbf{S}\mathbf{M}^{\mathrm{T}} + \mathbf{N}$$

in which $\mathbf{M} \in \mathbb{R}^{K\times R}$ is the mixing matrix and $\mathbf{N} \in \mathbb{R}^{M\times K}$ is additive Gaussian i.i.d. noise scaled such that a given SNR is attained. Hankelization along the columns of $\mathbf{X}$ leads to a third-order tensor $\mathcal{T}$ of size $\lfloor \frac{M+1}{2} \rfloor \times \lceil \frac{M+1}{2} \rceil \times K$ in which $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ are the floor and ceil operators, respectively. Hence, if the number of samples M doubles, the number of entries in $\mathcal{T}$ quadruples. If each of the R source signals is a sum of Q_r exponential polynomials of degree d_{qr}, $q = 1, \ldots, Q_r$, $\mathcal{T}$ can be decomposed into a sum of R multilinear

rank-$(L_r, L_r, 1)$ terms with $L_r = \sum_{q=1}^{Q_r}(d_{qr}+1)$ [80], i.e.,

$$\mathcal{T} = \sum_{r=1}^{R} (\mathbf{A}_r \mathbf{B}_r^{\mathrm{T}}) \otimes \mathbf{c}_r$$

in which $\mathbf{c}_r$ estimates the rth mixing vector $\mathbf{M}(:, r)$. The source signals can be estimated up to scaling and permutation by dehankelizing $\mathbf{A}_r \mathbf{B}_r^{\mathrm{T}}$, e.g., by averaging over the anti-diagonals. The example from [80] is slightly adapted in this experiment: two sources are mixed using

$$\mathbf{M} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix},$$

i.e., $K = R = 2$. The sources are given by

$$\begin{aligned} s_1(t) &= 0.5 \sin(6\pi t) \\ s_2(t) &= (4t^2 - 2.8t) \exp(-t), \end{aligned}$$

hence $L_1 = 2$ and $L_2 = 3$. Estimating R and L_r is out of the scope of this chapter; see, e.g., [80]. The true R and L_r are therefore used. M equidistant samples are taken between 0 s and 1 s.

In the first experiment, the SNR is varied for a fixed number of $M = 501$ samples and the relative error E on the mixing matrix is compared when using the full Hankelized tensor or the implicitly Hankelized tensor, i.e., in the structured format. Hankelization is performed using `hankelize`, which returns both the explicit and implicit tensorization. The resulting tensors of size $251 \times 251 \times 2$ are decomposed using `ll1_nls` starting from a random initialization. From the factor matrices, the signals are recovered using `dehankelize`, without constructing the full tensor. As shown in Figure 6.4, when increasing the SNR from 0 dB to 300 dB, the error decreases at the same rate for the explicit and the implicit tensorization until the SNR is 180 dB: while the error continues to decrease for the explicit tensorization, the error stagnates at $E \approx 10^{-10}$ for the implicit tensorization. This loss of accuracy is explained in subsection 6.4.1 and only occurs for a very high SNR and in the case the sources are exact sums of exponential polynomials. As illustrated in the following experiments, a trade-off between computational cost and accuracy can be made. In the case of a high SNR, the solution using the implicit tensorization can be used to initialize the algorithm with the full tensor.

In the second experiment, the scaling behavior in function of the number of samples is illustrated for $M = 501, 5\,001, 50\,001$ and $500\,001$ points. The resulting Hankelized tensors require 953 KiB, 95.3 MiB, 9.53 GiB and 953 GiB of memory, respectively, if formed explicitly. The SNR is fixed at 20 dB. We compute an unconstrained LL1 decomposition (`ll1_nls`) and a constrained

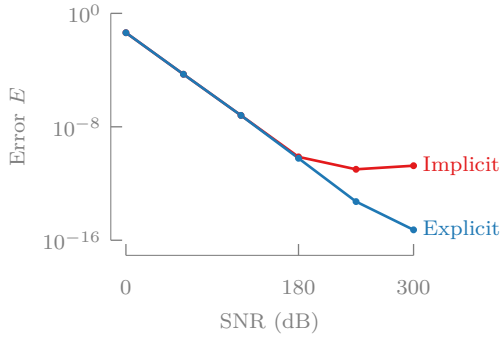


Figure 6.4: *For low and medium SNR, the errors on the mixing matrix for the explicit and implicit Hankelization approaches are equal. While the error for the implicit tensorization stagnates for SNR larger than* 180 dB*, the error continues to decrease when using the explicit tensorization. The errors are medians over 100 experiments, each using a best-out-of-five initializations strategy.*

BTD, which models the tensor as

$$\mathcal{T} = \sum_{r=1}^{R} \left(\mathbf{V}^{(r)} \mathbf{G}^{(r)} \mathbf{V}^{(r)^{\mathrm{T}}} \right) \otimes \mathbf{c}_r,$$

in which $\mathbf{V}^{(r)}$ are confluent Vandermonde matrices and $\mathbf{G}^{(r)}$ are upper anti-triangular matrices; see [80] for details. By imposing the structure in $\mathbf{V}^{(r)}$, the recovered signals are guaranteed to be exponential polynomials, and the underlying poles can be recovered easily. These parametric constraints are modeled in the SDF framework [262] using `struct_confvander` and `sdf_nls`. Figure 6.5 and Figure 6.6 show the median time and error on the mixing matrix over 50 noise realizations. Each algorithm is initialized three and six times using random variables for the LL1 and BTD model, respectively. The time and error for the best initialization, i.e., the one resulting in the lowest error on the mixing matrix, is retained. The timing results in Figure 6.5 clearly show that implicit Hankelization drastically reduces the computation time, allowing longer signals to be analyzed. The computation of the constrained BTD is sensitive to the initialization and is difficult from an optimization point-of-view, as the factors are ill-conditioned due to the generalized Vandermonde structure. This results in higher computation times and a lower accuracy for random initializations. Being able to analyze longer signals allows one to improve the accuracy for a fixed SNR, as is clear from Figure 6.6: by taking 100 times as many samples, E decreases by a factor 10.

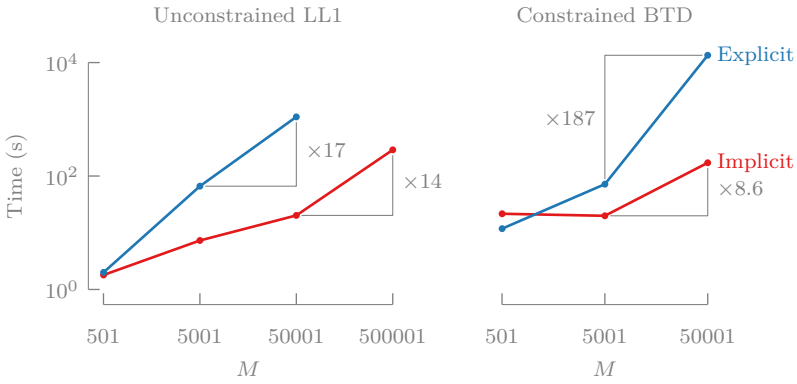


Figure 6.5: For both the unconstrained LL1 decomposition and the constrained BTD, the computation time using implicit tensorization increases more slowly, enabling large-scale applications. *For the LL1 decomposition, the time increases with a factor* $14 \approx 13$ $(\mathcal{O}(M \log_2 M))$ *using the efficient representation and with a factor* 17 *when the full tensor is used. The latter factor is better than the expected factor* 100 *as fewer iterations are required. Computing the constrained BTD is more difficult due to ill-conditioned factors and the variation in time is large causing larger deviatons from the expected scaling factors. The results are medians over 50 experiments with multiple initializations.*

6.5 Conclusion

By rewriting the objective function and gradient expressions commonly used to compute tensor decompositions, four core operations are exposed: squared norm, inner product, matricized tensor times Khatri-Rao product and matricized tensor times Kronecker product. By specializing these operations for efficient representations of tensors, both the computational and memory complexity are reduced from $\mathcal{O}(\text{tensor entries})$ to $\mathcal{O}(\text{parameters})$ as illustrated for the polyadic, Tucker and tensor train format, as well as for Hankelization and Löwnerization. The operations can be used in many decomposition algorithms and frameworks, including ALS, second-order algorithms and algorithms for constrained and/or coupled decompositions. The numerical consequences of exploiting efficient representations are studied in the case a highly accurate solution is required. Finally, important concepts such as Tucker or TT compression for constrained decompositions and implicit tensorization, allow large-scale datasets to be handled easily, as illustrated for nonnegative CPD and the Hankelization of mixtures of exponentials.

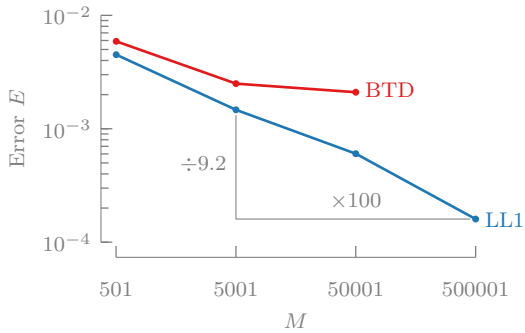


Figure 6.6: The relative error E of the estimated mixing matrix decreases when the number of samples M increases. *In the case of the unconstrained LL1 decomposition, E decreases with a factor $9.2 \approx 10$ when 100 times as many samples are used. The constrained BTD is more difficult from an optimization point-of-view and involves ill-conditioned factor matrices, resulting in a modest improvement in terms of error. The errors are computed using implicit tensorization and are medians over 50 noise realizations with multiple initializations. The results for full tensors are similar.*

Nonlinear least squares updating of the canonical polyadic decomposition

7

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 nonlinear least squares (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 alternating least squares and NLS methods, while maintaining a good accuracy for the decomposition.

This chapter is based on 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. DOI: 10.23919/EUSIPCO.2017.8081290. The figures have been updated for consistency. An additional experiment regarding the influence of updating on accuracy has been added in section 7.A.

7.1 Introduction

Tensor decompositions are powerful tools for various applications in machine learning and signal processing [65], [170], [243]. 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, [99], [223], [260]. Recently, dedicated methods have been designed for large, sparse, or incomplete tensors: see, for example, [219], [300], [304] and references therein. Also, support for structured and coupled tensor decompositions has been added to tensor toolboxes such as Tensorlab [262], [305].

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 [197], [208], [274]. While the decomposition might loose 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 [303], [306][1] 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 7.2. We derive our method in section 7.3, discuss numerical experiments in section 7.4 and conclude in section 7.5.

[1] See also Chapter 6.

7.2 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: $t_{i_1 \ldots i_N} = v_{i_1}^{(1)} v_{i_2}^{(2)} \cdots v_{i_N}^{(N)}$. 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 and inner product are denoted by $||\mathcal{T}||_{\mathrm{F}}$ and $\langle \mathcal{A}, \mathcal{B} \rangle$, respectively, 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 $\mathbf{M}^{\dagger}$. vec($\mathbf{X}$) denotes the vectorization of the matrix $\mathbf{X}$, i.e., putting the columns of $\mathbf{X}$ below each other and diag($\mathbf{x}$) forms a square matrix that has $\mathbf{x}$ as its diagonal. The notation $\begin{bmatrix} \mathbf{a}; \mathbf{b} \end{bmatrix}$ is a shorthand for $\begin{bmatrix} \mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$.

A third-order tensor has rank 1 if it is the outer product of three nonzero 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 $\mathbf{T}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}$ [260].

7.3 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 [260]. 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 7.1, forming a tensor $\mathcal{T}^{(\mathrm{up})}$. 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 $\mathcal{T}^{(\mathrm{up})}$. 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 $\mathbf{c}_{\mathrm{new}}^{\mathrm{T}}$: thus $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = \mathbf{Y}$ and $\mathbf{C} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{c}_{\mathrm{new}}^{\mathrm{T}} \end{bmatrix}$. $\mathbf{c}_{\mathrm{new}}$ can be obtained from the new slice by computing the least squares solution of $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\mathrm{new}} = \mathrm{vec}(\mathbf{M})$:

$$\begin{aligned} \mathbf{c}_{\mathrm{new}} &= (\mathbf{Y} \odot \mathbf{X})^{\dagger} \mathrm{vec}(\mathbf{M}) \\ &= \left[(\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}}(\mathbf{Y} \odot \mathbf{X})\right]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} \mathrm{vec}(\mathbf{M}) \\ &= \left[(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}) * (\mathbf{X}^{\mathrm{T}}\mathbf{X})\right]^{-1} (\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}} \mathrm{vec}(\mathbf{M}). \end{aligned} \tag{7.1}$$

As R is typically small, computing the inverse of the $R \times R$-matrix $[(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}) * (\mathbf{X}^{\mathrm{T}}\mathbf{X})]$ is not expensive, while $(\mathbf{Y} \odot \mathbf{X})^{\mathrm{T}}\mathrm{vec}(\mathbf{M})$ can be obtained without explicitly forming the Khatri–Rao product [293].

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 [303], [306]. 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. The section ends with an analysis of the complexity of the algorithm. The full algorithm is given in Algorithm 7.1.

7.3.1 Objective function

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

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

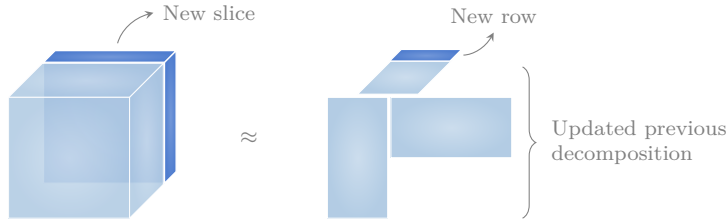


Figure 7.1: In the updating procedure, the decomposition of the old tensor is updated and a new row is added, both based on the new slice.

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

$$f = \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket - \mathcal{T}||_{\mathrm{F}}^2 + \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket - \mathbf{M}||_{\mathrm{F}}^2,$$

which can be expanded to

$$f = \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket ||_{\mathrm{F}}^2 - \left\langle \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket, \mathcal{T} \right\rangle + \frac{1}{2}||\mathcal{T}||_{\mathrm{F}}^2 + \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket ||_{\mathrm{F}}^2 - \langle \llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket, \mathbf{M} \rangle + \frac{1}{2}||\mathbf{M}||_{\mathrm{F}}^2.$$

One can note that $\frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket ||_{\mathrm{F}}^2 + \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket ||_{\mathrm{F}}^2 = \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket ||_{\mathrm{F}}^2$. The full tensor $\mathcal{T}$ is not stored during the updating process for memory efficiency: instead, we work with its CPD approximation $\llbracket \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rrbracket$, which is the best available guess of $\mathcal{T}$. Using this CPD approximation, we obtain the following objective function:

$$f \approx \frac{1}{2}|| \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket ||_{\mathrm{F}}^2 - \left\langle \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket, \llbracket \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rrbracket \right\rangle + \frac{1}{2}|| \llbracket \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rrbracket ||_{\mathrm{F}}^2 - \langle \llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket, \mathbf{M} \rangle + \frac{1}{2}||\mathbf{M}||_{\mathrm{F}}^2. \quad (7.2)$$

Equation (7.2) can be further simplified to avoid the construction of full tensors by exploiting the structure of the CPD. First,

$$\begin{aligned} || \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket ||_{\mathrm{F}}^2 &= \mathrm{vec}(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket)^{\mathrm{T}} \mathrm{vec}(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket) \\ &= \mathbf{1}_R^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A})^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A}) \mathbf{1}_R \\ &= \mathbf{1}_R^{\mathrm{T}} [(\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})] \mathbf{1}_R, \end{aligned} \quad (7.3)$$

where $\mathbf{1}_R$ is a vector of length R consisting of only ones. Similarly, the term $\frac{1}{2}|| \llbracket \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rrbracket ||_{\mathrm{F}}^2$ can be simplified and $\left\langle \llbracket \mathbf{A}, \mathbf{B}, \overline{\mathbf{C}} \rrbracket, \llbracket \mathbf{X}, \mathbf{Y}, \mathbf{Z} \rrbracket \right\rangle$ can be written as $\mathbf{1}_R^{\mathrm{T}} [(\mathbf{A}^{\mathrm{T}}\mathbf{X}) * (\mathbf{B}^{\mathrm{T}}\mathbf{Y}) * (\overline{\mathbf{C}}^{\mathrm{T}}\mathbf{Z})] \mathbf{1}_R$. Because $\llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket = \mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^{\mathrm{T}}$, both $\llbracket \mathbf{A}, \mathbf{B}, \underline{\mathbf{c}} \rrbracket$ and $\mathbf{M}$ are matrices and their inner product is $\mathbf{1}_R^{\mathrm{T}} [(\mathbf{A}\mathrm{diag}(\underline{\mathbf{c}})\mathbf{B}^{\mathrm{T}}) *$

$\mathbf{M}]\mathbf{1}_R$.

7.3.2 Gradient and Gramian

For an NLS algorithm, efficient evaluations of the gradient and Gramian of f are required [303]. Below, we give the derivation of the gradient $\nabla f = \left[\text{vec}\left(\frac{\partial f}{\partial \mathbf{A}}\right); \text{vec}\left(\frac{\partial f}{\partial \mathbf{B}}\right); \text{vec}\left(\frac{\partial f}{\partial \mathbf{C}}\right)\right]$. The expression for the Gramian is identical to the batch algorithm, as shown in [260].

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 (7.2) are not 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

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

leading to

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

Analogously, we have

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

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 (7.2) 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 $\mathbf{J}^{\text{T}}\mathbf{J}\mathbf{p}$, which can be obtained efficiently by exploiting the block-structure of the Gramian. Following Sorber et al. [260], if we write $\mathbf{p} = \left[\text{vec}\left(\mathbf{P}_1\right); \text{vec}\left(\mathbf{P}_2\right); \text{vec}\left(\mathbf{P}_3\right)\right]$, then $(\mathbf{J}^{\text{T}}\mathbf{J}\mathbf{p})_{1,1}$, the contribution of the diagonal $(1,1)$-block of $\mathbf{J}^{\text{T}}\mathbf{J}$ in $\mathbf{J}^{\text{T}}\mathbf{J}\mathbf{p}$, can be computed as $\text{vec}(\mathbf{P}_1[(\mathbf{B}^{\text{T}}\mathbf{B}) * (\mathbf{C}^{\text{T}}\mathbf{C})])$. 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 [25] 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 of the CG algorithm.

7.3.3 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 [201], [208]. In the first strategy $\mathbf{L} = \mathrm{diag}([\lambda^k, \lambda^{k-1}, \ldots, \lambda, 1])$ is used as weighting matrix, so that every old slice is scaled down by a factor λ whenever a new slice arrives. The second strategy 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: $\mathbf{L} = \mathrm{diag}([0, \ldots, 0, 1, \ldots, 1])$, where $\mathbf{L}$ contains M ones. Both strategies can be combined to obtain a truncated exponential window with $\mathbf{L} = \mathrm{diag}(\big[0, 0, \ldots, 0, \lambda^{M-1}, \lambda^{M-2}, \ldots, \lambda, 1\big])$.

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

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

The factor matrices $\mathbf{C}$ and $\mathbf{Z}$ are thus replaced by $\mathbf{L}\mathbf{C}$ 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 $\mathbf{L}\mathbf{C}$ by left multiplication with $\mathbf{L}^\dagger$, where $\mathbf{L}^\dagger$ is obtained by inverting the nonzero entries of $\mathbf{L}$. Otherwise, $\lambda\mathbf{L}\mathbf{C}$ 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 (7.2) 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 [314] for an extended discussion.

7.3.4 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 $\mathbf{J}_p^{\mathrm{T}}\mathbf{J}_p\mathbf{p}_p = -\nabla f_p$, 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 ∇f and PQ

Algorithm 7.1: NLS updating for CPD.

1: **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.
2: **Output:** Updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$.
3: Solve $(\mathbf{Y} \odot \mathbf{X})\mathbf{c}_{\text{new}} = \text{vec}(\mathbf{M})$ using (7.1)
4: 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 $R' = R$)
5: Concatenate $\mathbf{c}_{\text{new}}$ to $\mathbf{Z}$ to obtain the initialization $\left[\!\!\left[\mathbf{X}, \mathbf{Y}, \mathbf{L} \begin{bmatrix} \mathbf{Z} \\ \mathbf{c}_{\text{new}}^{\text{T}} \end{bmatrix} \right]\!\!\right]_R$
6: Remove column or add random column to the initialization if $R' \neq R$
7: 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 7.3
8: Recover $\mathbf{C}$ from $\mathbf{C} = \mathbf{L}^{\dagger}(\mathbf{LC})$ or store $\mathbf{LC}$
9: Return the updated factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$

evaluations of $\mathbf{J}_p^{\text{T}}\mathbf{J}_p\mathbf{p}_p$ 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 both $\frac{1}{2}|| [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] ||_{\text{F}}^2$ and $\frac{1}{2}||\mathbf{M}||_{\text{F}}^2$ are constant and only have to be computed once. Assuming $R' = R$, the terms $\frac{1}{2}|| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] ||_{\text{F}}^2$ and $\left\langle [\![\mathbf{A}, \mathbf{B}, \overline{\mathbf{C}}]\!], [\![\mathbf{X}, \mathbf{Y}, \mathbf{Z}]\!] \right\rangle$ can be computed in $\mathcal{O}\left(R^2 \max(I, J, M)\right)$ flop, with M the length of the window, using the simplification of (7.3). The last term $\langle [\![\mathbf{A}, \mathbf{B}, \underline{\mathbf{c}}]\!], \mathbf{M} \rangle$ can be computed in $\mathcal{O}(IJ)$ flop, totaling at a complexity of $\mathcal{O}\left(2R^2 \max(I, J, M) + IJ\right)$ flop.

The gradient ∇f 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 $\mathcal{O}\left(2R^2 \max(I, J, M) + IJR\right)$ flop, hence the total cost is $\mathcal{O}\left(6R^2 \max(I, J, M) + 3IJR\right)$ flop.

The Gramian-vector product $\mathbf{J}_p^{\text{T}}\mathbf{J}_p\mathbf{p}_p$ requires $\mathcal{O}\left(3R^2 \max(I, J, M)\right)$ flop per CG iteration [260] and thus $\mathcal{O}\left(3QR^2 \max(I, J, M)\right)$ flop per GN iteration. Preconditioning adds three $R \times R$ matrix inversions and Q matrix-vector products per GN iteration, totaling $\mathcal{O}\left(3R^3 + QR(I + J + M)\right)$ flop.

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 $\mathcal{O}((8P+3QP+2P')R^2 \max(I, J, M)+(3RP+P+P')IJ+3PR^3+PQR(I+J+M))$, 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 $\mathcal{O}(R(I + J + M))$ and the new tensor slice, which is $\mathcal{O}(IJ)$. The gradients and Gramian-vector products that are used during the execution of the method both require $\mathcal{O}(R(I + J + M))$ memory as well. In contrast, storing the full (windowed) tensor would require $\mathcal{O}(IJM)$ memory.

7.4 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. [208]. All computations are done in Tensorlab [305]. 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 standard normal distribution. The third factor matrix is sampled along a polynomial with degree three and coefficients drawn 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 uniformly distributed noise over the interval $[-0.5, 0.5]$ with a signal-to-noise ratio (SNR) of 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 7.1, 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. For smaller tensors, the updating method is slower than PARAFAC-SDT.

In Table 7.2, 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 close to that of the batch methods, while PARAFAC-SDT and PARAFAC-RLST perform a lot worse, especially for larger values of R. As only one iteration is performed by the updating

Table 7.1: Median of the CPU time (in ms) for a single update using the new updating method. *The results are compared to NLS and ALS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods.*

R	2	3	4	5	6
NLS (batch)	2375	4464	2557	3563	5522
ALS (batch)	910	1222	1400	1401	2352
SDT	48	71	98	136	172
RLST	570	607	623	775	822
Update 1 P=1,Q=5	60	81	104	140	169
Update 2 P=5,Q=25	166	217	296	398	495

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. Results for $P = 5$ and $Q = 25$ are also included in Tables 7.1 and 7.2. It can be seen that increasing the number of iterations does improve the results slightly. However, the execution time rises linearly with the number of performed iterations. In Figure 7.2, the errors of the different methods are plotted for the case $R = 6$ and SNR $= 50$ dB. The updating method achieves an accuracy that is close to that of the batch methods. Increasing the number of iterations improves the results marginally. 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 slightly larger than the error of the batch methods, but this is compensated by its superior speed. 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 yields a lower accuracy.

7.5 Conclusion

An NLS updating method is 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. 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, gra-

Table 7.2: Weighted mean errors for the new updating method. *The results are medians over 50 updates and are compared to batch NLS, batch ALS, PARAFAC-SDT and PARAFAC-RLST.*

R	2	4	6
NLS (batch)	$1.11 \cdot 10^{-2}$	$8.84 \cdot 10^{-3}$	$9.98 \cdot 10^{-3}$
ALS (batch)	$1.11 \cdot 10^{-2}$	$8.84 \cdot 10^{-3}$	$9.98 \cdot 10^{-3}$
SDT	$2.47 \cdot 10^{-2}$	$4.38 \cdot 10^{-2}$	$6.16 \cdot 10^{-2}$
RLST	$2.68 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$	$8.07 \cdot 10^{-1}$
Update 1 P=1,Q=5	$1.22 \cdot 10^{-2}$	$1.20 \cdot 10^{-2}$	$1.09 \cdot 10^{-2}$
Update 2 P=5,Q=25	$1.16 \cdot 10^{-2}$	$1.18 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$

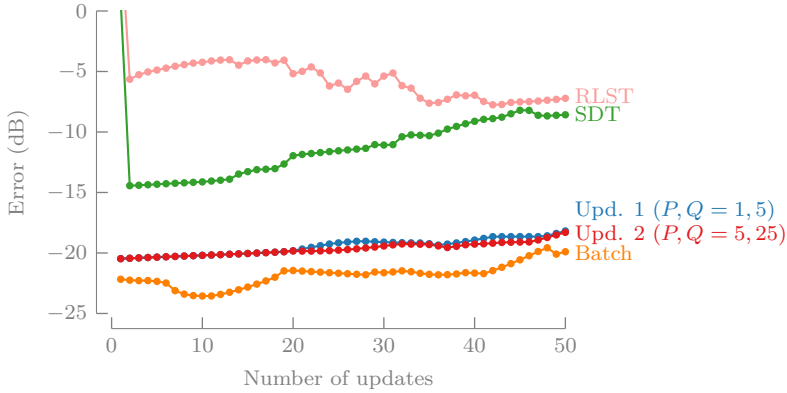


Figure 7.2: The proposed updating methods perform almost as well as the batch methods. *The weighted mean error is shown when using the new updating method, ALS and NLS batch methods and PARAFAC-SDT and PARAFAC-RLST updating methods for* $R = 6$ *and* $SNR = 50\,\text{dB}$.

dient 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.

7.A Updating and accuracy

In this experiment, the accuracy of the estimates of the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ is investigated when updating along the third mode, i.e., every update k a new mode-3 slice added. A $20 \times 20 \times 1020$ tensor of rank $R = 5$ is generated using factor matrices with random entries drawn from a uniform distribution $\mathcal{U}(0, 1)$. In the first experiment no noise is added, while Gaussian i.i.d. noise is added in the second experiment such that the SNR is 20 dB. As starting point, the rank-5 CPD $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is computed from the first 20 mode-3 slices, i.e., $\mathcal{T}(:, :, 1:20)$, using `cpd` from Tensorlab [305], after which the decomposition is updated slice by slice using the presented CPD updating algorithm with the algebraic initialization. We compare the relative factor matrix error for $\mathbf{A}$ and $\mathbf{C}$, after resolving scaling and permutation indeterminacies. (The error on $\mathbf{B}$ is similar to the error on $\mathbf{A}$.) The experiments are repeated 25 times with different random tensors. The maximum number of iterations is 5; the maximum number of CG iterations is 15. In the noiseless case, the function tolerance `TolFun` and step tolerance `TolX` are set to 10^{-32} and 10^{-16}, respectively. The other parameters are set to the default values.

The accuracy of the factor matrices is influenced by two error sources as can be seen from Figure 7.3: the numerical error which increases with the number of updates (see noiseless experiment) and the statistical error which decreases for $\mathbf{A}$ and $\mathbf{B}$ and stays the same for $\mathbf{C}$ (see noisy experiment). The numerical error is often dominated by the statistical error due to noise or model errors. However, for very long experiments the numerical error can become larger than the statistical error. One solution is to recompute (part of) the decomposition using all tensor slices, or, alternatively, one can keep track of a separate, higher accuracy approximation. As the number of data points for each variable in $\mathbf{A}$ and $\mathbf{B}$ increases with every update, the error decreases as expected from statistical theory which states that the variance

of an estimator reduces inversely proportional to the number of data points used to estimate the variable. As the number of data points per variable stays identical for factor matrix $\mathbf{C}$, the error remains constant.

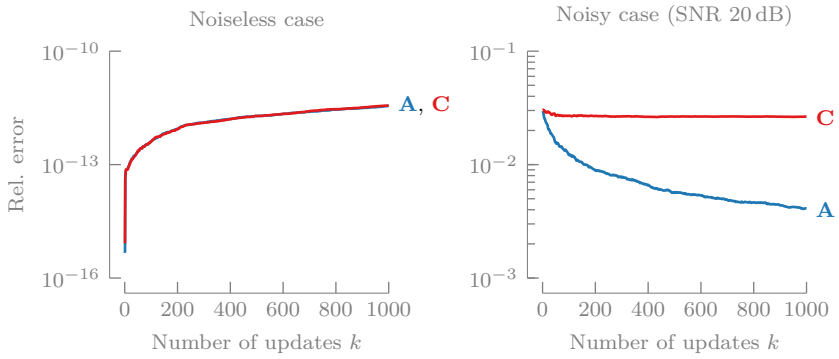


Figure 7.3: In the noiseless case, the error increases due to numerical error accumulation, while the error decreases in the noisy case thanks to statistical averaging. *After the first update, an increase in the error on the factor matrices from* $\mathcal{O}\left(10^{-16}\right)$ *to* $\mathcal{O}\left(10^{-13}\right)$ *can be seen in the noiseless case, which is caused by the use of the structured tensor framework; see Chapter 6. The resulting relative factor matrix errors are medians over 25 experiments with a tensor of dimensions* $20 \times 20 \times (20 + k)$ *of rank* $R = 5$.

Linear systems with a canonical polyadic decomposition constrained solution: algorithms and applications

8

ABSTRACT | Real-life data often exhibit some structure and/or sparsity, allowing one to use parsimonious models for compact representation and approximation. When considering matrix and tensor data, low-rank models such as the (multilinear) singular value decomposition (SVD), canonical polyadic decomposition (CPD), tensor train (TT), and hierachical Tucker (HT) model are very common. The solution of (large-scale) linear systems is often structured in a similar way, allowing one to use compact matrix and tensor models as well. In this chapter we focus on linear systems with a CPD constrained solution (LS-CPD). Our main contribution is the development of optimization-based and algebraic methods to solve LS-CPDs. Furthermore, we propose a condition that guarantees generic uniqueness of the obtained solution. We also show that LS-CPDs provide a broad framework for the analysis of multilinear systems of equations. The latter are a higher-order generalization of linear systems, similar to tensor decompositions being a generalization of matrix decompositions. The wide applicability of LS-CPDs in domains such as classification, multilinear algebra, and signal processing is illustrated.

This chapter is based on M. Boussé, N. Vervliet, I. Domanov, O. Debals, and L. De Lathauwer, "Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications", Technical Report 17-01, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017. The figures have been updated for consistency.

8.1 Introduction

Real-life data can often be modeled using compact representations because of some intrinsic structure and/or sparsity [54]. Well-known representations are low-rank matrix and tensor models such as nonnegative matrix factorization (NMF), the (multilinear) singular value decomposition (SVD), canonical polyadic decomposition (CPD), tensor train (TT), and hierarchical Tucker (HT) models [65], [78], [116], [123], [170], [211], [243]. Examples of data that can be represented or well approximated by such models are exponential polynomials, rational functions (and in a broader sense smooth signals), as well as periodic functions [36], [37], [126], [162]. When dealing with vector/matrix data, one often reshapes the data into higher-order tensors which are then modeled using low-rank approximations, enabling efficient processing in the compressed format. This strategy has been used in tensor-based scientific computing and signal processing to handle various large-scale problems [36], [37], [126], [129].

Similarly, the solution of a (large-scale) linear system can often be expressed by a low-rank tensor. Such problems are well-known in tensor-based scientific computing; see [126]. They arise, e.g., after discretizing high-dimensional partial differential equations. The low-rank model ensures efficient computations and a compact representation of the solution. In such large-scale problems, one often assumes that the coefficient matrix and/or right-hand side have some additional structure or can also be expressed using a tensor model. Several methods have been developed for linear systems with a Kronecker structured coefficient matrix and a CPD structured solution such as the projection method [19], alternating least squares (ALS) [28], and a gradient method [106]. TTs or HT models are also often used because they combine large compression rates and good numerical properties [123], [211].

In this chapter, we present a new framework for linear systems of equations with a CPD constrained solution, abbreviated as LS-CPD. In other words, we want to solve linear systems of the form:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{with} \quad \mathbf{x} = \text{vec}\,(\text{CPD})\,,$$

in which $\text{vec}(\cdot)$ is a vectorization. A simple second-order rank-1 example is $\mathbf{x} = \text{vec}(\mathbf{u} \otimes \mathbf{v})$ with $\otimes$ the outer product. In particular we develop algebraic as well as optimization-based algorithms that properly address the CPD structure. A naive method to solve LS-CPDs could be to first solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ without structure and subsequently decompose a tensorized version $\text{unvec}(\mathbf{x})$ of the obtained solution. This approach works well if the linear system is overdetermined, but, in contrast to our algebraic and optimization-based methods, fails in the underdetermined case. The proposed optimization-based method computes a solution of the LS-CPD problem by minimizing a least-squares

objective function. We have derived expressions for the gradient, Jacobian, and approximation of the Hessian which are the ingredients for standard quasi-Newton (qN) and nonlinear least squares (NLS) techniques. We use the complex optimization framework in Tensorlab, a toolbox for tensor computations in MATLAB, as numerical optimization solver [258]–[260], [305]. The optimization-based methods allow us to work much more efficiently and avoid error accumulation in contrast to the naive or algebraic methods. The latter two methods can be used to obtain a good initialization for the optimization-based methods when considering perturbed LS-CPD problems. Our framework can be extended to other tensor decompositions such as the block term decomposition (BTD), multilinear singular value decomposition (MLSVD), low-multilinear rank approximation (LMLRA), TT or HT models [77], [78], [126], [243].

Furthermore, LS-CPDs can be interpreted as multilinear systems of equations which are a generalization of linear systems of equations. The latter can be expressed by a matrix-vector product between the coefficient matrix and the solution vector, e.g., $\mathbf{A}\mathbf{x} = \mathbf{b}$, or, equivalently, $\mathbf{A} \cdot_2 \mathbf{x}^{\mathrm{T}}$, using the mode-$n$ product [170]. The generalization to a multilinear system is then straightforward because it can be expressed by tensor-vector products between the coefficient tensor and multiple solution vectors: $\mathcal{A} \cdot_2 \mathbf{x}^{\mathrm{T}} \cdot_3 \mathbf{y}^{\mathrm{T}} = \mathbf{b}$. This is very similar to tensor decompositions which are higher-order generalizations of matrix decompositions [65], [170], [243]. However, in contrast to tensor decompositions, the domain of multilinear systems is relatively unexplored. To the best of the authors' knowledge, only a few cases have been studied in a disparate manner such as the fully symmetric rank-1 tensor case with a particular coefficient structure [92], sets of bilinear equations [18], [70], [152], and a particular type of multilinear systems that can be solved via so-called tensor inversion [42]. LS-CPDs provide a general framework to solve multilinear systems; see Figure 8.1.

The CPD structure in LS-CPDs strongly reduces the number of parameters needed to represent the solution. For example, a cubic third-order tensor of size $I \times I \times I$ contains I^3 entries but its CPD needs only $\mathcal{O}(3RI)$ parameters with R the number of terms in the decomposition. The possibly very compact representation of the solution enables one to solve the LS-CPD problem for the underdetermined case in a compressed-sensing (CS) style [54], [101]. A similar idea has been studied for the low-rank matrix case [283]. In contrast to well-known CS reconstruction conditions, we derive a uniqueness condition for LS-CPDs that holds with probability one. In particular, we derive a generic uniqueness condition for the solution $\mathbf{x}$ of the LS-CPD problems given a coefficient matrix $\mathbf{A}$ of which the entries are drawn from absolutely continuous probability density functions.

LS-CPDs appear in a wide range of applications; see, e.g., [208], [277], [296], but the CPD structure is often not recognized or not fully exploited. In this chapter, the applicability of LS-CPDs is illustrated in three different

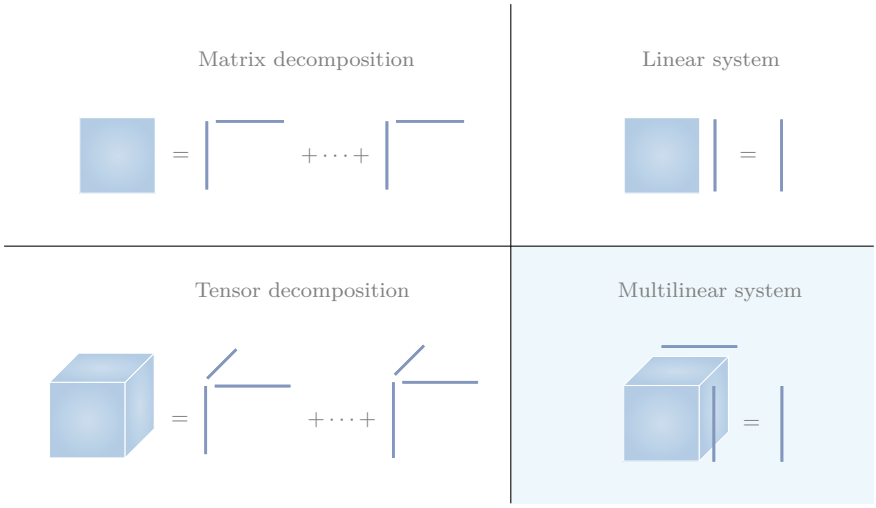


Figure 8.1: Tensor decompositions are a higher-order generalization of matrix decompositions and are well-known tools in many applications within various domains. Although multilinear systems are a generalization of linear systems in a similar way, this domain is relatively unexplored. LS-CPDs can be interpreted as multilinear systems of equations, providing a broad framework for the analysis of these types of problems.

domains: classification, multilinear algebra, and signal processing. In the first case, we show that tensor-based classification can be formulated as the computation of an LS-CPD. Although we illustrate the technique with face recognition [39], one can consider other classification tasks such as irregular heartbeat classification and various computer vision problems [38], [295], [296]. Next, the construction of a real-valued tensor that has particular multilinear singular values is formulated as an LS-CPD. By properly exploiting the symmetry in the resulting problem, our method is faster than literature methods. We conclude with the blind deconvolution of constant modulus such as 4-QAM or BPSK signals using LS-CPDs.

In the remainder of this introduction we give an overview of the notation, basic definitions, and multilinear algebra prerequisites. In section 8.2 we define LS-CPDs and briefly discuss structure and generic uniqueness. Next, we develop an algebraic algorithm and an optimization-based algorithm to compute LS-CPDs in section 8.3. Numerical experiments and applications are presented in sections 8.4 and 8.5, respectively. We conclude the chapter and discuss possible future work in section 8.6.

8.1.1 Notation and definitions

A tensor is a higher-order generalization of a vector (first-order) and a matrix (second-order). We denote tensors by calligraphic letters, e.g., $\mathcal{A}$. Vectors and matrices are denoted by bold lower and bold uppercase letters, respectively, e.g., $\mathbf{a}$ and $\mathbf{A}$. A mode-n vector of a tensor $\mathcal{A} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ (with $\mathbb{K}$ meaning $\mathbb{R}$ or $\mathbb{C}$) is defined by fixing every index except the nth, e.g., $\mathbf{a}_{i_1 \ldots i_{n-1} : i_{n+1} \ldots i_N}$, and is a natural extension of a row or a column of a matrix. The mode-n unfolding of $\mathcal{A}$ is the matrix $\mathbf{A}_{(n)}$ with the mode-n vectors as its columns (following the ordering convention in [170]). An Mth-order slice of $\mathcal{A}$ is obtained by fixing all but M indices. The vectorization of $\mathcal{A}$, denoted as $\text{vec}(\mathcal{A})$, maps each element $a_{i_1 i_2 \ldots i_N}$ onto $\text{vec}(\mathcal{A})_j$ with $j = 1 + \sum_{k=1}^{N} (i_k - 1) J_k$ and $J_k = \prod_{m=1}^{k-1} I_m$ (with $\prod_{m}^{k-1}(\cdot) = 1$ if $m > k-1$). The unvec$(\cdot)$ operation is defined as the inverse of vec$(\cdot)$.

The nth element in a sequence is indicated by a superscript between parentheses, e.g., $\{\mathbf{A}^{(n)}\}_{n=1}^{N}$. The complex conjugate, transpose, conjugated transpose, inverse, and pseudoinverse are denoted as $\bar{\cdot}$, $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$, $\cdot^{-1}$ and $\cdot^{\dagger}$, respectively. A vector of length K with all entries equal to one is denoted as $\mathbf{1}_K$. The identity matrix of size $K \times K$ is denoted as $\mathbf{I}_K$. The binomial coefficient is denoted by $C_n^k = \frac{n!}{(n-k)!k!}$. $\mathbf{A} = \text{diag}(\mathbf{a})$ is a diagonal matrix with the elements of $\mathbf{a}$ on the main diagonal.

The outer and Kronecker product are denoted by $\otimes$ and $\otimes$, respectively, and are related through a vectorization: $\text{vec}(\mathbf{a} \otimes \mathbf{b}) = \mathbf{b} \otimes \mathbf{a}$. The mode-$n$ product of a tensor $\mathcal{A} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{B} \in \mathbb{K}^{J_n \times I_n}$, denoted by $\mathcal{A} \cdot_n \mathbf{B} \in \mathbb{K}^{I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots I_N}$, is defined element-wise as $(\mathcal{A} \cdot_n \mathbf{B})_{i_1 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 i_2 \ldots i_N} b_{j_n i_n}$. Hence, each mode-$n$ vector of the tensor $\mathcal{A}$ is multiplied with the matrix $\mathbf{B}$, i.e., $(\mathcal{A} \cdot_n \mathbf{B})_{(n)} = \mathbf{B}\mathbf{A}_{(n)}$. The inner product of two tensors $\mathcal{A}, \mathcal{B} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\langle \mathcal{A}, \mathcal{B} \rangle$ and defined as $\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1}^{I_1} \sum_{i_2}^{I_2} \cdots \sum_{i_N}^{I_N} a_{i_1 i_2 \ldots i_N} \bar{b}_{i_1 i_2 \ldots i_N}$. The Khatri–Rao and Hadamard product are denoted by $\odot$ and $*$, respectively.

An Nth-order tensor has rank one if it can be written as the outer product of N nonzero vectors. The rank of a tensor is defined as the minimal number of rank-1 terms that generate the tensor as their sum. The mode-n rank of a tensor is defined as the rank of the mode-n unfolding. The multilinear rank of an Nth-order tensor is equal to the tuple of mode-n ranks.

8.1.2 Multilinear algebraic prerequisites

The CPD is a powerful model for various applications within signal processing, biomedical sciences, computer vision, data mining and machine learning [65], [170], [243].

Definition 1. A *polyadic decomposition* (PD) writes an Nth-order tensor $\mathcal{T} \in$

$\mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ as a sum of R rank-1 terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \otimes \mathbf{u}_r^{(2)} \otimes \cdots \otimes \mathbf{u}_r^{(N)} \stackrel{\text{def}}{=} \left[\!\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\!\right], \tag{8.1}$$

in which the columns of the factor matrices $\mathbf{U}^{(n)} \in \mathbb{K}^{I_n \times R}$ are equal to the factor vectors $\mathbf{u}_r^{(n)}$ for $1 \leq r \leq R$. The PD is called *canonical* (CPD) if R is equal to the rank of $\mathcal{T}$, i.e., R is minimal.

The decomposition is *essentially unique* if it is unique up to trivial permutation of the rank-1 terms and scaling and counterscaling of the factors in the same rank-1 term. In the matrix case ($N = 2$) the CPD is not unique without additional assumptions for $R > 1$. Uniqueness is typically expected under rather mild conditions when $N > 2$; see, e.g., [95]–[98], [179] and references therein.

The multilinear singular value decomposition (MLSVD) of a higher-order tensor is a multilinear generalization of the singular value decomposition (SVD) of a matrix [65], [78], [243].

Definition 2. A *multilinear singular value decomposition* (MLSVD) writes a tensor $\mathcal{T} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ as the product

$$\mathcal{T} = \mathcal{S} \bullet_1 \mathbf{U}^{(1)} \bullet_2 \mathbf{U}^{(2)} \cdots \bullet_N \mathbf{U}^{(N)} \stackrel{\text{def}}{=} \left[\!\!\left[\mathcal{S}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\!\right]. \tag{8.2}$$

The factor matrices $\mathbf{U}^{(n)} \in \mathbb{K}^{I_n \times I_n}$, for $1 \leq n \leq N$, are unitary matrices and the core tensor $\mathcal{S} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$ is ordered and all-orthogonal [78].

The (truncated) MLSVD is a powerful tool in various applications such as compression, dimensionality reduction, and face recognition [83], [170], [296]. The decomposition is related to the low-multilinear rank approximation (LMLRA) and the Tucker decomposition (TD); see [78], [304] and references therein. The mode-n unfolding of (8.2) is given by:

$$\mathbf{T}_{(n)} = \mathbf{U}^{(n)} \mathbf{S}_{(n)} \left(\mathbf{U}^{(N)} \otimes \cdots \otimes \mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n-1)} \otimes \cdots \otimes \mathbf{U}^{(1)} \right)^{\mathrm{T}}.$$

8.2 Linear systems with a CPD constrained solution

First, we define linear systems with a CPD constrained solution in subsection 8.2.1. Next, we discuss structure of the coefficient matrix and generic uniqueness in subsections 8.2.2 and 8.2.3, respectively.

8.2.1 Definition

In this chapter, linear systems of equations of which the solution can be represented by a tensor decomposition are considered. We limit ourselves to linear systems with a CPD structured solution, abbreviated as LS-CPD, but one can also use other decompositions such as the MLSVD, TT or HT [78], [126], [211]. Concretely, consider a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with coefficient matrix $\mathbf{A} \in \mathbb{K}^{M \times K}$, solution vector $\mathbf{x} \in \mathbb{K}^{K}$, and right-hand side $\mathbf{b} \in \mathbb{K}^{M}$. As such, we define LS-CPD as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{with} \quad \mathbf{x} = \text{vec}\left(\left[\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right]\!\right]\right) \tag{8.3}$$

with $\mathbf{U}^{(n)} \in \mathbb{K}^{I_n \times R}$, for $1 \leq n \leq N$, and $K = \prod_{n=1}^{N} I_n$. Equation (8.3) can be interpreted as a decomposition of a tensor $\mathcal{X} = \text{unvec}(\mathbf{x})$ that is only *implicitly* known via the solution $\mathbf{x}$ of a linear system. Rather than K variables, the CPD structure allows the vector $\mathbf{x}$ of length K to be represented by only $\mathcal{O}(RI')$ variables with $I' = \sum_{n=1}^{N} I_n$, or, when accommodating for scaling indeterminacies, $R(I' - N + 1)$ free variables. For example, consider the following second-order rank-1 structure $[x; y; z] \otimes [u; v; w]$ which is equivalent with $[1; y/x; z/x] \otimes [ux; vx; wx]$, reducing the number of variables by one. For higher-order structures, this extends to a reduction by $N - 1$, i.e., from $\mathcal{O}\left(I^N\right)$ to $\mathcal{O}\left(NI\right)$ entries. This compact representation allows one to solve the structured linear system in (8.3) in the underdetermined case ($M < K$), enabling a compressed-sensing-style approach [54], [101].

We show that LS-CPDs are multilinear systems of equations. Let $\mathcal{A}$ be a tensor of order $N+1$ with dimensions $M \times I_1 \times I_2 \times \cdots \times I_N$ such that its mode-1 unfolding $\mathbf{A}_{(1)}$ equals the coefficient matrix $\mathbf{A}$, i.e., we have $\mathbf{A}_{(1)} = \mathbf{A}$. We can then rewrite (8.3) as a set of inner products:

$$\left\langle \mathcal{A}_m, \left[\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right]\!\right] \right\rangle = b_m, \text{ for } 1 \leq m \leq M, \tag{8.4}$$

in which $\mathcal{A}_m = \mathcal{A}(m, :, :, \ldots, :)$ is the Nth-order "horizontal slice" of $\mathcal{A}$. If $N = R = 1$, we obtain a linear system of equations and (8.4) reduces to:

$$\langle \mathbf{a}_m^{\mathrm{T}}, \mathbf{x} \rangle = b_m, \quad \text{for} \quad 1 \leq m \leq M,$$

with $\mathbf{a}_m^{\mathrm{T}}$ the mth row of $\mathbf{A}$. Clearly, (8.4) is a set of M multilinear equations. For example, consider the following simple LS-CPD with $N = 2$ and $R = 1$:

$$\mathbf{A}\text{vec}(\mathbf{u} \otimes \mathbf{v}) = \mathbf{b}, \quad \text{or, equivalently,} \quad \mathbf{A}(\mathbf{v} \otimes \mathbf{u}) = \mathbf{b} \tag{8.5}$$

with $\mathbf{A} = \mathbf{A}_{(1)} \in \mathbb{K}^{M \times IJ}$, $\mathbf{u} \in \mathbb{K}^{I}$, and $\mathbf{v} \in \mathbb{K}^{J}$. Equation (8.5) is clearly a compact form of the following set of multilinear equations (with $I = J = 2$

and $M = 4$):

$$\begin{cases} a_{111}v_1u_1 + a_{121}v_1u_2 + a_{112}v_2u_1 + a_{122}v_2u_2 = b_1, \\ a_{211}v_1u_1 + a_{221}v_1u_2 + a_{212}v_2u_1 + a_{222}v_2u_2 = b_2, \\ a_{311}v_1u_1 + a_{321}v_1u_2 + a_{312}v_2u_1 + a_{322}v_2u_2 = b_3, \\ a_{411}v_1u_1 + a_{421}v_1u_2 + a_{412}v_2u_1 + a_{422}v_2u_2 = b_4. \end{cases}$$

or, equivalently, we have $\mathcal{A} \cdot_1 \mathbf{u}^{\mathrm{T}} \cdot_2 \mathbf{v}^{\mathrm{T}} = \mathbf{b}$.

8.2.2 LS-CPD as CPD by exploiting structure of A

For particular types of structure on the coefficient matrix $\mathbf{A}$ in (8.3), the LS-CPD problem can be reformulated as a (constrained) tensor decomposition. Two examples are investigated here. First, if the coefficient matrix in (8.3) is a diagonal matrix $\mathbf{D} = \operatorname{diag}(\mathbf{d})$, the LS-CPD model reduces to a *weighted* CPD of a tensor $\mathcal{B} = \operatorname{unvec}(\mathbf{b})$ [218], [279], i.e., we have:

$$\mathcal{B} = \mathcal{D} * \left[\!\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\!\right]$$

with $\mathcal{D}$ a tensor defined such that $\mathcal{D} = \operatorname{unvec}(\mathbf{d})$. This model can also be used to handle missing entries by setting the corresponding weights to zero [2], [304]. It is clear that an LS-CPD reduces to a CPD if $\mathbf{D}$ is the identity matrix.

Next, we consider a coefficient matrix $\mathbf{A} \in \mathbb{K}^{M \times K}$ that has a Kronecker product structure: $\mathbf{A} = \mathbf{A}^{(N)} \otimes \mathbf{A}^{(N-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}$ with $\mathbf{A}^{(n)} \in \mathbb{K}^{J_n \times I_n}$ such that $M = \prod_{n=1}^{N} J_n$ and $K = \prod_{n=1}^{N} I_n$. Note that

$$\operatorname{vec}\left(\left[\!\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right]\!\!\right] \right) = \left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(1)} \right) \mathbf{1}_R.$$

One can then show that (8.3) can be written as [189]:

$$\begin{aligned} \left(\mathbf{A}^{(N)} \otimes \mathbf{A}^{(N-1)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right) \left(\mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(1)} \right) \mathbf{1}_R &= \mathbf{b}, \\ \left(\mathbf{A}^{(N)}\mathbf{U}^{(N)} \odot \mathbf{A}^{(N-1)}\mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{A}^{(1)}\mathbf{U}^{(1)} \right) \mathbf{1}_R &= \mathbf{b}, \\ \operatorname{vec}\left(\left[\!\!\left[\mathbf{A}^{(1)}\mathbf{U}^{(1)}, \mathbf{A}^{(2)}\mathbf{U}^{(2)}, \ldots, \mathbf{A}^{(N)}\mathbf{U}^{(N)} \right]\!\!\right] \right) &= \mathbf{b}, \end{aligned}$$

which is equivalent to:

$$\left[\!\!\left[\mathbf{A}^{(1)}\mathbf{U}^{(1)}, \mathbf{A}^{(2)}\mathbf{U}^{(2)}, \ldots, \mathbf{A}^{(N)}\mathbf{U}^{(N)} \right]\!\!\right] = \mathcal{B}. \tag{8.6}$$

Expression (8.6) is a CPD with linear constraints on the factor matrices and is also known as the CANDELINC model [57], [170]; note that compatibility

of the dimensions of $\mathbf{U}^{(n)}$ and $\mathbf{A}^{(n)}$ is essential to reformulate the LS-CPD as (8.6). Expression (8.6) can be computed using projection or by using a specific algorithm if the tensor $\mathcal{B}$ has missing entries [302].

8.2.3 Generic uniqueness

We show that generic uniqueness is possible when the number of equations is larger than the number of free variables plus one. More specifically, we present a bound on M guaranteeing uniqueness of $\mathbf{x}$ in (8.3) for a generic $M \times K$ coefficient matrix $\mathbf{A}$. Generic uniqueness means that we have uniqueness with probability one when the entries of $\mathbf{A}$ are drawn from absolutely continuous probability density functions. We refer the reader to [95]–[98], [179] and references therein regarding (generic) uniqueness conditions for the factor matrices in the CPD of $\mathcal{X}$. Our main result states that in order to have a generically unique solution, we need at least as many equations as free variables (i.e., after removing scaling indeterminacies) plus one.

Lemma 1. Let $\mathbf{A}$ be a generic $M \times K$ matrix with $K = I_1 \cdots I_N$. Define $\mathbf{b} = \mathbf{A}\text{vec}(\mathcal{X}_0)$ with $\mathcal{X}_0$ a $I_1 \times \cdots \times I_N$ tensor with rank less than or equal to R. In that case, the solution vector $\mathbf{x}$ in (8.3) is unique if $M \geq (I_1 + \cdots + I_N - N + 1)R + 1$.

Proof. Consider an irreducible algebraic variety $V \in \mathbb{K}^K$ of dimension d_V. It is known that a generic plane of dimension less than or equal to $K - d_V - 1$ does not intersect with V [256, Theorem A.8.1, p. 326]. It is clear that a generic plane of dimension $K - M$ can be interpreted as the null space of a generic $M \times K$ matrix. Hence, if $\mathbf{A}$ is a generic $M \times K$ matrix, $\mathbf{v}_0 \in V$ and $\mathbf{b} := \mathbf{A}\mathbf{v}_0$, then the problem

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad \text{with} \quad \mathbf{x} \in V \tag{8.7}$$

has a unique solution whenever $K - M \leq K - d_V - 1$ or $M \geq d_V + 1$. We interpret (8.3) as (8.7) in which V is the Zariskii closure of the set of $I_1 \times \cdots \times I_N$ tensors whose rank does not exceed R. Since a generic tensor in V can be parameterized with at most $(I_1 + \cdots + I_N - N + 1)R$ parameters, it follows that $d_V \geq (I_1 + \cdots + I_N - N + 1)R$. Hence, a solution vector $\mathbf{x}$ in (8.3) is unique if $M \geq d_V + 1 \geq (I_1 + \cdots + I_N - N + 1)R + 1$. □

8.3 Algorithms

First, we derive an algebraic method to solve an LS-CPD with $R = 1$ in subsection 8.3.1. Next, we develop an optimization-based algorithm for general LS-CPDs in subsection 8.3.2.

8.3.1 Algebraic computation

We present an algebraic method to solve (8.9). The derivation is closely related to [76]. Importantly, all steps can be performed by means of conventional linear algebra. The overall algebraic procedure is summarized in Algorithm 8.1. This method finds the exact solution in the case of exact problems, but can also be used to obtain a good initialization for optimization-based methods in the case of perturbed problems.

It is well-known that a tensor $\mathcal{X}$ of order N has rank one if and only if all its matrix unfoldings have rank one, i.e., we have:

$$\operatorname{rank}\left(\mathbf{X}_{(n)}\right) = R = 1, \quad \text{for } 1 \leq n \leq N. \tag{8.8}$$

In this particular case a solution $\mathbf{x}$ to (8.3) is also a solution of

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{with} \quad \mathbf{x} = \operatorname{vec}(\mathcal{X}), \text{ where } \mathcal{X} \text{ satisfies (8.8)} \tag{8.9}$$

and a solution to (8.9) is also a solution to (8.3). The case $R > 1$ relates to linear systems with a MLSVD constrained solution; see [41]. We can compute a solution of (8.3) *algebraically* in two steps as follows. First, we use (8.9) to recover $\mathcal{X}$. Next, we compute the (exact) rank-1 CPD of $\mathcal{X}$.

Trivial case

Assume that the solution of the unstructured linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is unique, i.e., the null space of the extended matrix $\begin{bmatrix}\mathbf{A} & \mathbf{b}\end{bmatrix}$ is one-dimensional. In that case, we can compute the solution to (8.9) by ignoring the multilinear structure (8.8), i.e., we solve for $\mathbf{x}$ and subsequently compute a CPD of $\mathcal{X} = \operatorname{unvec}(\mathbf{x})$. Clearly, the tensor $\mathcal{X}$ is unique if $\mathbf{b} \neq \mathbf{0}$ or is unique up to a scaling factor if $\mathbf{b} = \mathbf{0}$. This approach is the naive method that we have mentioned in section 8.1.

Reduction of the general case to the trivial case

We explain how to find a solution of (8.9) when the dimension of the null space of $\begin{bmatrix}\mathbf{A} & \mathbf{b}\end{bmatrix}$ is larger than one, e.g., when $\mathbf{A}$ is a fat matrix or rank-deficient. We limit ourselves to the case where $\mathbf{b} \neq \mathbf{0}$, which implies that the dimension of the null space of $\mathbf{A}$ is at least one. It can be shown that the case where $\mathbf{b} = \mathbf{0}$ follows in a similar way.

Let $\mathbf{f}^{(0)}$ be a particular solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ and let the vectors $\mathbf{f}^{(l)}$, for $1 \leq l \leq L$, form a basis of the L-dimensional null space of $\mathbf{A}$. Consider the tensorized versions of $\mathbf{f}^{(l)}$ denoted by $\mathcal{F}^{(l)} \in \mathbb{K}^{I_1 \times I_2 \times \cdots \times I_N}$, for $0 \leq l \leq L$. In order to solve (8.9), we have to find values c_l, for $1 \leq l \leq L$, such that

$\mathcal{X} = \mathcal{F}^{(0)} + c_1\mathcal{F}^{(1)} + \cdots + c_L\mathcal{F}^{(L)}$ satisfies (8.8), i.e., we have that:

$$\operatorname{rank}\left(\mathbf{X}_{(n)}\right) = \operatorname{rank}\left(\mathbf{F}^{(0)}_{(n)} + c_1\mathbf{F}^{(1)}_{(n)} + \cdots + c_L\mathbf{F}^{(L)}_{(n)}\right) = 1, \quad 1 \leq n \leq N. \tag{8.10}$$

We can reformulate (8.10) as the following LS-CPD problem:

$$\tilde{\mathbf{A}}(\mathbf{c} \otimes \mathbf{c}) = \mathbf{0} \quad \text{with} \quad \mathbf{c} = [1 \; c_1 \; \cdots \; c_L]^T \tag{8.11}$$

with, as explained below, $\tilde{\mathbf{A}}$ constructed from the tensors $\mathcal{F}^{(l)}$ such that each row of $\tilde{\mathbf{A}}$ is a vectorized $(L+1) \times (L+1)$ symmetric matrix. We make the assumption that the intersection of the null space of $\tilde{\mathbf{A}}$ with the subspace of vectorized symmetric matrices is one-dimensional. In practice this assumption is satisfied when the difference between the number of rows and columns of $\tilde{\mathbf{A}}$ is sufficiently large. In that case, the solution $\mathbf{c} \otimes \mathbf{c}$ is unique and can be computed as explained for the trivial case, from which $\mathbf{c}$ can be easily recovered.

We explain the construction of $\tilde{\mathbf{A}}$ in more detail. First, partition $\tilde{\mathbf{A}}$ as follows:

$$\tilde{\mathbf{A}} = \left[\tilde{\mathbf{A}}^{(1)^{\mathrm{T}}} \quad \tilde{\mathbf{A}}^{(2)^{\mathrm{T}}} \quad \cdots \quad \tilde{\mathbf{A}}^{(N)^{\mathrm{T}}}\right]^{\mathrm{T}}, \tag{8.12}$$

where the matrices $\tilde{\mathbf{A}}^{(n)}$ correspond to the constraints in (8.10). Consider the following definition.

Definition 3. The second compound matrix $\mathcal{C}_2(\mathbf{F})$ of an $I \times J$ matrix $\mathbf{F}$, with $2 \leq \min(I, J)$, is a $C_I^2 \times C_j^2$ matrix containing all 2×2 minors of $\mathbf{F}$ ordered lexicographically [144].

It is well-known that the following algebraic identity holds for any 2×2 matrices $\mathbf{F}^{(0)}, \ldots, \mathbf{F}^{(L)}$ and values $c_0, \ldots, c_L$:

$$\begin{aligned}&\det(c_0\mathbf{F}^{(0)} + c_1\mathbf{F}^{(1)} + \cdots + c_L\mathbf{F}^{(L)}) = \\ &\frac{1}{2}\sum_{j_1,j_2=1}^{L+1} c_{j_1}c_{j_2}\left[\det(\mathbf{F}^{(j_1)} + \mathbf{F}^{(j_2)}) - \det(\mathbf{F}^{(j_1)}) - \det(\mathbf{F}^{(j_2)})\right].\end{aligned} \tag{8.13}$$

By applying (8.13) to each 2×2 submatrix of $c_0\mathbf{F}^{(0)}_{(n)} + c_1\mathbf{F}^{(1)}_{(n)} + \cdots + c_L\mathbf{F}^{(L)}_{(n)}$, we obtain:

$$\begin{aligned}&\mathcal{C}_2\left(c_0\mathbf{F}^{(0)}_{(n)} + c_1\mathbf{F}^{(1)}_{(n)} + \cdots + c_L\mathbf{F}^{(L)}_{(n)}\right) = \\ &\frac{1}{2}\sum_{j_1,j_2=1}^{L+1} c_{j_1}c_{j_2}\left[\mathcal{C}_2\left(\mathbf{F}^{(j_1)}_{(n)} + \mathbf{F}^{(j_2)}_{(n)}\right) - \mathcal{C}_2\left(\mathbf{F}^{(j_1)}_{(n)}\right) - \mathcal{C}_2\left(\mathbf{F}^{(j_2)}_{(n)}\right)\right].\end{aligned} \tag{8.14}$$

Condition (8.10) states that all 2×2 minors of the matrix $\mathbf{X}_{(n)} = \mathbf{F}^{(0)}_{(n)} + c_1\mathbf{F}^{(1)}_{(n)} + \cdots + c_L\mathbf{F}^{(L)}_{(n)}$ are zero, or, in other words, we have that $\mathcal{C}_2\left(\mathbf{X}_{(n)}\right) = \mathbf{0}$.

Hence, according to (8.14), we have:

$$\sum_{j_1,j_2=1}^{L+1} c_{j_1} c_{j_2} \left[\mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_1)} + \mathbf{F}_{(n)}^{(j_2)} \right) - \mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_1)} \right) - \mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_2)} \right) \right] = 0,$$

with $c_0 = 1$, which is equivalent to

$$\tilde{\mathbf{A}}^{(n)} (\mathbf{c} \otimes \mathbf{c}) = \mathbf{0}, \quad \text{with} \quad \mathbf{c} = [1 \; c_1 \; \ldots \; c_L]^T, \; 1 \leq n \leq N,$$

in which $\tilde{\mathbf{A}}^{(n)}$ has size $C_{I_n}^2 C_{KI_n^{-1}}^2 \times (L+1)^2$ and is defined column-wise as follows:

$$\tilde{\mathbf{a}}_{j_2+(L+1)(j_1-1)}^{(n)} = \text{vec} \left(\mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_1)} + \mathbf{F}_{(n)}^{(j_2)} \right) - \mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_1)} \right) - \mathcal{C}_2 \left(\mathbf{F}_{(n)}^{(j_2)} \right) \right). \tag{8.15}$$

In Algorithm 8.1, the number of rows of $\tilde{\mathbf{A}}$ should be at least the dimension of the subspace of the symmetric $L+1 \times L+1$ matrices minus one. Hence, a *necessary* condition for the algebraic computation is that $\sum_{n=1}^{N} C_{I_n}^2 C_{KI_n^{-1}}^2 \geq (L+1)(L+2)/2 - 1 \geq (K-M+1)(K-M+2) - 1$. Note that L satisfies $L = K - \dim(\text{range}(\mathbf{A})) \geq K - M$ by the rank nullity theorem. The computational complexity of Algorithm 8.1 is dominated by the construction of $\tilde{\mathbf{A}}$.

Algorithm 8.1: Algebraic algorithm to solve $\mathbf{Ax} = \mathbf{b}$ in which $\mathbf{x}$ has a rank-1 CPD structure.

1: **Input:** $\mathbf{A}$ and nonzero $\mathbf{b}$
2: **Output:** $\{\mathbf{u}^{(n)}\}_{n=1}^{N}$
3: Find $\mathbf{f}^{(0)} \in \mathbb{K}^K$ such that $\mathbf{A}\mathbf{f}^{(0)} = \mathbf{b}$
4: Find $\mathbf{f}^{(l)} \in \mathbb{K}^K$, for $1 \leq l \leq L$, that form a basis for $\text{null}(\mathbf{A}) \in \mathbb{K}^{K \times L}$
5: Reshape $\mathbf{f}^{(l)}$ into $I_1 \times I_2 \times \cdots \times I_N$ tensors $\mathcal{F}^{(l)}$, for $0 \leq l \leq L$
6: Construct $\tilde{\mathbf{A}}^{(1)}, \ldots, \tilde{\mathbf{A}}^{(N)}$ as in (8.15) and construct $\tilde{\mathbf{A}}$ as in (8.12)
7: Find a nonzero solution of $\tilde{\mathbf{A}}\tilde{\mathbf{c}} = 0$ (if $\text{null}(\tilde{\mathbf{A}})$ is one-dimensional)
8: Find the vector $\mathbf{c} = [1 \; c_1 \; \ldots \; c_L]^T$ such that $\mathbf{c} \otimes \mathbf{c}$ is proportional to $\tilde{\mathbf{c}}$
9: Construct $\mathcal{X} = \mathcal{F}^{(0)} + c_1 \mathcal{F}^{(1)} + \cdots + c_L \mathcal{F}^{(L)}$
10: Compute the rank-1 CPD of $\mathcal{X} = [\![\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(N)}]\!]$

8.3.2 Optimization-based methods

In this subsection, we solve the LS-CPD problem in (8.3) via a least squares approach, leading to the following optimization problem:

$$\min_{\mathbf{z}} f = \frac{1}{2} ||\mathbf{r}(\mathbf{z})||_{\text{F}}^2 \tag{8.16}$$

in which the residual $\mathbf{r}(\mathbf{z}) \in \mathbb{K}^M$ is defined as

$$\mathbf{r}(\mathbf{z}) = \mathbf{A}\text{vec}\left(\left[\!\left[\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right]\!\right]\right) - \mathbf{b}$$

where we have concatenated the optimization variables in a vector $\mathbf{z} \in \mathbb{K}^{RI'}$ with $I' = \sum_{n=1}^{N} I_n$ as $\mathbf{z} = \left[\text{vec}\left(\mathbf{U}^{(1)}\right); \text{vec}\left(\mathbf{U}^{(2)}\right); \cdots; \text{vec}\left(\mathbf{U}^{(N)}\right)\right]$. To solve the NLS problem (8.16), we use the Gauss–Newton (GN) method which is a particular nonlinear least squares (NLS) algorithm [209]. The latter requires expressions for the objective function, gradient, Gramian, and Gramian-vector product. Although we focus on the GN method, the expressions can be used to implement other NLS algorithms as well as quasi-Newton (qN) algorithms. In order to implement the methods we use the complex optimization framework from [258], [259] which provides implementations for qN and NLS algorithms as well as line search, plane search, and trust region methods.

The GN method solves (8.16) by linearizing the residual vector $\mathbf{r}(\mathbf{z})$ and solving a least squares problem in each iteration k:

$$\min_{\mathbf{p}_k} \frac{1}{2} ||\mathbf{r}(\mathbf{z}_k) + \mathbf{J}_k \mathbf{p}_k||_{\text{F}}^2 \quad \text{s.t.} \quad ||\mathbf{p}_k|| \leq \Delta_k$$

with step $\mathbf{p}_k = \mathbf{z}_{k+1} - \mathbf{z}_k$ and trust-region radius Δ_k [209]. The Jacobian $\mathbf{J} = \partial \mathbf{r}(\mathbf{z}) / \partial \mathbf{z} \in \mathbb{K}^{M \times RI'}$ is evaluated at $\mathbf{z}_k$. The exact solution to the linearized problem is given by the normal equations:

$$\mathbf{J}_k^{\text{H}} \mathbf{J}_k \mathbf{p}_k = -\mathbf{J}_k^{\text{H}} \mathbf{r}(\mathbf{z}_k), \quad \text{or, equivalently,} \quad \mathbf{H}_k \mathbf{p}_k = -\overline{\mathbf{g}}_k. \tag{8.17}$$

In the NLS formulation $\mathbf{H} \in \mathbb{K}^{RI' \times RI'}$ is the Gramian of the Jacobian which is an approximation to the Hessian of f [209]. The conjugated gradient $\overline{\mathbf{g}} \in \mathbb{K}^{RI'}$ is defined as $\overline{\mathbf{g}} = (\partial f / \partial \mathbf{z})^{\text{H}}$. The normal equations are solved inexactly using several preconditioned conjugated gradient (CG) iterations to reduce the computational complexity. After solving (8.17), the variables can be updated as $\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{p}_k$. While a dogleg trust-region method is used here, other updating methods such as line and plane search can be used as well, see [209] for details. In the remainder of this subsection we derive the required expressions for the GN method summarized in Algorithm 8.2.

Objective function

We evaluate the objective function f by taking the sum of squared entries of the residual $\mathbf{r}(\mathbf{z})$. The latter can be computed by using contractions as follows:

$$\mathbf{r}(\mathbf{z}) = \sum_{r=1}^{R} \mathcal{A} \bullet_2 \mathbf{u}_r^{(1)^{\text{T}}} \bullet_3 \mathbf{u}_r^{(2)^{\text{T}}} \cdots \bullet_{N+1} \mathbf{u}_r^{(N)^{\text{T}}} - \mathbf{b}.$$

Algorithm 8.2: LS-CPD using Gauss–Newton with dogleg trust region.

1: **Input:** $\mathbf{A}$, $\mathbf{b}$, and $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$
2: **Output:** $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$
3: **while** not converged **do**
4: Compute gradient $\mathbf{g}$ using (8.18).
5: Use PCG to solve $\mathbf{H}\mathbf{p} = -\overline{\mathbf{g}}$ for $\mathbf{p}$ using Gramian-vector products as in (8.21) using a (block)-Jacobi preconditioner, see subsection 8.3.2.
6: Update $\mathbf{U}^{(n)}$, for $1 \leq n \leq N$, using dogleg trust region from $\mathbf{p}$, $\mathbf{g}$, and function evaluation (8.16).
7: **end while**

Gradient

We partition the gradient as

$$\mathbf{g} = \left[\mathbf{g}^{(1,1)}; \mathbf{g}^{(1,2)}; \ldots; \mathbf{g}^{(1,R)}; \mathbf{g}^{(2,1)}; \ldots; \mathbf{g}^{(N,R)}\right]$$

in which the subgradients $\mathbf{g}^{(n,r)} \in \mathbb{K}^{I_n}$ are defined as

$$\mathbf{g}^{(n,r)} = \left(\mathbf{J}^{(n,r)}\right)^{\mathrm{T}} \overline{\mathbf{r}(\mathbf{z})} \tag{8.18}$$

in which $\mathbf{J}^{(n,r)}$ is defined as

$$\mathbf{J}^{(n,r)} = \frac{\partial \mathbf{r}(\mathbf{z})}{\mathbf{u}_r^{(n)}} = \left(\mathcal{A} \bullet_2 \mathbf{u}_r^{(1)^{\mathrm{T}}} \cdots \bullet_n \mathbf{u}_r^{(n-1)^{\mathrm{T}}} \bullet_{n+2} \mathbf{u}_r^{(n+1)^{\mathrm{T}}} \cdots \bullet_{N+1} \mathbf{u}_r^{(N)^{\mathrm{T}}}\right)_{(1)}. \tag{8.19}$$

Equation (8.19) equals the (n,r)th sub-Jacobian, using a similar partitioning for $\mathbf{J}$. The sub-Jacobians require a contraction in all but the first and nth mode and are precomputed.

Gramian of the Jacobian

We partition the Gramian $\mathbf{H}$ into a grid of $NR \times NR$ blocks $\mathbf{H}^{(n,r,m,l)}$ with $1 \leq n, m \leq N$ and $1 \leq r, l \leq R$. Each block $\mathbf{H}^{(n,r,m,l)}$ is defined by:

$$\mathbf{H}^{(n,r,m,l)} = \left(\mathbf{J}^{(n,r)}\right)^{\mathrm{H}} \mathbf{J}^{(m,l)}, \tag{8.20}$$

using the sub-Jacobians in (8.19). Equation (8.20) approximates the second-order derivative of f with respect to the variables $\mathbf{u}_r^{(n)}$ and $\mathbf{u}_l^{(m)}$.

As preconditioned CG (PCG) is used, only matrix vector-products are needed. The full Gramian is never constructed because one can exploit the block structure to compute fast matrix-vector products. Hence, in each iteration we compute Gramian-vector products of the form $\mathbf{z} = \mathbf{J}^{\mathrm{H}}\mathbf{J}\mathbf{y}$ as follows:

Table 8.1: The per-iteration computational complexity of the NLS algorithm for LS-CPD is dominated by the computation of the Jacobian. *The algorithm uses a trust region approach to determine the update, which requires it_{TR} additional evaluations of the objective function.*

	Calls per iteration	Complexity
Objective function	$1 + \text{it}_{\text{TR}}$	$\mathcal{O}(MRI^N)$
Jacobian	1	$\mathcal{O}(MRNI^N)$
Gradient	1	$\mathcal{O}(MRNI)$
Gramian	1	$\mathcal{O}(MRNI^2)$
Gramian-vector	it_{CG}	$\mathcal{O}(MRNI)$

$$\mathbf{z}^{(n,r)} = \left(\mathbf{J}^{(n,r)}\right)^{\text{H}} \left(\sum_{n=1}^{N}\sum_{r=1}^{R} \mathbf{J}^{(n,r)}\mathbf{y}^{(n,r)}\right), \text{ for } 1 \leq n \leq N, \text{ and } 1 \leq r \leq R, \tag{8.21}$$

in which we partitioned $\mathbf{z}$ and $\mathbf{y}$ in a similar way as before.

In this chapter, we use either a block-Jacobi or Jacobi preconditioner to improve convergence or reduce the number of CG iterations. In the former case, we compute the $(I_n \times I_n)$ Gramians $\mathbf{H}^{(n,n,r,r)}$, for $1 \leq n \leq N$ and $1 \leq r \leq R$, *and* their inverses in each iteration. Combining both operations leads to a per-iteration computational complexity of $\mathcal{O}(MI_n^2 + I_n^3)$ which is relatively expensive, especially for large problems. One can instead use a Jacobi preconditioner which uses a diagonal approximation of the Gramian and, consequently, an inexpensive computation of the inverse. The diagonal elements are computed as the inner product $\mathbf{J}_{i_n}^{(n,r)^{\text{H}}}\mathbf{J}_{i_n}^{(n,r)}$, for $1 \leq i_n \leq I_n$, leading to an overall computational complexity of $\mathcal{O}(MI_n + I_n)$ which is relatively inexpensive. We compare the effectiveness of the Jacobi and block-Jacobi preconditioner in section 8.4.

Complexity

We report the per-iteration complexity of the NLS algorithm for LS-CPD in Table 8.1. For simplicity, we assume that $I_1 = I_2 = \cdots = I_N = I$ in (8.3). Clearly the computational complexity is dominated by the computation of the sub-Jacobians in (8.19). The computational complexity can be reduced by computing the contractions in (8.19) as efficiently as possible. Note that the evaluation of the objective function is a factor N less expensive.

Efficient contractions

The per-iteration complexity of the NLS algorithm is relatively high, however, only a few iterations are often necessary in order to obtain convergence. One

can reduce the overall computation time of the algorithm by reducing the computational cost per iteration or the number of iterations. We have shown that the computation of the Jacobians is relatively expensive; see Table 8.1. Computing the sub-Jacobians requires the sequential computation of $N-1$ contractions of the form $\mathcal{A} \cdot_n \mathbf{x}^{(n)^{\mathrm{T}}} \in \mathbb{K}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$ which are defined as

$$(\mathcal{A} \cdot_n \mathbf{x}^{(n)^{\mathrm{T}}})_{i_1 \ldots i_{n-1} i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 \ldots i_N} x_{i_n}^{(n)} \tag{8.22}$$

with $\mathcal{A} \in \mathbb{K}^{I_1 \times \cdots \times I_N}$ and a vector $\mathbf{x}^{(n)} \in \mathbb{K}^{I_n}$. Clearly, it is important to perform the contractions as efficiently as possible to reduce the per-iteration complexity of the algorithm. Note that the computation of the contractions can be done in a memory-efficient way by computing the contractions sequentially via the matrix unfoldings and permuting the first mode of $\mathcal{A}$ to the middle. This approach guarantees that $\mathcal{A}$ is permuted in memory at most once.

One way to compute contractions efficiently is by exploiting all possible structure of the coefficient tensor $\mathcal{A}$ in (8.22). For example, if $\mathbf{A}$ is the identity matrix, the LS-CPD problem reduces to a CPD. In that case, the Gramians and their inverses can be computed efficiently by storing the Gramians of the factor matrices; see [260]. If $\mathbf{A}$ has a Kronecker product structure, the LS-CPD problem reduces to a CANDELINC model, as shown in subsection 8.2.2, which can be computed efficiently in both the dense and sparse case; see [170], [302]. For specific types of structure in $\mathbf{A}$, forward-adjoint oracles [91] can be generalized to the multilinear case.

Let us illustrate how we can compute efficient contractions in the case of a sparse $\mathcal{A}$. Assume we have a vector $\mathbf{a} \in \mathbb{K}^M$ containing the M nonzero values of $\mathcal{A}$ and corresponding index sets $S_m = \{i_1^{(m)}, i_2^{(m)}, \ldots, i_N^{(m)}\}$, for $1 \leq m \leq M$. We can then compute (8.22) efficiently as $\mathbf{w} = \mathbf{v} * \mathbf{u}_{S'_n}^{(n)}$ with $S'_n = \{i_n^{(1)}, i_n^{(2)}, \ldots, i_n^{(M)}\}$, for $1 \leq n \leq N$. As such, we obtain a new index-value pair with $\mathbf{w} \in \mathbb{K}^M$ and $R_m = S_m \backslash i_n^{(m)}$ for $1 \leq m \leq M$.

8.4 Numerical experiments

First, two proof-of-concept experiments are conducted to illustrate the algebraic and optimization-based methods in subsection 8.4.1. Next, we compare accuracy and time complexity of the naive, algebraic, and NLS method in subsection 8.4.2. We also compare algebraic and random initialization methods for the NLS algorithm in subsection 8.4.3. In subsection 8.4.4, we compare the Jacobi and block-Jacobi preconditioner for the NLS algorithm. All computations are done with Tensorlab [305]. We define the relative error $\epsilon_{\mathbf{x}}$ as the relative difference in Frobenius norm $\|\mathbf{x} - \hat{\mathbf{x}}\|_{\mathrm{F}} / \|\mathbf{x}\|_{\mathrm{F}}$ with $\hat{\mathbf{x}}$ an esti-

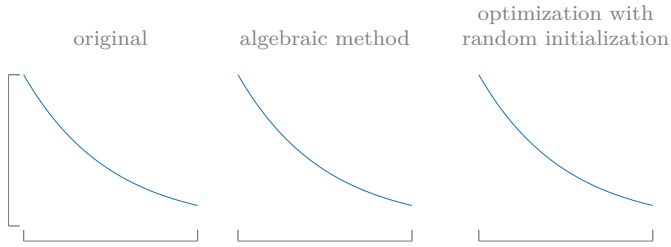


Figure 8.2: Our algebraic method and optimization-based method (with random initialization) can perfectly reconstruct an exponential solution vector in the noiseless case.

mate for $\mathbf{x}$. We use factor matrices in which the elements are drawn from the standard normal distribution, unless stated otherwise, to generate tensors. In that case the factor matrices are well-conditioned because the expected angle between the factor vectors is 90° for large matrices. The coefficient matrices $\mathbf{A}$ are constructed in a similar way, unless stated otherwise. We use i.i.d. Gaussian noise to perturb the entries of a tensor. The noise is scaled to obtain a given signal-to-noise ratio (SNR) (with the signal equal to the noiseless tensor). If we consider a perturbed LS-CPD problem, we perturb the right-hand side in (8.3), unless stated otherwise.

8.4.1 Proof-of-concept

We give two simple proof-of-concept experiments, illustrating our algorithms for linear systems with a solution that can be represented or well approximated by a low-rank model. First, consider an LS-CPD with a solution $\mathbf{x} \in \mathbb{K}^K$ that is constrained to be an exponential, i.e., $x_k = e^{-2k}$ evaluated in K equidistant samples in $[0, 1]$. It is known that sums of exponentials can be exactly represented by low-rank tensors [36], [37], [80]. In this case, the corresponding tensor $\mathcal{X} = \text{unvec}(\mathbf{x})$ has rank one ($R = 1$) [80]. We choose $N = 3$, $I_1 = I_2 = I_3 = I = 4$, $K = I^3 = 64$, and $M = 34$. We compute a solution using the algebraic method and the NLS algorithm with random initialization. Perfect reconstruction of the exponential is obtained with both methods as shown in Figure 8.2.

In the previous experiment the solution vector could be exactly represented by a low-rank tensor. Many signals, such as Gaussians, rational functions, and periodic signals, can also be well approximated by a low-rank model [36], [37]. In this experiment, we consider an LS-CPD of which the solution vector $\mathbf{x} = \text{vec}(\mathbf{X})$ is a rational function:

$$x_k = \frac{1}{(k - 0.3)^2 + 0.04^2} + \frac{1}{(k - 0.8)^2 + 0.06^2},$$

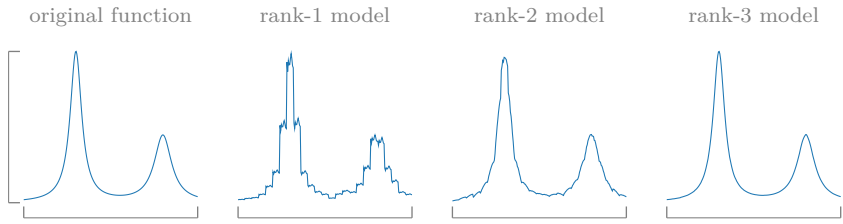


Figure 8.3: Our optimization-based method (with random initialization) can reconstruct the rational solution vector in the noiseless case. *Increasing the rank of the CPD model improves the accuracy of the solution. For example, the rank-3 model is almost indistinguishable from the original function.*

evaluated at K equidistant samples in $[0, 1]$. We take $N = 2$, $I_1 = 10$, $I_2 = 25$, $K = I_1 I_2 = 250$, and $M = 350$. The NLS algorithm with random initialization is used for $R = \{1, 2, 3\}$ to compute a solution. In Figure 8.3, one can see that the accuracy of the approximation increases when using higher rank values.

8.4.2 Comparison of methods

We compare the algebraic method in Algorithm 8.1, the NLS method in Algorithm 8.2, and the naive method, i.e., the trivial case of the algebraic method, see subsection 8.3.1. Remember that the latter can be computed by first solving the unstructured system and afterwards fitting the CPD structure on the obtained solution. Consider an LS-CPD with $N = 3$, $I_1 = I_2 = I_3 = I = 3$, $R = 1$, $K = I^3 = 27$. We choose $M^{(\min)} \leq M \leq K$ with $M^{(\min)} = 8$ which equals the minimal value of M to obtain a (generically) unique solution according to Lemma 1. We report the median relative error on the solution $\epsilon_{\mathbf{x}}$ and the time across 100 experiments in Figure 8.4. The naive method fails when $M < K$ because we solve an underdetermined linear system, resulting in a nonunique solution due to the nonemptiness of the null space of $\mathbf{A}$. The algebraic method works well, but fails if $M \leq 10$ because then the dimension of the null space of $\tilde{\mathbf{A}}$ in (8.11) is larger than one, see subsection 8.3.1. For $M = K$, the algebraic method coincides with the naive method. The NLS method performs well for all M using five random initializations. Note that NLS typically needs many random initializations when M is close to $M^{(\min)}$. The accuracy is slightly higher than the algebraic method. The computational cost of the algebraic method increases when M decreases because $\tilde{\mathbf{A}}$ in (8.11) depends quadratically on L which is the dimension of the null space of $\mathbf{A}$.

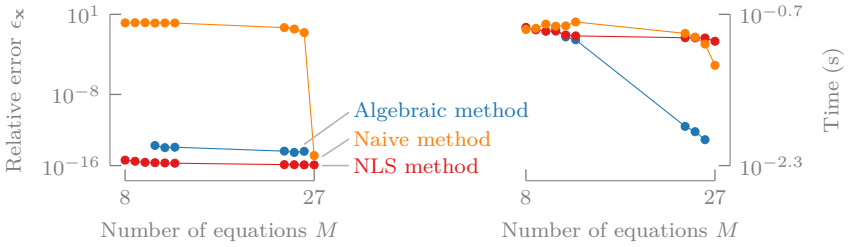


Figure 8.4: The naive method fails for an underdetermined LS-CPD while the NLS and algebraic method both perform well. The computational complexity of the algebraic method is much higher than the other two methods, especially for the highly underdetermined case (i.e., M close to the number of free variables).

8.4.3 Initialization methods

The algebraic method in Algorithm 8.1 finds the exact solution in the case of exact problems. In the case of perturbed problems, however, the solution can be used to obtain a good initialization for optimization-based methods such as the NLS algorithm from subsection 8.3.2. Often the algebraic solution provides a better starting value for optimization-based algorithms than a random initialization. We illustrate this for an underdetermined LS-CPD of the form (8.3) with $N = 3$, $R = 1$, $I_1 = I_2 = I_3 = I = 4$, $K = I^3 = 64$, and $M = 60$. We compute a solution using the NLS algorithm with random and algebraic initialization. In Figure 8.5, we report the median number of iterations across 20 experiments for several values of the SNR; we also show the convergence plot for 20 dB SNR on the left. By starting the NLS algorithm from the algebraic solution instead of using a random initialization, we need fewer iterations to achieve convergence. Importantly, the algebraic method can still find a solution in the noisy case but the accuracy is typically low. Optimization-based methods such as the NLS algorithm can use this solution as an initialization and improve the accuracy.

8.4.4 Preconditioner

The overall computation time of the NLS algorithm can be reduced by reducing the computational cost per iteration or the number of iterations. Remember that we solve the normal equations in the NLS algorithm inexactly via a number of PCG iterations. Good preconditioning is essential to lower the number of CG iterations and, consequently, reduce the per-iteration complexity of the NLS algorithm. Here, we compare the Jacobi and block-Jacobi preconditioner (PC), see section 8.3.

Consider an LS-CPD problem with $N = 3$, $R = 3$, $I_1 = 250$, $I_2 = I_3 = 10$,

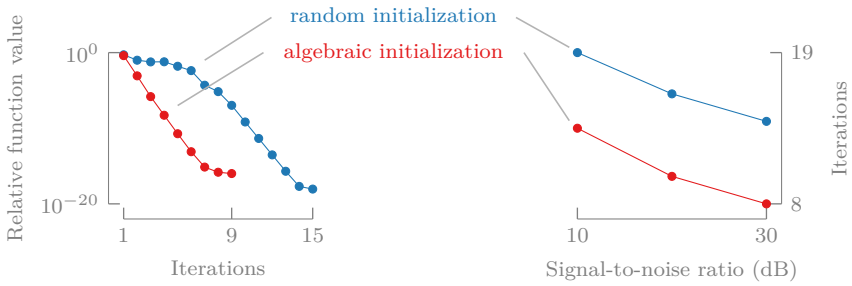


Figure 8.5: By initializing the NLS algorithm with the algebraic solution instead of using a random initialization, fewer iterations are needed to achieve convergence.

Table 8.2: Both PCs reduce the number of CG iterations in the underdetermined and square case. In the highly underdetermined case only the block-Jacobi PC can reduce the number of CG iterations. *We reported the average (and standard deviation of the) number of CG iterations across 50 experiments*

Scenario	No PC	Jacobi PC	block-Jacobi PC
highly underdetermined	810 (0)	790 (30)	644 (55)
underdetermined	181 (38)	56 (2)	46 (2)
square	45 (12)	12 (2)	12 (2)

$K = I_1 I_2 = 25000$. We consider three different scenarios: the highly underdetermined case ($M = R(I_1 + I_2 + I_3) + 5 = 815$), the underdetermined case ($M = 1.5R(I_1 + I_2 + I_3) = 1215$), and the square case $M = K = 25000$. We simulate a typical iteration of the NLS algorithm as follows. We compute the Gramian $\mathbf{H}$ and the gradient $\mathbf{g}$ for random factor matrices $\mathbf{U}^{(n)}$ and solve the normal equations in (8.17) using PCG until convergence (i.e., up to a relative error on the residual of 10^{-6}). In Table 8.2 we report the average number of CG iterations across 50 experiments when using no PC, the Jacobi PC, and the block-Jacobi PC for the three different scenarios. In this experiment, the block-Jacobi preconditioner reduces the number of CG iterations more than the Jacobi preconditioner, especially for the highly underdetermined case. In the square case, both PCs have similar performance, but the Jacobi PC is preferred because of its lower computational complexity.

8.5 Applications

LS-CPDs provide a generic framework that can be used in a wide range of applications. In this chapter, we illustrate with three applications in classification, multilinear algebra and signal processing, respectively. First, LS-

CPDs are used for tensor-based face recognition in subsection 8.5.1. The technique is very generic and can be used for other classification tasks as well. For example, a similar method was used in [38] for irregular heartbeat classification in the analysis of electrocardiogram data. Next, it is shown in subsection 8.5.2 that tensors that have particular multilinear singular values can be constructed using LS-CPDs. Finally, in subsection 8.5.3, LS-CPDs are used for the blind deconvolution of constant modulus signals.

8.5.1 Tensor-based face recognition using LS-CPDs

LS-CPDs can be used for generic classification tasks which is illustrated here using tensor-based face recognition [39], [296]. Consider a set of matrices of size $M_x \times M_y$ representing the facial images of J persons, taken under I different illumination conditions. All *vectorized* images of length $M = M_x M_y$ are stacked in a third-order tensor $\mathcal{D} \in \mathbb{K}^{M \times I \times J}$ with modes pixels (px) $\times$ illumination (i) $\times$ persons (p). Next, we perform a multilinear analysis by computing a (truncated) MLSVD of the tensor $\mathcal{D}$, i.e., we have:

$$\mathcal{D} \approx \mathcal{S} \cdot_1 \mathbf{U}_{\text{px}} \cdot_2 \mathbf{U}_{\text{i}} \cdot_3 \mathbf{U}_{\text{p}}.$$

with $\mathbf{U}_{\text{px}}$, $\mathbf{U}_{\text{i}}$, and $\mathbf{U}_{\text{p}}$ forming an orthonormal basis for the pixel, illumination, and person mode, respectively. The core tensor $\mathcal{S}$ explains the interaction between the different modes. The vectorized image $\mathbf{d} \in \mathbb{K}^M$ for a *particular* illumination i and person p satisfies:

$$\mathbf{d} = (\mathcal{S} \cdot_1 \mathbf{U}_{\text{px}}) \cdot_2 \mathbf{c}_{\text{i}}^{\text{T}} \cdot_3 \mathbf{c}_{\text{p}}^{\text{T}} \tag{8.23}$$

with $\mathbf{c}_{\text{i}}^{\text{T}}$ and $\mathbf{c}_{\text{p}}^{\text{T}}$ rows of $\mathbf{U}_{\text{i}}$ and $\mathbf{U}_{\text{p}}$, with $\mathbf{U}_{\text{p}}$ acting as a database. The mode-1 unfolding of (8.23) is an LS-CPD of the form (8.5):

$$\mathbf{d} = \mathbf{U}_{\text{px}} \mathbf{S}_{(1)} (\mathbf{c}_{\text{p}} \otimes \mathbf{c}_{\text{i}}).$$

Consider a previously unknown image $\mathbf{d}^{(\text{new})}$ of a person that is included in the database. Classification or recognition of this person corresponds to finding the coefficient vector $\mathbf{c}_{\text{p}}$, i.e., we solve an LS-CPD of the form:

$$\mathbf{d}^{(\text{new})} = \mathbf{U}_{\text{px}} \mathbf{S}_{(1)} \left(\mathbf{c}_{\text{p}}^{(\text{new})} \otimes \mathbf{c}_{\text{i}}^{(\text{new})} \right),$$

resulting into estimates $\tilde{\mathbf{c}}_{\text{p}}^{(\text{new})}$ and $\tilde{\mathbf{c}}_{\text{i}}^{(\text{new})}$. The coefficient vector for the person dimension $\tilde{\mathbf{c}}_{\text{p}}^{(\text{new})}$ is compared with the rows of $\mathbf{U}_{\text{p}}$ using the Frobenius norm of the difference (after fixing scaling and sign invariance). We then classify the person in the image according to the label of the closest match.

Let us illustrate the above strategy for the extended YaleB dataset[1]. This

[1] Available from http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html.

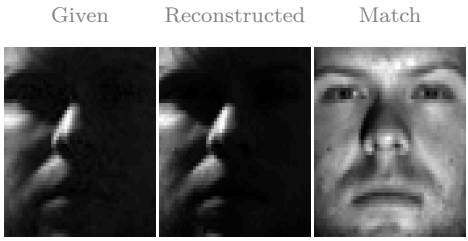


Figure 8.6: Correct classification of an image of a person under a *new* illumination condition. *We can identify the person even though the picture is mostly dark.*

real-life dataset consists of cropped facial images of 39 persons in 64 illumination conditions. We remove illumination conditions for which some of the images are missing and retain one of the conditions as test data, resulting into $I = 56$ conditions. We vectorize each image of 51×58 pixels into a vector of length $M = 2958$ for $J = 37$ persons. The resulting data tensor $\mathcal{D}$ has size $2958 \times 56 \times 37$. We compute the MLSVD of $\mathcal{D}$ using a randomized algorithm called `mlsvd_rsi`, which is faster than non-randomized algorithms but achieves similar accuracy [306]. We compress the pixel mode to reduce noise influences. As such, we obtain a core tensor $\mathcal{S} \in \mathbb{K}^{500\times56\times37}$ and matrices $\mathbf{U}_{\text{px}} \in \mathbb{K}^{2958\times500}$, $\mathbf{U}_{\text{i}} \in \mathbb{K}^{56\times56}$, and $\mathbf{U}_{\text{p}} \in \mathbb{K}^{37\times37}$. We use the NLS algorithm to compute a solution, starting from a random initialization. We project the new image $\mathbf{d}^{(\text{new})}$ onto the column space of the pixel matrix $\mathbf{U}_{\text{px}}$ in order to decrease computation time, i.e., $\mathbf{b} = \mathbf{U}_{\text{px}}{}^{\text{T}}\mathbf{d}^{(\text{new})}$. We compare the estimated coefficient vector with $\mathbf{U} = \mathbf{U}_{\text{p}}$. To accommodate for scaling and sign invariance, we normalize the rows of $\mathbf{U}$ and $\tilde{\mathbf{c}}_{\text{p}}^{(\text{new})}$ as follows: a vector $\mathbf{c}$ is normalized as $\text{sign}(c_1)\frac{\mathbf{c}}{||\mathbf{c}||}$. On the left in Figure 10.1, we see the facial image of a person that is known to our model but for a new illumination condition. In the middle one can see the reconstruction of the image using the estimated coefficient vectors. Moreover, we correctly classified the person as the person on the right in Figure 10.1.

8.5.2 Constructing a tensor that has particular multilinear singular values

Constructing a matrix with particular singular values is trivial. One can simply use the SVD: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\text{T}}$ in which $\mathbf{\Sigma}$ is a diagonal matrix containing the given singular values and $\mathbf{U}$ and $\mathbf{V}$ are random orthogonal matrices. For tensors, this is not straightforward. It is of fundamental importance to understand the behavior of multilinear singular values [100], [131], [133]. In this section, we show how one can construct an all-orthogonal tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ with particular multilinear singular values using an LS-CPD. Consider the

following expressions:

$$\begin{cases} \mathbf{T}_{(1)}\mathbf{T}_{(1)}^{\mathrm{T}} = \boldsymbol{\Sigma}^{(1)}, \\ \mathbf{T}_{(2)}\mathbf{T}_{(2)}^{\mathrm{T}} = \boldsymbol{\Sigma}^{(2)}, \\ \mathbf{T}_{(3)}\mathbf{T}_{(3)}^{\mathrm{T}} = \boldsymbol{\Sigma}^{(3)}, \end{cases} \tag{8.24}$$

in which $\boldsymbol{\Sigma}^{(n)} = \mathrm{diag}(\boldsymbol{\sigma}^{(n)^2})$ is a diagonal matrix containing the squared multilinear singular values $\boldsymbol{\sigma}^{(n)}$, $n = 1, 2, 3$. Expression (8.24) states that $\mathcal{T}$ is all-orthogonal and has multilinear singular values $\boldsymbol{\sigma}^{(n)}$. In order to reformulate (8.24) as an LS-CPD, we only take the upper triangular parts into account because of symmetry in the left- and right-hand side in (8.24), leading to the following equations for the first expression in (8.24):

$$\sum_{j,k} t_{ijk}t_{ijk} = \left(\sigma_i^{(1)}\right)^2, \text{ for } 1 \leq i \leq I,$$

$$\sum_{j,k} t_{i_1jk}t_{i_2jk} = 0, \text{ for } 1 \leq i_1 < i_2 \leq I,$$

and similarly for the second and third expression. We can write this more compactly as an LS-CPD:

$$\mathbf{A}(\mathbf{u} \otimes \mathbf{u}) = \mathbf{b} \quad \text{with} \quad \mathbf{u} = \mathrm{vec}(\mathcal{T}).$$

$\mathbf{A}$ is a binary and sparse matrix of size $I \times J^2$ with $I = \sum_{n=1}^{N} \frac{I_n(I_n+1)}{2}$ and $J = \prod_{n=1}^{N} I_n$. The right-hand side $\mathbf{b} \in \mathbb{K}^I$ is defined as

$$\mathbf{b} = \left[\mathrm{triu}\left(\boldsymbol{\Sigma}^{(1)}\right); \mathrm{triu}\left(\boldsymbol{\Sigma}^{(2)}\right); \cdots; \mathrm{triu}\left(\boldsymbol{\Sigma}^{(N)}\right)\right]$$

in which each entry is either zero or a squared multilinear singular value. One can show that the Jacobian for this particular problem is also a sparse matrix of size $I \times J$ with $\sum_{n=1}^{N} I_n$ nonzeros in each column. More specifically, the Jacobian has the form: $\mathbf{J} = \mathbf{J}^{(1)} + \mathbf{J}^{(2)}$ with $\mathbf{J}^{(1)}$ and $\mathbf{J}^{(2)}$ the derivative to the first and second $\mathbf{u}$, respectively. Computing the sub-Jacobians $\mathbf{J}^{(n)}$ is reduced to filling in elements of $\mathcal{T}$ at the correct position in $\mathbf{J}^{(n)}$ for the orthogonality constraints. For the multilinear singular value constraints, one has to multiply by two. By exploiting the structure, no additional operations are required. We implemented this using a C/mex function that replaces entries to avoid the overhead of constructing sparse matrices in MATLAB. The Gramian of the Jacobian is computed using sparse matrix-vector products.

We compare the optimized NLS algorithm with the alternating projection method (APM) [131] in terms of computation time needed to construct an $I_1 \times I_2 \times I_3$ tensor with given multilinear singular values. We take $I_1 = I_2 = 10\alpha$ and $I_3 = 5\alpha$ in which $\alpha = \{1, 5, 10\}$. The multilinear singular

Table 8.3: The LS-CPD method for constructing a tensor with particular multilinear singular values is faster than APM. *This is illustrated by comparing the median computation time (in seconds) across 20 experiments for an* $I_1 \times I_2 \times I_3$ *tensor with* $I_1 = I_2 = 10\alpha$ *and* $I_3 = 5\alpha$ *in which* $\alpha = \{1, 5, 10\}$.

	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$
Alternating projection method (APM) [131]	0.100	34.4	1747
Construction of $\mathbf{A}$	0.004	1.4	23
Initialization (i.e., one iteration of APM)	0.002	0.4	12
LS-CPD	0.023	15.4	444
Total computation time of LS-CPD	0.029	17.2	479

values are chosen by constructing a tensor that can be written as a sum of a multilinear rank-$(L_1, 1, L_1)$ term and a multilinear rank-$(1, L_2, L_2)$ term with $L_1 = I_3 - 1$ and $L_2 = I_3 + 1$. The elements of the factor matrices are drawn from the standard normal distribution. We normalize the multilinear singular values such that the tensor has unit Frobenius norm. We initialize the NLS algorithm with the solution obtained after one iteration of APM. In Table 8.3, we report the median computation time across 20 experiments. The time to construct $\mathbf{A}$ is reported separately because it depends only on the size of the tensor and its construction has to be performed only once. Clearly, the computation time of LS-CPD is much lower than APM, even if we include the time needed to construct $\mathbf{A}$.

8.5.3 Blind deconvolution of constant modulus signals

LS-CPDs can also be used in signal processing applications. We illustrate this by reformulating the blind deconvolution of a constant modulus (CM) signal [297] as the computation of an LS-CPD. In this chapter, we investigate the single-input-single-output (SISO) case using an autoregressive (AR) model [192], i.e., we have:

$$\sum_{l=0}^{L} w_l \cdot y[k-l] = s[k] + n[k], \text{ for } 1 \leq k \leq K, \tag{8.25}$$

with $y[k]$, $s[k]$, and $n[k]$ the measured output, the input, and the additive noise at the kth time instance, respectively. The lth filter coefficient is denoted as w_l. Assume we have $K + L - 1$ samples $y[-L+1], \ldots, y[K]$ and let $\mathbf{Y} \in \mathbb{K}^{L \times K}$ be a Toeplitz matrix defined as $y_{lk} = y[k-l]$. Also, the filter coefficients are collected in $\mathbf{w} \in \mathbb{K}^L$ and the source vector $\mathbf{s} \in \mathbb{K}^K$ is defined as $s_k = s[k]$. We ignore the noise in the derivation of our method for simplicity. Equation (8.25) can then be expressed in matrix form as:

$$\mathbf{Y}^{\mathrm{T}}\mathbf{w} = \mathbf{s}. \tag{8.26}$$

The goal of blind deconvolution is to find the vector $\mathbf{w}$ using only the measured output values [1]. In order to make this problem identifiable, additional prior knowledge has to be exploited. Here, we assume that the input signal has constant modulus (CM), i.e., each sample s_k satisfies the following property [87]:

$$|s_k|^2 = s_k \cdot \overline{s}_k = c, \text{ for } 1 \leq k \leq K \tag{8.27}$$

with c the squared constant modulus which is known *a priori*. By using (8.26) in (8.27), we obtain:

$$(\mathbf{y}_k^{\mathrm{T}}\mathbf{w})\left(\overline{\mathbf{y}_k^{\mathrm{T}}\mathbf{w}}\right) = c, \quad \text{or, equivalently,} \quad (\mathbf{y}_k \otimes \overline{\mathbf{y}}_k)^{\mathrm{T}}(\mathbf{w} \otimes \overline{\mathbf{w}}) = c, \tag{8.28}$$

in which $\mathbf{y}_k$ is the kth column of $\mathbf{Y}$, for $1 \leq k \leq K$. Taking into account all equations, (8.28) reduces to the following LS-CPD:

$$\left(\mathbf{Y} \odot \overline{\mathbf{Y}}\right)^{\mathrm{T}}(\mathbf{w} \otimes \overline{\mathbf{w}}) = c \cdot \mathbf{1}_K. \tag{8.29}$$

We illustrate the approach by means of the following straightforward example. Consider an AR model of degree $L = 5$ with uniformly distributed coefficients between zero and one, sample length $K = 100$, and $c = 1$. We perturb the measurements with additive Gaussian noise which is scaled to obtain a particular signal-to-noise ratio (SNR). We solve (8.29) by relaxing $\overline{\mathbf{w}}$ to $\mathbf{v}$:

$$\left(\mathbf{Y} \odot \overline{\mathbf{Y}}\right)^{\mathrm{T}}(\mathbf{w} \otimes \mathbf{v}) = c \cdot \mathbf{1}_K \tag{8.30}$$

using the NLS algorithm with the Jacobi PC and starting from the algebraic solution. In Figure 8.7, we report the median relative error on $\mathbf{w}$ and the median run-time across 50 experiments for several values of the signal-to-noise ratio (SNR). We compare our approach to the naive method, i.e., the method that relaxes the Kronecker structure in (8.30), solves the system, and subsequently fits the Kronecker structure to the least-squares solution. These are the core elements of the well-known analytical constant modulus algorithm (ACMA) [297], [298]. We also compare with a state-of-the-art SISO CM algorithm (CMA) called optimal step-size CMA (OSCMA) [282], [313]. It is clear that our *generic* LS-CPD method obtains more accurate results than the relaxation-based technique and achieves similar accuracy as the *dedicated* OSCMA method. Also, the run-time of the LS-CPD approach is only slightly higher than the OSACM method in this example, but can be further reduced by exploiting the structure in the coefficient matrix.

8.6 Conclusion and future research

We presented a new framework for linear systems with a CPD constrained solution (LS-CPD). We defined the LS-CPD problem, discussed links between particular types of structured coefficient matrices and the CPD problem, and

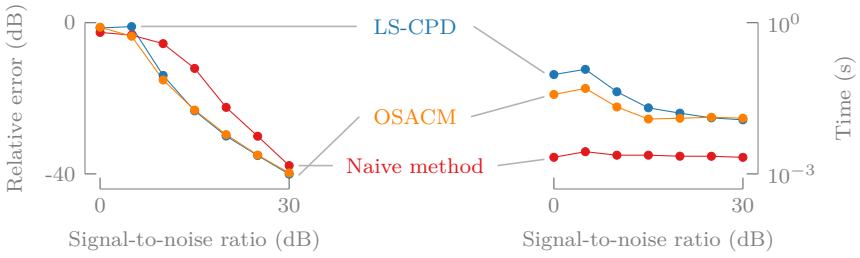


Figure 8.7: The LS-CPD approach obtains more accurate results than the naive method and achieves similar accuracy as the dedicated OSACM method. The run-time of LS-CPD is slightly higher than OSACM for this example. *The naive method has a lower run-time than the other methods.*

derived a condition guaranteeing generic uniqueness of the solution. In contrast to the naive method, the proposed algebraic and optimization-based methods allow one to solve the LS-CPD problem in the underdetermined case. Although we focused on the Gauss–Newton (GN) method, the derivations of the expressions for the objective function, gradient, Gramian, and Gramian-vector product can also be used to implement various nonlinear least squares and quasi-Newton algorithms. Numerical experiments show that the algebraic method is a good starting point for optimization-based methods. We also compared the effectiveness of two preconditioners for the GN method. The wide applicability of LS-CPDs is illustrated with three applications from classification, multilinear algebra, and signal processing. Importantly, we have explained that many classification tasks can be formulated as the computation of an LS-CPD. In order to reduce the per-iteration computational complexity of the NLS algorithm, application-dependent structure can be exploited, as we have shown with the construction of tensors that have particular singular values. The focus of this chapter was on CPD constrained solutions, however, our framework can be extended to other tensor decompositions such as the MLSVD, TT or HT models.

Part III

Applications

Efficient use of CALPHAD based data in phase-field spinodal decomposition simulations for a quaternary system through decomposed thermodynamic tensor models

9

ABSTRACT | A successful coupling of the phase-field and CALPHAD methods for a thermodynamic consistent description of the system's free energy in the phase-field model is challenging for multicomponent alloys. The many coupling schemes presented in the literature all tend to suffer from inefficiencies and limitations when applied to higher-order systems. This chapter presents a novel coupling approach treating the collection of data calculated as a function of composition with CALPHAD models as thermodynamic data tensors. As the number of entries in the tensor depends exponentially on the number of components of the system, it quickly becomes unfeasible to obtain or store such a tensor. To break this dependency, we compute a canonical polyadic decomposition using few samples of this tensor, allowing the thermodynamic tensors to be represented with a good accuracy as a sum of rank-1 terms of which the size will only grow linearly with the number of components considered in the model. The gain in efficiency and the data reduction obtained with the tensor decomposition in the novel coupling scheme will thus increase when more and more elements are considered in the simulations. The performance and viability of this coupling scheme is analyzed for spinodal decomposition simulations for liquid Ag–Cu–Ni–Sn alloys.

This chapter is based on Y. Coutinho, N. Vervliet, L. De Lathauwer, and N. Moelans, "Efficient use of CALPHAD based data in phase-field spinodal decomposition simulations for a quaternary system through decomposed thermodynamic tensor models", Technical Report 18–51, ESAT-STADIUS, KU Leuven, Belgium, 2018. The figures have been updated for consistency.

9.1 Introduction

The understanding of a material's behavior and structure over its life cycle is critical for defining its range of applications. Characterization of a material's microstructure often provides valuable insights on the mechanical, electrical, thermal and other properties. Deviating from a more traditional alloy design approach, multicomponent alloys have taken the attention of many materials scientists and metallurgists in recent years as a new field for exploration. High-entropy alloys, superalloys, bulk metallic glasses, lead-free solders and complex concentrate alloys, among others, tend to present properties not usually found in unary or binary alloys. Due to the number of chemical elements involved when designing multicomponent alloys, even small adjustments to one or more of the components concentrations can lead to distinguished microstructures and properties. However, the number of components also represents the biggest challenge for alloy design. Most of the tools and methodologies developed for traditional alloys are not compatible with or not efficient enough for the study of multicomponent systems.

The phase-field method is a powerful tool for simulating microstructure evolution in materials. In this technique, microstructural features are defined by phase-field variables which are functions of space and time. The evolution of these variables is governed by two types of partial differential equations (PDEs): the Allen-Cahn equation [10], which controls the evolution of non-conserved variables, typically representing the grains or domains structure and orientation, and the Cahn-Hilliard equation [50] for conserved phase-field variables, which describes local composition and has a mass conservation constraint imposed. The greatest strength of the phase-field method is that the microstructure evolution is guided by the system's free energy, which can be provided as function of composition and temperature, allowing the models to be heavily parameterized and to represent complex physics with fidelity. The formulation of a free energy expression is not a simple task and grows in complexity with every element added to the system, representing a challenge for the successful application of phase-field models when simulating the behavior of multicomponent alloys.

For a thermodynamic consistent description of the free energy density of the coexisting phases, the phase field method is frequently coupled with CALPHAD models [168]. In this phenomenological methodology, Gibbs free energy expressions are formulated as a function of the molar fraction of the system components and temperature with polynomials fitted to experimental data and sometimes ab-initio calculations. The main advantage is that the description of higher-order systems can be extrapolated from binary and ternary interaction parameters with good accuracy. This makes the CALPHAD method a desirable tool to be coupled with multicomponent phase-field simulations. In practice, different approaches have been proposed to implement this coupling of a phase-field model with CALPHAD data to fa-

cilitate microstructure evolution simulations for multicomponent systems of which the most noticeable are:

- directly using the composition and temperature-dependent CALPHAD expressions for the Gibbs energies of the different phases and their derivatives w.r.t. the molar fractions of the different components in the phase-field free energy equations [104], [127], [174];
- coupling the phase-field implementation with CALPHAD based thermodynamic software through programming interfaces [55], [122] to evaluate the Gibbs free energies and chemical potentials for discrete volume elements in the system and as a function of time;
- using polynomial expressions of the composition dependence on the free energy fitted to thermodynamic data and phase equilibria in the phase-field free energy [128], [153], [320];
- use of tables [141] of precalculated Gibbs energies and chemical potentials in discrete composition points.

The complexity of a coupling by directly using the CALPHAD expression is highly dependent on which models are used to describe the solution's thermodynamics. It is, for example, relatively simple if all phases show the behavior of a substitutional solution. However, for phases for which certain atoms occupy preferential positions (or sublattices), it becomes computationally expensive or even impossible to implement minimization procedures which relate the phase-field concentration variables with the lattice site fractions [234]. Furthermore, if the interaction parameters are stored in a commercial database, their values are often not accessible by the user.

Coupling with thermodynamic software using a programming interface is straightforward to implement, but accessing an external program at each time step in each cell of the simulation domain consumes computing resources valuable for the phase-field simulation. For this reason, an optimization procedure has been proposed where the thermodynamic functions are approximated at intermediate compositions by interpolation based on function values evaluated for nearby compositions [33] with the objective of reducing the number of times that the external software is accessed. However, the complexity and computational cost of multidimensional interpolation schemes increases drastically with the number of components [182].

Fitting a polynomial to the free energy behaves poorly when larger ranges of temperature and composition are considered and often leads to unphysical representations of the thermodynamic properties. It is, for example, difficult to prevent concentrations or molar fractions from taking unphysical values, i.e., below 0 or above 100%.

Using tables directly in the phase-field simulations is still manageable for isothermal systems with three components. However, as the order of the

system increases, the amount of data that needs to be generated or stored in these tables grows exponentially. This makes the use of tables extremely inefficient, since large amounts of computer memory are required during simulations. Moreover, this issue becomes worse if more components are considered, which is often referred to as the curse of dimensionality [304]. In fact, for quaternary or higher-order systems, these tables or multiway numerical arrays constitute what is known in multilinear algebra as tensors. To avoid confusion with the common meaning of a tensor in mechanics (i.e., stress tensor) [170], the term thermodynamic data tensor is used here.

For all existing coupling schemes, the advantages found for binary and ternary systems, such as straightforward implementation, speed and accuracy disappear for higher-order systems. In this chapter, we propose an alternative method: the use of a tensor decomposition is explored as an efficient alternative to the data extracted from CALPHAD databases in phase-field simulations.

A canonical polyadic decomposition (CPD) with linearly constrained factors is used to model the thermodynamic tensors as sums of rank-1 terms. It is shown that such tensors are incomplete due to a molar fraction constraint and that by using the decomposition the curse of dimensionality is broken, allowing the tensor to be represented with a number of coefficients which grows linearly as a function of the number of components instead of exponentially.

The objective of this chapter is to introduce a new approach for the use of multicomponent CALPHAD based data in phase-field simulations via a thermodynamic tensor decomposition. The efficiency of this novel coupling scheme is illustrated with phase-field spinodal decomposition simulations of a liquid alloy composed of Ag–Cu–Ni–Sn, which exhibits a miscibility gap at 1400 K. In section 9.2 the phase-field model used for the simulations is described, followed by the CALPHAD thermodynamic model in section 9.3. Next, in section 9.4, the concept of a thermodynamic tensor and its decomposition into rank-1 terms are introduced, and the practical details of the computation of the model discussed in subsection 9.4.1. Section 9.5 contains details about the phase-field simulation and how the different coupling schemes with CALPHAD data are implemented; see Figure 9.3 for a schematic overview. The performance and viability of the tensor model is verified in section 9.6 and it is shown that its accuracy is comparable to that of a full CALPHAD model expression. Its use is as simple and intuitive as a polynomial approximation while avoiding unphysical effects. Finally, its implementation is as straightforward as the use of tables or a thermodynamic software interface, but without the consequences caused by the curse of dimensionality.

9.2 Phase-field model

Spinodal decomposition simulations are often performed as a first step when developing or implementing new phase-field models and techniques, as composition fields suffice to track the microstructure [154], [173], [174], [182], [239], [280]. On a volume-fixed frame of reference [13] and for a quaternary system ($C = 4$), $N = C - 1 = 3$ conserved variables are needed. Let x_1 be the molar fraction of silver, x_2 that of copper and x_3 that of nickel. The molar fraction x_d of tin, the dependent component, can be obtained from $x_d = 1 - x_1 - x_2 - x_3$. The evolution of the independent conserved variables $x_i(r,t)$ with $i = 1, 2, 3$, in space, defined by position vector $\mathbf{r}$, and time t are evaluated with a Cahn-Hilliard type of equation [199]:

$$\left(\frac{1}{V_\mathrm{m}}\right)\frac{\partial x_i(\mathbf{r},t)}{\partial t} = -\nabla \cdot J_i, \tag{9.1}$$

in which V_m is a constant coefficient approximating the molar volume of the alloy. $\nabla \cdot J_i$ is the gradient of the diffusion flux of component i w.r.t. the spatial coordinates and is related to the generalized diffusion potentials $(\delta F/\delta x_j)$ of all independent components [13]:

$$J_i = \sum_j^N -M_{ij}\nabla\frac{\delta F}{\delta x_j}. \tag{9.2}$$

The kinect paramater M_{ij} links the flux of i with all the driving forces. As data on diffusion coefficients for quaternary systems is scarce, a simplified dependency on the concentration assuming an equal atomic mobility β for all components in a phase is adopted [13], [174]:

$$M_{ij} = \begin{cases} \frac{1}{V_\mathrm{m}}\left[\beta x_i(1-x_i)\right], & i = j \\ \frac{1}{V_\mathrm{m}}\left[\beta x_i(-x_i)\right], & i \neq j. \end{cases}$$

The free energy functional F can be expanded for a quaternary nonuniform system [50] as

$$F(x_1,x_2,x_3) = \int_V f_0(x_1,x_2,x_3) + \sum_i^C \frac{k_i}{2}(\nabla x_i)^2 \mathrm{d}V, \tag{9.3}$$

where $f_0(x_1,x_2,x_3)$ is the homogeneous free energy density of the phase as a function of the molar fraction of the independent components and k_i is the positive energy gradient coefficient for component i. The functional

derivative of F w.r.t. each component $i = 1, 2, 3$, is given by

$$\frac{\delta F}{\delta x_i} = \frac{\partial f_0(x_1, x_2, x_3)}{\partial x_i} - k_i \nabla^2 x_i. \tag{9.4}$$

The thermodynamic relation [104]

$$\frac{\partial f_0}{\partial x_i} = \frac{1}{V_\mathrm{m}}(\mu_i - \mu_d) = \frac{\tilde{\mu}_i}{V_m}, \tag{9.5}$$

can be used, in which $\tilde{\mu}_i$ is the diffusion potential of component i. This diffusion potential $\tilde{\mu}_i$ is equal to the chemical potential μ_i of element i minus the chemical potential of the dependent component μ_d. The equations simulating the evolution of the conserved variables $x_i(\mathbf{r}, t)$ are obtained by substituting (9.2), (9.4), and (9.5) in (9.1):

$$\left(\frac{1}{V_\mathrm{m}}\right) \frac{\partial x_i(\mathbf{r}, t)}{\partial t} = \nabla \sum_j^C M_{ij} \nabla \left(\frac{\tilde{\mu}_i}{V_\mathrm{m}} - k_i \nabla^2 x_i\right); \tag{9.6}$$

in which the model parameters M_{ij}, k_i and $\tilde{\mu}_i$ for $i = 1, 2, 3$, are needed as input.

9.3 CALPHAD thermodynamic model

The molar Gibbs free energy expression of the liquid phase as a function of the molar fractions of the elements x_i and temperature T is modeled using the CALPHAD methodology with a random substitutional solution model [234] consisting of the terms:

$$G_\mathrm{m} = G^\mathrm{o} + G^\mathrm{id}_\mathrm{mix} + G^\mathrm{xs}_\mathrm{mix}, \tag{9.7}$$

in which G^o represents the Gibbs free energy of a mechanical mixture of all pure components i:

$$G^\mathrm{o} = \sum_{i=1}^{C} x_i G^\mathrm{o}_i, \tag{9.8}$$

in which G^o_i is the temperature-dependent energy per mole for component i for the liquid phase w.r.t. the SGTE [94] reference state, and C is the number of components. $G^\mathrm{id}_\mathrm{mix}$ gives the entropy of mixing assuming ideal substitutional solution behavior:

$$G^\mathrm{id}_\mathrm{mix} = \bar{R}T \sum_{i=1}^{C} x_i \log_\mathrm{e} x_i, \tag{9.9}$$

with $\bar{R}$ the ideal gas constant[1] and $G^{\text{xs}}_{\text{mix}}$ is the term modeling the behavior deviating from an ideal solution, which is described by a Redlich-Kister-Muggianu polynomial [202]. Expanded for a quaternary system with a dependency on the molar fractions of all four elements, $G^{\text{xs}}_{\text{mix}}$ is given by

$$\begin{aligned} G^{\text{xs}}_{\text{mix}} = &\sum_i \sum_{j>i} x_i x_j \sum_v L^v_{ij}(x_i - x_j)^v \\ &+ \sum_{k>j} x_i x_j x_k (x_i L^0_{ijk} + x_j L^1_{ijk} + x_k L^2_{ijk}). \end{aligned} \tag{9.10}$$

The binary interaction parameters L_{ij} are expanded based on v terms available in the thermodynamic database. L_{ijk} are the ternary interaction parameters, with the indices i, j and k going over the four elements. The temperature-dependent expressions for all binary and ternary interaction parameters L_{ij}, L_{ijk} and the Gibbs energies G^{o}_i of the pure elements for the liquid phase are obtained from the COST 531 lead-free solder thermodynamic database [177]. Quaternary interaction terms are ignored, since they can be assumed to be much smaller than the binary and ternary interaction terms.

9.4 Thermodynamic tensor model

Tensors or multiway arrays are a higher-order generalization of vectors (first order) and matrices (second order) and are indexed as $(i, j, k, \ldots)$ with as many indices as the order of the tensor [65], [243]. In the case of the thermodynamic data, the modes of the tensor can be the molar fraction of each component in the system, the temperature, the pressure or some other variables used for the calculation of some thermodynamic property.

For instance, a tensor $\mathcal{G}$ can be built by calculating and storing values of molar Gibbs free energy of a phase using the CALPHAD expressions in (9.7)–(9.10). At a constant temperature, this tensor will have a number of modes or its order equal to the number of independent components N. Since the molar fractions of the system are always constrained to $x_1 + x_2 + x_3 + \ldots + x_N + x_d = 1$, this tensor is necessarily incomplete [304], because entries with unphysical combinations of molar fractions cannot be calculated, as can be seen in Figure 9.1 for a quaternary system.

Furthermore, the choice of a step size δ_x, with which data is collected, dictates the intervals in between the entries of the tensor and consequently the number of points where the calculations are taken. This step size can differ for each mode of the tensor, but for simplicity and since all modes are representing molar fractions, the same δ_x is used along all modes, in this chapter.

[1] $\bar{R}$ is used in this chapter for the ideal gas constant to avoid confusion with the rank parameter R, which is introduced in the next section.

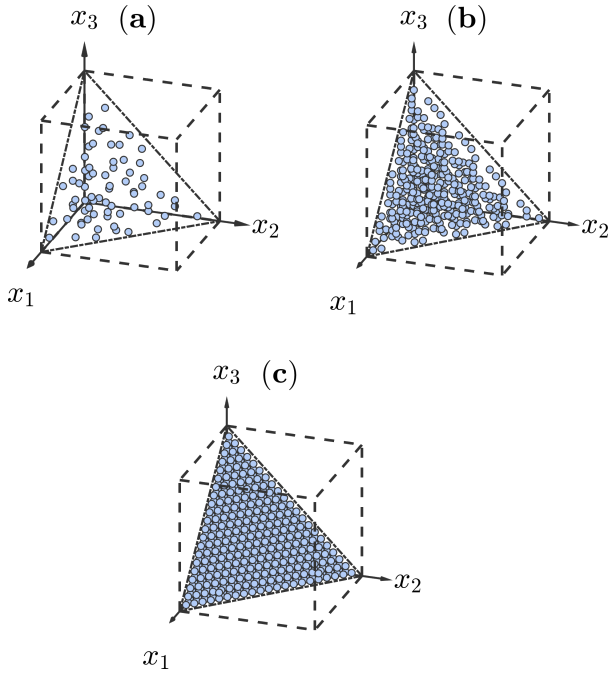


Figure 9.1: Representation of the Gibbs free energy tensor with different resolutions used in the decomposition model. (**a**) *is the training set* $\mathcal{G}_\mathcal{T}$ *used as input in the optimization procedure.* (**b**) *is the validation set* $\mathcal{G}_\mathcal{V}$ *used to test the approximation of the coefficients and to avoid overfitting.* (**c**) *is a tensor with a resolution as desired to be used in the simulations* $\mathcal{G}_\mathcal{S}$*. Note that the illustrations do not represent the real number of points in these tensors.*

The greatest challenge of using tensors to store and provide phase-field simulations with precalculated molar Gibbs free energy information is their exponential dependency on its order [304]. However, by modeling the tensor with a low rank decomposition, this dependency is broken and becomes linear, enabling thermodynamic data from phases containing multiple components to be supplied to phase-field simulations efficiently.

In a polyadic decomposition (PD) a tensor $\mathcal{G}$ of order N is written as a sum of R rank-1 terms, each being the outer product of N nonzero factor vectors $\mathbf{a}_r^{(n)}$, $n = 1, \ldots, N$ (see Figure 9.2 for $N = 3$):

$$\mathcal{G} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \mathbf{a}_r^{(2)} \otimes \cdots \otimes \mathbf{a}_r^{(N)}.$$

(In the remainder of this chapter, we focus on third-order tensors, i.e., $N = 3$.) When the number R of rank-1 terms is minimal to exactly represent the

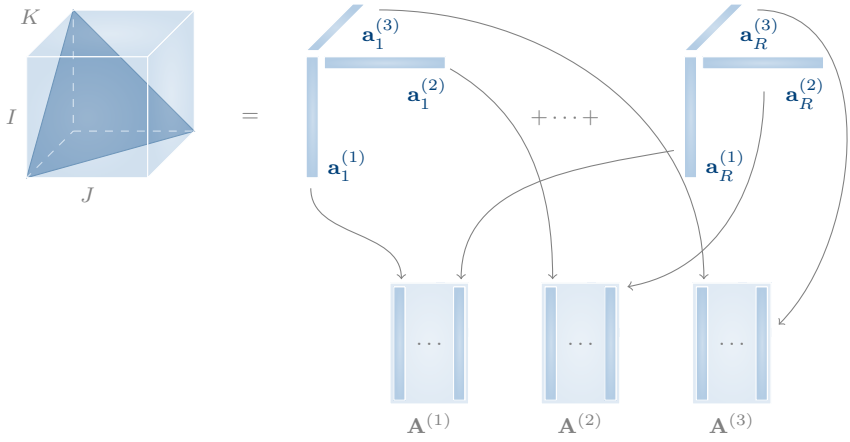


Figure 9.2: Illustration of a canonical polyadic decomposition (CPD). *The third-order incomplete tensor with dimensions $I \times J \times K$ is written as a sum of R rank-1 terms, each of which is the outer product of three factor vectors $\mathbf{a}_r^{(1)}$, $\mathbf{a}_r^{(2)}$ and $\mathbf{a}_r^{(3)}$. The collection of all R factor vectors of a given mode leads to the construction of a factor matrix with dimension $I \times R$, $J \times R$ or $K \times R$.*

tensor, the polyadic decomposition is canonical (CPD). While determining the exact rank is a hard problem in general, for specific applications an approximation with a low rank R is often sufficient and allows the tensor to be modeled with a desired accuracy [65], [243].

By collecting all R factor vectors for a given mode n as columns of a factor matrix $\mathbf{A}^{(n)} = [\mathbf{a}_r^{(n)}, \ldots, \mathbf{a}_r^{(n)}]$, the CPD can be represented compactly as:

$$\mathcal{G} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}]\!].$$

Each factor matrix $\mathbf{A}^{(n)}$ models the contribution of each mode to the tensor and its number of columns is equal to the number of rank-1 terms R. If only one entry of the tensor is needed, it is not required to calculate the entire tensor. The information of a single entry of the tensor can be obtained as

$$\mathcal{G}_{ijk} = \sum_{r=1}^{R} a_{i,r}^{(1)} \cdot a_{j,r}^{(2)} \cdot a_{k,r}^{(3)}.$$

The coupling with phase-field simulations takes advantage of this aspect; see subsection 9.5.3.

As the Gibbs free energy is a smooth function, this prior knowledge can be exploited by imposing low degree polynomial constraints on the factor vectors. More specifically, each entry of a factor matrix $a_{i,r}^{(n)}$ can be computed

as the dot product between two vectors:

$$a_{i,r}^{(n)} = \mathbf{b}_i^{\mathrm{T}} \cdot \mathbf{u}_r^{(n)}.$$

If the vector $\mathbf{b}$ contains evaluated monomials in x and $\mathbf{u}_r^{(n)}$ is a vector with coefficients, we have

$$a_{i,r}^{(n)} = \begin{bmatrix} x_i^d & x_i^{(d-1)} & \dots & x_i^1 & x_i^0 \end{bmatrix} \cdot \begin{bmatrix} u_{d,r}^{(n)} \\ u_{(d-1),r}^{(n)} \\ \vdots \\ u_{1,r}^{(n)} \\ u_{0,r}^{(n)} \end{bmatrix},$$

which is the representation of a polynomial with degree d:

$$a_{i,r}^{(n)} = u_{d,r}^{(n)} x_i^d + u_{(d-1),r}^{(n)} x_i^{(d-1)} + \ldots + u_{1,r}^{(n)} x_i^1 + u_{0,r}^{(n)} x_i^0. \tag{9.11}$$

The entries in $\mathbf{b}_i$ are related to the steps defined by δ_x with

$$\mathbf{x} = \begin{bmatrix} \delta_x & 2\delta_x & \dots & (1 - N\delta_x) \end{bmatrix},$$

hence $x_i = i\delta_x$. The vector $\mathbf{x}$ contains only known information and is equal for all modes since the same molar fraction step δ_x is chosen. The coefficients in $\mathbf{u}_r^{(n)}$ describe the individual and unknown contribution of each mode, in other words, $\mathbf{u}_r^{(n)}$ models the contribution of G_i^{o} in (9.8) and the binary and ternary interaction parameters in (9.10) to the tensor. In terms of the factor matrices the constraint can also be written as:

$$\mathbf{A}^{(n)} = \mathbf{B}\mathbf{U}^{(n)}, \tag{9.12}$$

with matrix $\mathbf{B}$ having $d + 1$ columns and with the number of rows being related to the choice of step size δ_x and matrix $\mathbf{U}^{(n)}$ with a size of $d+1$ rows by R columns. Using cross validation, a polynomial of degree $d = 4$ is used in this chapter. A later inspection of the full CALPHAD expression (9.7) confirmed that the maximum degree of the molar fractions variables $d = 4$. Furthermore, by using this degree in the polynomial constraint, the matrix form leads to the known matrix $\mathbf{B}$:

$$\mathbf{B} = \begin{bmatrix} \mathbf{x}^4 & \mathbf{x}^3 & \mathbf{x}^2 & \mathbf{x}^1 & \mathbf{x}^0 \end{bmatrix},$$

in which the powers are taken element-wise. By taking the first and second derivatives of (9.11) w.r.t. x, matrices $\dot{\mathbf{B}}$ and $\ddot{\mathbf{B}}$, which model the first and

second derivatives of the Gibbs free energy are obtained:

$$\dot{\mathbf{B}} = \begin{bmatrix} 4\mathbf{x}^3 & 3\mathbf{x}^2 & 2\mathbf{x} & 2 & 0 \end{bmatrix},$$
$$\ddot{\mathbf{B}} = \begin{bmatrix} 12\mathbf{x}^2 & 6\mathbf{x} & 6 & 0 & 0 \end{bmatrix},$$

while the unknown coefficients in (9.11) are not affected by the derivatives and are the same for a given mode. These coefficients of each unknown matrix $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$ for a presumed rank R are the terms needed to be obtained to model the tensor:

$$\mathbf{U}^{(n)} = \begin{bmatrix} u_{4,1} & \dots & u_{4,R} \\ u_{3,1} & \dots & u_{3,R} \\ u_{2,1} & \dots & u_{2,R} \\ u_{1,1} & \dots & u_{1,R} \\ u_{0,1} & \dots & u_{0,R} \end{bmatrix}.$$

The Gibbs free energy tensor $\mathcal{G}$, the diffusion potentials tensor $\mathcal{P}^{(i)} = \partial\mathcal{G}/\partial x_i$ w.r.t. the three independent molar fractions and the derivatives of the diffusion potentials $\mathcal{D}^{(i,j)} = \partial\mathcal{P}^{(i)}/\partial x_j$ can then be obtained by using a combination of the matrices $\mathbf{B}$, $\dot{\mathbf{B}}$ and $\ddot{\mathbf{B}}$ with $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$, e.g.,

$$\begin{aligned} \mathcal{G} &= [\![\mathbf{B}\mathbf{U}^{(1)}, \mathbf{B}\mathbf{U}^{(2)}, \mathbf{B}\mathbf{U}^{(3)}]\!], \\ \mathcal{P}^{(1)} &= [\![\dot{\mathbf{B}}\mathbf{U}^{(1)}, \mathbf{B}\mathbf{U}^{(2)}, \mathbf{B}\mathbf{U}^{(3)}]\!], \\ \mathcal{D}^{(2,2)} &= [\![\mathbf{B}\mathbf{U}^{(1)}, \ddot{\mathbf{B}}\mathbf{U}^{(2)}, \mathbf{B}\mathbf{U}^{(3)}]\!], \\ \mathcal{D}^{(1,3)} &= [\![\dot{\mathbf{B}}\mathbf{U}^{(1)}, \mathbf{B}\mathbf{U}^{(2)}, \dot{\mathbf{B}}\mathbf{U}^{(3)}]\!]. \end{aligned}$$

This procedure can be equivalently repeated for all components $i, j = 1, 2, 3$, to obtain the derivatives w.r.t. all system components. Hence, in order to describe all data only three coefficient matrices $\mathbf{U}^{(n)}$ are needed and often this is done by using optimization procedures as the one discussed in the next section. Once the coefficients are estimated, the factor matrices $\mathbf{A}^{(n)}$ which hold the information about the system can be calculated with (9.12). The factor matrices for a given number of modes and rank R are referred to as the rank-R thermodynamic tensor model (TTM).

9.4.1 Tensor model computation

Using Thermo-Calc TC-Toolbox and a MATLAB script, tensors for the Gibbs free energy, diffusion potentials (9.5) and derivatives w.r.t. all components are collected for the liquid phase at 1400 K using a series of nested for-loops and the COST 531 database for the Ag–Cu–Ni–Sn system with a step size $\delta_x = 0.0005$. On an Intel i7-7700 computer with 16 GB of RAM it requires 21 min to collect all ten tensors (as seen below) simultaneously, resulting in

a mat-file with a size of 80 MB:

$$\begin{array}{ccc} & \mathcal{G}_{\mathcal{V}}, & \\ \mathcal{P}_{\mathcal{V}}{}^{(1)}, & \mathcal{P}_{\mathcal{V}}{}^{(2)}, & \mathcal{P}_{\mathcal{V}}{}^{(3)}, \\ \mathcal{D}_{\mathcal{V}}{}^{(1,1)}, & \mathcal{D}_{\mathcal{V}}{}^{(2,2)}, & \mathcal{D}_{\mathcal{V}}{}^{(3,3)}, \\ \mathcal{D}_{\mathcal{V}}{}^{(1,2)}, & \mathcal{D}_{\mathcal{V}}{}^{(2,3)}, & \mathcal{D}_{\mathcal{V}}{}^{(3,1)}. \end{array}$$

These tensors are then assigned as validation sets $\mathcal{V}$, and have because of the choice of step size 1 290 240 samples. From each diffusion potential tensor, subsets are extracted by taking every fifth data point and after filtering some boundary samples, three new tensors are obtained with 11 482 entries each. These tensors are used as training sets $\mathcal{T}$ when computing the decomposition. One extra preprocessing step is to subtract from the training subsets the contribution from the Gibbs free energy term for ideal mixing (9.9), giving:

$$\mathcal{P}_{\mathcal{T}}{}^{(1)}, \quad \mathcal{P}_{\mathcal{T}}{}^{(2)}, \quad \mathcal{P}_{\mathcal{T}}{}^{(3)}.$$

This procedure is conducted to avoid the modeling of unnecessary data, since the $G_{\text{mix}}^{\text{id}}$ term is only a function of the molar fractions x_i, temperature T and the ideal gas constant $\bar{R}$. All these terms are known and can be added back after the optimization procedure. Moreover, this term has a logarithmic dependence which is difficult to model with polynomial functions.

The optimization procedure is conducted by approximating the coefficients contained in the matrices $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$ which minimizes the least squares objective function (9.13), using the data-independent variant of the linearly constrained CPD algorithm (CDPLI DI) presented in [302]. The coupling of the different tensors is implemented as described in [306];

$$\begin{aligned} \min_{\mathbf{U}^{(1)},\mathbf{U}^{(2)},\mathbf{U}^{(3)}} & \frac{1}{2}\left\|\mathcal{P}_{\mathcal{T}}{}^{(1)} - [\![\dot{\mathbf{B}}\mathbf{U}^{(1)}, \mathbf{B}\mathbf{U}^{(2)}, \mathbf{B}\mathbf{U}^{(3)}]\!]\right\|^2 \\ & + \frac{1}{2}\left\|\mathcal{P}_{\mathcal{T}}{}^{(2)} - [\![\mathbf{B}\mathbf{U}^{(1)}, \dot{\mathbf{B}}\mathbf{U}^{(2)}, \mathbf{B}\mathbf{U}^{(3)}]\!]\right\|^2 \\ & + \frac{1}{2}\left\|\mathcal{P}_{\mathcal{T}}{}^{(3)} - [\![\mathbf{B}\mathbf{U}^{(1)}, \mathbf{B}\mathbf{U}^{(2)}, \dot{\mathbf{B}}\mathbf{U}^{(3)}]\!]\right\|^2, \end{aligned} \tag{9.13}$$

in which $\mathcal{P}_{\mathcal{T}}{}^{(1)}$, $\mathcal{P}_{\mathcal{T}}{}^{(2)}$ and $\mathcal{P}_{\mathcal{T}}{}^{(3)}$ are the diffusion potentials training sets, $\mathbf{B}$ and $\dot{\mathbf{B}}$ the known matrices. The optimization is started by generating a few initializations, each by assigning random values to the coefficients in the unknown matrices $\mathbf{U}^{(n)}$. The result is then compared with the validation sets for the Gibbs free energy, potentials and derivatives, only the best initialization is kept and new ones are generated. This procedure continues until a desired accuracy is reached. Decompositions for ranks $R = 3, \ldots, 12$ are computed and the time spent on each decomposition computation varies

Table 9.1: Alloys compositions selected for spinodal decomposition simulations. *Six alloys are present and their respective molar fraction for silver (Ag), copper (Cu) and nickel (Ni). The molar fraction of tin (Sn) is dependent and can be obtained with* $x_1+x_2+x_3+x_d = 1$. *The alloys compositions are chosen at different points inside the miscibility gap of the liquid phase.*

Alloy	x_1	x_2	x_3
1	0.44	0.10	0.22
2	0.23	0.20	0.40
3	0.30	0.25	0.30
4	0.35	0.30	0.25
5	0.24	0.15	0.47
6	0.46	0.30	0.15

from 1 to 6 seconds.

This procedure only uses the diffusion potentials to model the tensor completely, meaning that no further computations are necessary for the Gibbs free energy of the derivatives of the potentials. Once the unknown coefficient matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}$ and $\mathbf{U}^{(4)}$ are obtained any entry can be calculated as explained in section 9.4. More details about the minimization algorithm and other parameters are discussed in [302, section 6].

9.5 Phase-field simulation details

In the case of an isothermal spinodal decomposition model, the CALPHAD molar Gibbs free energy at the considered temperature T and divided by a constant molar volume (9.14) is usually used as an approximation for the homogeneous free energy density in the phase-field free energy functional (9.3).

$$f_0(x_1, x_2, x_3) \cong \frac{G_\mathrm{m}(x_1, x_2, x_3, 1 - x_1 - x_2 - x_3, T)}{V_\mathrm{m}}. \tag{9.14}$$

Six alloy compositions are selected for simulations within the liquid miscibility gap of the Ag–Cu–Ni–Sn system and are listed in Table 9.1. Their respective phase diagrams and additional thermodynamic information are given as supplementary material.

The phase-field evolution (9.6) is implemented in MATLAB for 1D and 2D simulations using finite difference discretization, with an explicit Euler approach for discretization of the time and central schemed Laplacian for the spatial coordinates. Three similar scripts are prepared, the difference being in the source of the diffusion potentials. The first script couples the phase-field model with a full Gibbs free energy expression for the liquid phase using the CALPHAD model explained in section 9.3. A second script uses the coupling with Thermo-Calc's TC-Toolbox for MATLAB. Finally, the last

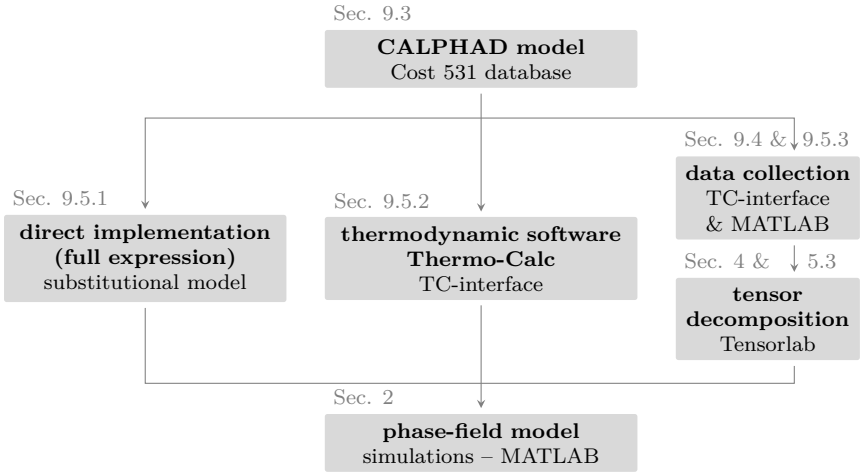


Figure 9.3: The different schemes to use a CALPHAD model in a phase-field model are compared in this chapter. *In section 9.2, the phase-field model is described and formulated with a phase free energy density expression required as input. This information is provided by a CALPHAD model explained in section 9.3. For this chapter, three different approaches are implemented and compared. First the full model expression is inserted directly in the phase-field model subsection 9.5.1. In the second scheme an interface with a thermodynamic software is used (subsection 9.5.2), where the required information will be calculated and provided to the phase-field simulation as requested. Finally, the novel approach presented in this chapter as introduced in subsection 9.5.3 is used.*

Table 9.2: Parameters used in (9.6) for conducting the phase-field simulations. *The first column displays the parameters for one-dimensional simulations, the second column for two-dimensions. Both differ only in the domain size, the 1D simulation had a simulation domain of 250 points and the 2D* 250×250 *grid points.*

Simulation parameters	1D	2D
domain size	$5 \cdot 10^{-8}$ m	$1.5 \cdot 10^{-10}$ m^2
grid spacing	$2 \cdot 10^{-10}$ m	$2 \cdot 10^{-10}$ m
duration	$1 \cdot 10^{-8}$ s	$1 \cdot 10^{-8}$ s
mobility	$1 \cdot 10^{-5}$ m/s	$1 \cdot 10^{-5}$ m/s
molar volume	$1 \cdot 10^{-5}$ m^3	$1 \cdot 10^{-5}$ m^3
energy gradient coeff.	$5 \cdot 10^{-11}$	$5 \cdot 10^{-11}$

scheme which uses the coupling with the TTMs. Additional parameters used in the phase-field simulatons are given in Table 9.2.

All simulation scripts are developed in MATLAB version R2017a and the simulations are conducted on an Intel i7-7700 computer with 16 GB of memory and Windows 10. Thermo-Calc 2017a is used for coupling with MATLAB via the TC-Toolbox. More details about each coupling schemes is presented in the following subsections.

9.5.1 Full CALPHAD model expressions coupling

Composition dependent expressions for the diffusion potentials and second derivatives are obtained from the CALPHAD Gibbs energy expression (9.7) using a symbolic differentiation function in MATLAB R2017a. These expressions of the diffusion potentials are included in the simulation program implementing the phase-field model (9.6).

9.5.2 Interface with thermodynamic software

The thermodynamic software Thermo-Calc [14] offers a TC-Toolbox for MATLAB, which is used to retrieve the values of the diffusion potentials for the current local composition at every grid point and for every time step. This information is used to evaluate (9.6) in the phase-field simulation program.

9.5.3 Use of the decomposed tensor model

Once the matrices with the unknown coefficients $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, $\mathbf{U}^{(3)}$ are estimated, the factor matrices $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, $\mathbf{A}^{(3)}$, $\dot{\mathbf{A}}^{(1)}$, $\dot{\mathbf{A}}^{(2)}$, $\dot{\mathbf{A}}^{(3)}$, $\ddot{\mathbf{A}}^{(1)}$, $\ddot{\mathbf{A}}^{(2)}$, $\ddot{\mathbf{A}}^{(3)}$ and thus the TTM can be calculated for a given rank using the known basis matrices $\mathbf{B}$, $\dot{\mathbf{B}}$, $\ddot{\mathbf{B}}$ from (9.12), for any step size δ_x, which can be chosen to be different from the step size used to generate the training tensor. A step size $\delta_x = 0.0001$ is used here, which gives thus a considerably larger number of entries compared to the validation and training tensors. This is

illustrated in Figure 9.1 (c), which represents a tensor with a higher resolution in comparison with the tensors in 9.1 (a) and (b). Note, however, that this tensor is never constructed explicitly as explained in section 9.4. This is one major advantage of using a decomposition technique, with the TTM estimated, thermodynamic information of a point in the system can be calculated by finding the row on the respective factor matrix that corresponds to that mode value. By collecting the contribution of all modes any required information can be obtained with equations (9.15)–(9.18). The Gibbs free energy ideal mixing contribution G^{id}_{mix} subtracted in the preprocessing step needs to be added back to the Gibbs free energy, potentials and derivatives by also considering the derivatives of these terms:

$$\mathcal{G}(i,j,k) = \sum_{r=1}^{R} a^{(1)}_{i,r} \cdot a^{(2)}_{j,r} \cdot a^{(3)}_{k,r} + \bar{R}T \sum_{i} x_i \log_e x_i + x_d \log_e x_d, \tag{9.15}$$

$$\mathcal{P}^{(1)}(i,j,k) = \sum_{r=1}^{R} \dot{a}^{(1)}_{i,r} \cdot a^{(2)}_{j,r} \cdot a^{(3)}_{k,r} + \bar{R}T(\log_e x_1 - \log_e x_d), \tag{9.16}$$

$$\mathcal{D}^{(2,2)}(i,j,k) = \sum_{r=1}^{R} a^{(1)}_{i,r} \cdot \ddot{a}^{(2)}_{j,r} \cdot a^{(3)}_{k,r} + \bar{R}T\left(\frac{1}{x_2} + \frac{1}{x_d}\right), \tag{9.17}$$

$$\mathcal{D}^{(1,3)}(i,j,k) = \sum_{r=1}^{R} \dot{a}^{(1)}_{i,r} \cdot a^{(2)}_{j,r} \cdot \dot{a}^{(3)}_{k,r} + \bar{R}T\frac{1}{x_d}. \tag{9.18}$$

This procedure can be equivalently taken for the other diffusion potentials and second-order derivatives.

9.6 Results and discussion

9.6.1 Validation for the tensor model

The key advantage of using a decomposition to represent thermodynamic tensors can be seen in Figure 9.4, which gives for an increasing order the number of entries in a tensor collected with a molar fraction step size of $\delta_x = 0.0001$ and the number of coefficients in TTMs with $R = 3, 6, 9$ representing the same data.

To assess the accuracy of the TTMs with different ranks, a comparison with data calculated with TC-Toolbox is conducted. By using the TTM, tensors of the Gibbs free energy, diffusion potentials and derivatives are constructed with the step size $\delta_x = 0.0001$ and these are referred as $\mathcal{G}_{\mathcal{S}}$, $\mathcal{P}_{\mathcal{S}}{}^{(i)}$ and $\mathcal{D}_{\mathcal{S}}{}^{(i,j)}$, $i, j = 1, 2, 3$. Equivalently, tensors with the same step size are collected using TC-Toolbox to be used as reference and are named $\mathcal{G}_{\mathcal{H}}$, $\mathcal{P}_{\mathcal{H}}{}^{(i)}$ and $\mathcal{D}_{\mathcal{H}}{}^{(i,j)}$. Both tensors have a high resolution, occupy a large size as a computer file and

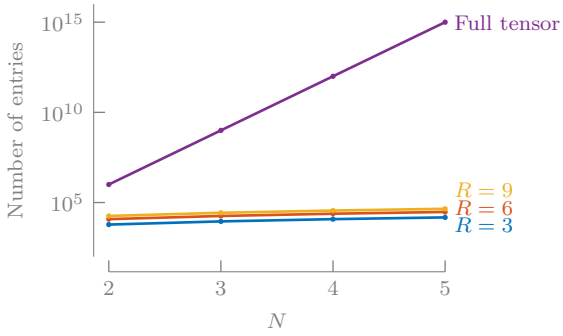


Figure 9.4: *The exponential dependence of the tensor on its order is broken by using a TTM, for which the number of coefficients grows only linearly with N. In the plot, it can be seen that the number of entries in the tensor increases exponentially in the order. In contrast, for the TTMs with ranks $R = 3, 6, 9$, for example, the number of coefficients necessary to represent the data with good accuracy depends only linearly on the order of the tensor.*

are inefficient to be processed. However, as discussed before, the intention is to avoid the construction of tensors like this at all cost. This procedure is conducted here for the sole purpose of analysing the tensor model. When coupling with a phase-field simulation or for other applications a much more efficient approach can be used to access this data point by point as explained in subsection 9.5.3.

The range normalized percent error AE% formulation

$$\mathrm{AE}\% = \mathrm{median}\left(\frac{|\mathcal{G}_S - \mathcal{G}_H|}{\max \mathcal{G}_S - \min \mathcal{G}_S}\right) \cdot 100\%,$$

is used to estimate the error between the tensors constructed with the TTM and the tensors collected with TC-Toolbox. The choice of a range normalized error is necessary due to the fact that the interval of the data goes from a large negative number to a large positive with a few values approaching zero, which results in high peaks when plotting the standard relative error even for a small deviation between TC-Toolbox and TTM values.

The range normalized percent error $AE\%$ is plotted in Figure 9.5 for the Gibbs free energy, potentials and derivatives. This plot provides a valuable first guess on the optimal rank. TTMs with $R = 3$ and 4 have the highest error, which is close to 1% of the range of the thermodynamic data values. However, for $R = 5, 6, 7$, we observe a great improvement in the accuracy, while this improvement levels off for higher ranks.

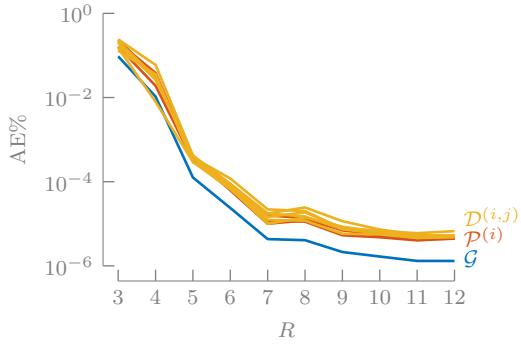


Figure 9.5: The largest improvements in accuracy are seen for TTMs with $R = 5, 6, 7$; For higher ranks, i.e., $R > 7$, no noticeable improvements are observed. *This figure contains the median range normalized absolute error plot in percentage, comparing data calculated with all TTMs as a function of rank* $(R = 3, \ldots, 12)$ *with data calculated with TC-Toolbox.*

9.6.2 Validation for one-dimensional simulations

One-dimensional simulations of the six alloy compositions present in Table 9.1 are conducted; each alloy has its initial condition with random noise around the concentrations of the components created five times for verifying the reproducibility, thus leading to 30 systems to be simulated. The Gibbs free energy of the liquid is provided by two different coupling schemes. First, the phase-field method is coupled with the CALPHAD model expression and the resulting data is taken as reference. Followed by simulations which are coupled with TTMs with $R = 3, \ldots, 10$, leading to a total of 330 simulations.

The performance analysis of the coupling approaches is conducted by comparing the composition profile using a percent relative error

$$RE\% = \left(\frac{\left| \Omega - \hat{\Omega}_{r=1,\ldots,R} \right|}{|\Omega|} \right) \cdot 100\%$$

in which Ω refers to a molar fraction profile over the whole simulation domain taken as reference and $\hat{\Omega}_r$ the profile from simulations conducted with the use of TTMs with different ranks. The comparison is always taken between concentration profiles of the same elements on each alloy and with the same initial condition.

The results are shown in Figure 9.6. The first row presents the cumulative probability function (CPF) of the error distribution plotted for TTMs with ranks $R = 3, \ldots, 10$, it shows that ranks above 6 have an overall better accuracy. Adding to that, the TTM with $R = 5$ also exhibits a good performance. On the second row the quantiles 0.9 and 0.99 applied to the error distribution are plotted as a function of simulation time. The 0.99-quantile, for instance,

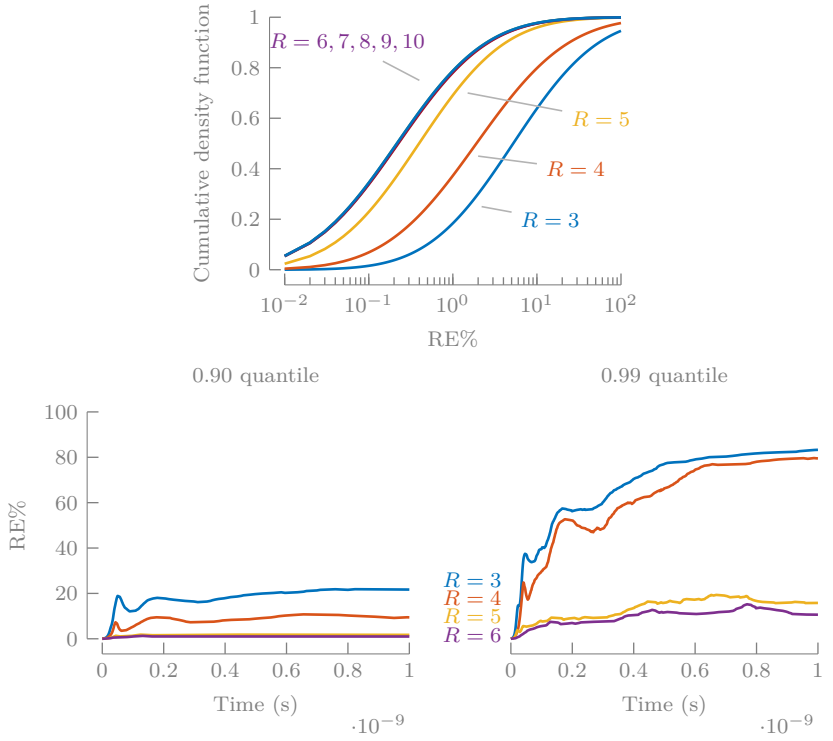


Figure 9.6: No improvements to the accuracy of the simulations are observed for $R > 6$. *This is also seen during the simulation according to the quantiles. (Top) Cumulative probability for TTMs with* $R = 3, \ldots, 10$. *(Bottom) 0.90 and 0.99-quantiles in function of the simulation time step.*

plots the value of relative error where 90% of the data falls below and with the added information about the simulation time it is possible to observe that the error during the simulations with TTM with $R = 5$ and above remains almost constant. Remember that a quantile 0.99 represents only the 1% of the data which has the highest error which approaching the maximum error.

Based on the CPF, it is evident that no reasonable improvements to the accuracy of the simulations are seen using ranks higher than six. It is important to note that the objective is not only to find the rank with the best accuracy, since in that case a higher rank could be selected. An optimal rank can be chosen depending on the desired accuracy for a particular application considering that both phase-field simulations and TTM estimation benefit speed-wise from a smaller rank. Another important remark is the fact that simulations with rank above ten are not considered as the analysis in Figure 9.5 already provided the idea that ranks greater than ten do not

improve the accuracy of the simulations considerably. The results of the one-dimensional simulations confirm that there are no obvious improvements for $R > 6$.

The implementation of the coupling between TC-Toolbox and the phase-field simulations is conducted only for 1D simulations due to its low speed, which takes in average 430 min. Since a quadratic increase of this time is expected, two-dimensional simulations are not feasible with this implementation. Note that no optimization techniques as the one discussed in [33], [182] are used here since it is not our intention to test the speed of the coupling with thermodynamics software. As 2D simulations are considered not to be feasible with this scheme, the further validation of the TTM models of different ranks for two-dimensional systems is done with respect to the simulations with full evaluation of the CALPHAD expressions.

9.6.3 Validation for two-dimensional simulations

Two-dimensional simulations of the six alloys compositions are conducted by coupling with both the CALPHAD full expression as explained in subsection 9.5.1 and with the TTMs with $R = 3, \ldots, 10$ obtained with the decomposition presented in section 4. For each alloy one single composition profile is created, meaning that in total 48 simulations are conducted with a two-dimensional domain.

A selection of microstructures from the 2D simulations can been seen in Table 9.3. The CALPHAD model expression directly coupled in the phase-field model is shown in the first column. All the other columns represent microstructures produced with the coupling with TTMs of different ranks. The reader is referred to supplementary material for a collection with all the two-dimensional microstructures produced in this study.

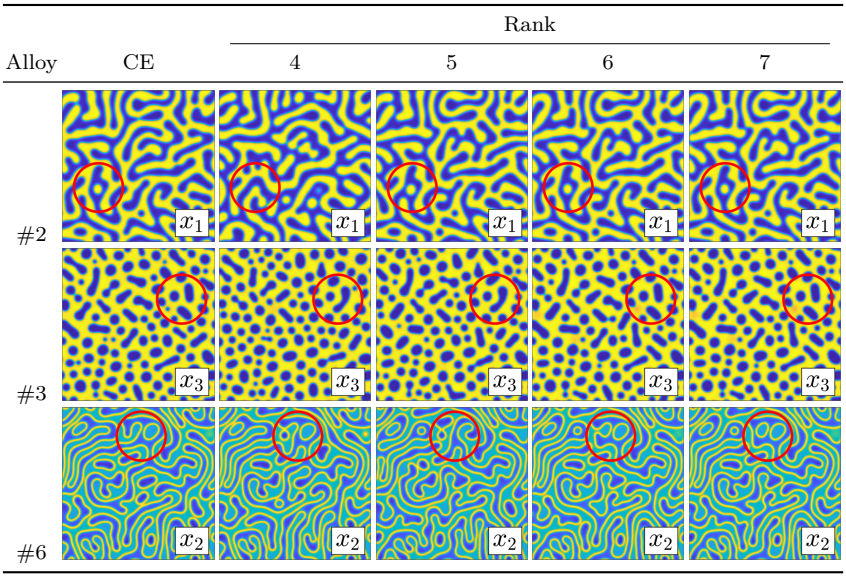
The inspection of the microstructure reinforce the information obtained from the analysis of the 1D simulations. A few features that are highlighted by red circles are present in simulations with both the CALPHAD expression and those with TTMs $R \geq 6$, but absent in microstructures obtained with TTM with $R = 3, 4$. In general the TTM with $R = 5$ also presents accurate results, but as mentioned before it is only above $R = 6$ that no further improvements are observed.

Data collected from 2D simulations are displayed in Figure 9.7, with the cumulative probability function on the first row and quantiles as a function of the simulation time on the second row.

The behavior of the 2D simulations is in general similar to the 1D case. It is possible to observe an increase in accuracy as the rank of the TTM increases, but in both the 1D and the 2D simulations, no further improvements are observed for ranks six.

Additionally, volume fraction measurements are conducted on the microstructures of each alloy and the results are presented in Figure 9.8. Sim-

Table 9.3: Certain features present in the 2D microstructures resulting from simulation with the full CALPHAD expression can also be seen in microstructures from simulations with TTMs $R \geq 6$. *The table contains a selection of microstructures obtained from the 2D simulations for visual comparison. The first column shows the microstructures obtained with direct coupling with the CALPHAD expression and the following columns the coupling through thermodynamic tensor models for different ranks. Each row represents one alloy and one component as described by the label.*



ulations coupled with the CALPHAD full expression CE are plotted along simulations using TTMs with ranks $R = 3, \ldots, 6$ as a function of simulation time. All measurements are taken for the liquid phase one, which equilibrium concentrations are included in the supplementary material. The gray lines represent the equilibrium volume fraction of the phases obtained from Thermo-Calc.

The measurements of volume fraction show that the use of a TTM $R = 6$ do not show evident differences from simulations using the full CALPHAD expression. For the system considered in this current chapter a TTM $R6$ gives the highest accuracy, since no improvements are observed when using a higher rank. However, depending on the application a TTM with $R = 5$ or $R = 4$ for instance, can already be sufficient, as can be seen in Figure 9.8.

9.7 Conclusion

The coupling between phase-field models and data calculated with the CALPHAD approach via tensor decomposition has a great potential. As seen

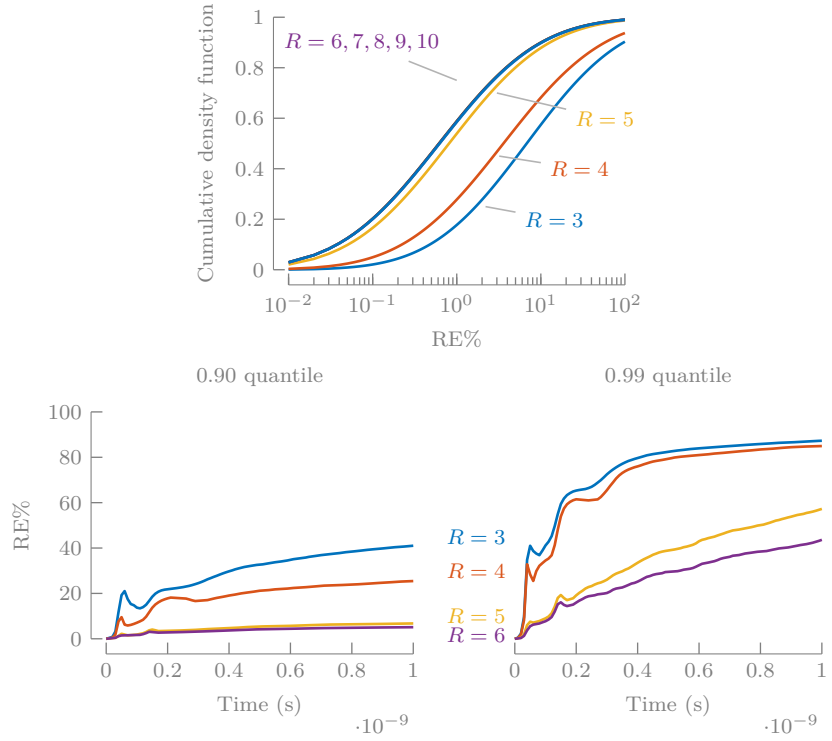


Figure 9.7: Analysis of the 2D simulation confirm the results obtained from 1D, with no improvements to the accuracy being observed by using TTMs with $R > 6$. *Two-dimensional analysis by comparing simulation conduced via directly coupling with the CALPHAD model expression with simulations coupled with TTMs* $R = 3, \ldots, 10$. *(Top) Cumulative probability. (Bottom) 0.90 and 0.99-quantiles in function of the simulation time step.*

through the analysis of the tensor data and the phase-field simulations, a TTM with $R = 6$ can be used to accurately couple phase-field models with data calculated from CALPHAD models. Simulations conducted with rank 6 exhibit a median error of less than 1% for the considered system, which is acceptable and expected for simulations. No further improvements to the accuracy of the simulation is observed by using a higher rank TTM.

An initial step conducted over all ranks provided a first guess about how each individual TTM would behave. This is confirmed by the one-dimensional simulations and reinforced later with two-dimensional simulations. Many advantages support the use of the thermodynamic tensor decomposition model:

- Compact representation of a large set of data with high accuracy and with a number of coefficients that increases linearly in size when new

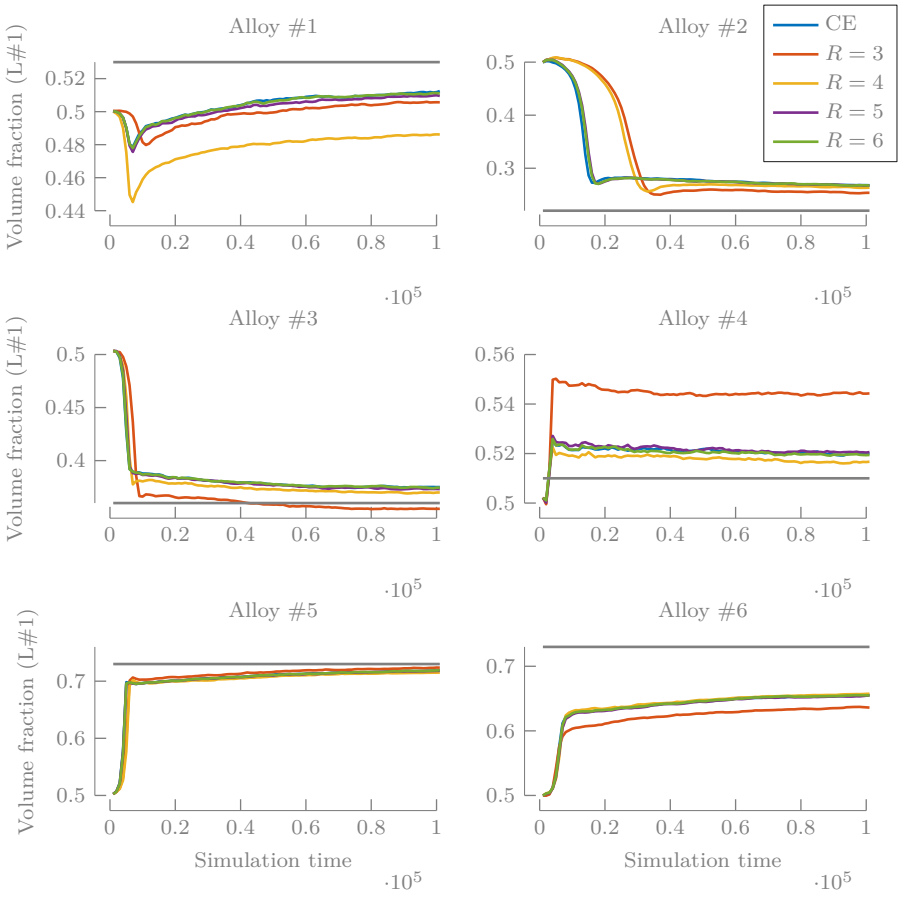


Figure 9.8: The measurements of volume fraction shown that depending on the application a rank $R = 5$ or even $R = 4$ can lead to accurate results. *Volume fraction measurements for the liquid phases, plotted for simulation coupled with the full expression CE and with simulations coupled with data from the TTMs with ranks* $R = 3, \ldots, 6$. *The gray line represent the equilibrium volume fraction obtained from calculation using TC-Toolbox at the same condition of pressure and temperature.*

components are added, which is a fundamental aspect for higher-order multicomponent alloys.

- Straight-forward collection and decomposition procedure, allowing for an easy iteration between different systems.

- Use of thermodynamic information contained in encrypted commercial databases in phase-field multicomponent simulations.

- The possibility of taking temperature as a system dimension and thus having the thermodynamic information described for a large temperature range.

Additional study of this novel coupling technique between phase-field models and CALPHAD expressions using tensor decomposition techniques needs to be conducted. A similar evaluation applied for solid phases described with a CALPHAD sublattice model is necessary to allow a more general use of this coupling scheme. The evaluation of the TTMs can be optimized further with a more detailed study of the decomposition parameters as a function of the number of components.

Face recognition as a Kronecker product equation

10

ABSTRACT | Various parameters influence face recognition such as expression, pose, and illumination. In contrast to matrices, tensors can be used to naturally accommodate for the different modes of variation. The multilinear singular value decomposition (MLSVD) then allows one to describe each mode with a factor matrix and the interaction between the modes with a coefficient tensor. In this chapter, we show that each image in the tensor satisfying an MLSVD model can be expressed as a structured linear system called a Kronecker product equation (KPE). By solving a similar KPE for a new image, we can extract a feature vector that allows us to recognize the person with high performance. Additionally, more robust results can be obtained by using multiple images of the same person under different conditions, leading to a coupled KPE. Finally, our method can be used to update the database with an unknown person using only a few images instead of an image for each combination of conditions. We illustrate our method for the extended Yale Face Database B, achieving better performance than conventional methods such as Eigenfaces and other tensor-based techniques.

This chapter is based on M. Boussé, N. Vervliet, O. Debals, and L. De Lathauwer, "Face recognition as a Kronecker product equation", in *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec. 2017, pp. 276–280. The figures have been updated for consistency.

10.1 Introduction

Face recognition is an important problem in computer vision with many applications within domains such as information security, surveillance, and biometric identification [315]. Although many recognition systems use matrix-based methods, face recognition is inherently a multidimensional problem due to variations in facial expression, pose, illumination conditions, etc. [295]. Linear algebra is of limited use because it only captures a single variation using a mode of a matrix. For example, the well-known Eigenfaces method stacks vectorized images in the second mode, obtaining a matrix with modes pixels $\times$ persons [285]. Although some methods have tried to accommodate for different conditions [61], [115], the multidimensional structure remains a challenging problem for matrix-based methods.

Recently, tensor tools have gained increasing popularity in signal processing and machine learning applications [65], [243]. Tensors are higher-order generalizations of vectors (first order) and matrices (second order). The higher-order structure allows one to explicitly accommodate for the multidimensional structure of facial images: each mode of a tensor can represent a single variation of the image [295]. For example, a set of (vectorized) images of several persons under different illumination conditions can be represented by a third-order tensor with modes pixels $\times$ illuminations $\times$ persons. An important tensor tool is the multilinear singular value decomposition (MLSVD) of a higher-order tensor which is a generalization of the well-known singular value decomposition (SVD) [78]. The MLSVD allows one to approximately represent the tensor by a set of factor matrices that are each related to a single mode and a core tensor that explains the interaction between the different modes. This type of representation is also used in TensorFaces, enabling improved accuracy in face recognition in comparison with conventional techniques such as Eigenfaces [296]. Several other tensor-based methods have been proposed; see [137], [140].

In this chapter, we explain that tensor-based face recognition using the MLSVD model can be expressed as a Kronecker product equation (KPE). A KPE is a linear system of equations with a Kronecker product constrained solution for which the authors have developed a generic framework in [40]. We show that by solving a KPE for a new unlabeled image, one can obtain a feature vector that enables better recognition rates than conventional methods. In practice, the robustness can be improved by coupling multiple images of the same person under different conditions, leading to a set of KPEs that are coupled. Additionally, our method allows one to add a new unknown person using only a few images instead of an image for each combination of conditions. We illustrate our method for the extended Yale Face Database B which contains facial images of multiple persons under different illuminations [114]. Our KPE-based method achieves higher performance than conventional Eigenfaces and the tensor-based approach in [296].

We conclude this section with an overview of the notation and basic definitions. We also define the MLSVD and KPEs. Next, we reformulate face recognition as a KPE in section 10.2. We apply our approach to a real-life dataset in section 10.3.

10.1.1 Notations and basic definitions

We denote vectors, matrices, and tensors by bold lowercase, e.g., $\mathbf{a}$, bold uppercase, e.g., $\mathbf{A}$, and calligraphic letters, e.g., $\mathcal{A}$, respectively. A natural extension of the rows and columns of a matrix, is a mode-n vector of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, defined by fixing every index except the nth, e.g., $\mathbf{a}_{i_1 \cdots i_{n-1} : i_{n+1} \cdots i_N}$. A mode-$n$ unfolding of $\mathcal{A}$ is a matrix $\mathbf{A}_{(n)}$ with the mode-n vectors as its columns (following the ordering convention in [170]). The vectorization of $\mathcal{A}$, denoted as $\text{vec}(\mathcal{A})$, maps each element $a_{i_1 i_2 \cdots i_N}$ onto $\text{vec}(\mathcal{A})_j$ with $j = 1 + \sum_{k=1}^{N} (i_k - 1) J_k$ and $J_k = \prod_{m=1}^{k-1} I_m$. We indicate the nth element in a sequence by a superscript between parentheses, e.g., $\{\mathbf{A}^{(n)}\}_{n=1}^{N}$.

The outer and Kronecker product are denoted by $\otimes$ and $\otimes$, respectively. The mode-n product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{B} \in \mathbb{R}^{J_n \times I_n}$ is a tensor $\mathcal{A} \cdot_n \mathbf{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots I_N}$ and is defined element-wise as $(\mathcal{A} \cdot_n \mathbf{B})_{i_1 \cdots i_{n-1} j_n i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 i_2 \cdots i_N} b_{j_n i_n}$. Hence, each mode-$n$ vector of the tensor $\mathcal{A}$ is multiplied with the matrix $\mathbf{B}$, i.e., $(\mathcal{A} \cdot_n \mathbf{B})_{(n)} = \mathbf{B}\mathbf{A}_{(n)}$.

10.1.2 Multilinear singular value decomposition

The multilinear singular value decomposition (MLSVD) of a higher-order tensor is a generalization of the singular value decomposition (SVD) of a matrix [65], [78], [243]. The MLSVD writes a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ as the product

$$\mathcal{A} = \mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot_2 \mathbf{U}^{(2)} \cdots \cdot_n \mathbf{U}^{(N)},$$

in which $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times I_n}$ is a unitary matrix, $n = 1, \ldots, N$, and the core tensor $\mathcal{S} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is ordered and all-orthogonal; see [78] for more details. The mode-n rank of an Nth-order tensor is equal to the rank of the mode-n unfolding. The multilinear rank of the tensor is equal to the tuple of mode-n rank values. The MLSVD is related to the low-multilinear rank approximation (LMLRA) and the Tucker decomposition (TD); see, e.g., [78], [170], [304]. The decomposition has been used successfully in applications such as compression and dimensionality reduction [83], [170].

10.1.3 (Coupled) Kronecker product equations

A KPE is a linear system of equations with a Kronecker product constrained solution. Here, we limit ourselves to problems with the following simple

Kronecker product structure:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{with} \quad \mathbf{x} = \mathbf{v} \otimes \mathbf{u}, \tag{10.1}$$

in which $\mathbf{A} \in \mathbb{R}^{M \times K}$, $\mathbf{x} \in \mathbb{R}^{K}$, and $\mathbf{b} \in \mathbb{R}^{M}$. The solution $\mathbf{x}$ can be expressed as a Kronecker product $\mathbf{v} \otimes \mathbf{u}$ with $\mathbf{u} \in \mathbb{R}^{I}$ and $\mathbf{v} \in \mathbb{R}^{J}$ such that $K = IJ$. As a matter of fact, a KPE is a simple case of a linear system with a solution that can be represented as a matrix or tensor decomposition [40]. Expression (10.1) can be solved by first solving the system without structure and subsequently decomposing a matricized version of the solution. This approach works well if $\mathbf{A}$ has full column rank, but, in contrast to the methods in [40], fails when $\mathbf{A}$ is rank deficient or in the underdetermined case. The methods in [40] compute the least-squares (LS) solution of (10.1).

A coupled KPE (cKPE) is a set of KPEs that have a common coefficient vector. We limit ourselves to cKPEs of the form:

$$\mathbf{A}(\mathbf{v} \otimes \mathbf{u}^{(q)}) = \mathbf{b}^{(q)} \quad \text{for} \quad q = 1, \ldots, Q,$$

with $\mathbf{A} \in \mathbb{R}^{M \times K}$, $\mathbf{v} \in \mathbb{R}^{I}$, $\mathbf{u}^{(q)} \in \mathbb{R}^{J}$, and $\mathbf{b}^{(q)} \in \mathbb{R}^{M}$ such that $K = IJ$. By defining $\mathbf{U} \in \mathbb{R}^{J \times Q}$ with $\mathbf{u}_q = \mathbf{u}^{(q)}$ and $\mathbf{B} \in \mathbb{R}^{M \times Q}$ with $\mathbf{b}_q = \mathbf{b}^{(q)}$, we obtain

$$\mathbf{A}(\mathbf{v} \otimes \mathbf{U}) = \mathbf{B}.$$

10.2 Face recognition using KPEs

10.2.1 Tensorization and MLSVD model

Higher-order tensors can explicitly accommodate for the multidimensional nature of facial images by representing each variation by a mode of the tensor [295], [296]. Although our method can be used for any combination of variations, we illustrate the strategy for the following particular case. Consider a set of facial images of J persons taken under I different illumination conditions. Each image is represented by a matrix of size $M_x \times M_y$ with M_x and M_y pixels in the x- and y-direction, respectively. All *vectorized* images of length $M = M_x M_y$ are stacked into a third-order tensor $\mathcal{D} \in \mathbb{R}^{M \times I \times J}$ with modes pixels (px) $\times$ illuminations (i) $\times$ persons (p).

Next, we compute a truncated MLSVD of the tensor $\mathcal{D}$:

$$\mathcal{D} \approx \mathcal{S} \cdot_1 \mathbf{U}_{\text{px}} \cdot_2 \mathbf{U}_{\text{i}} \cdot_3 \mathbf{U}_{\text{p}}, \tag{10.2}$$

in which $\mathbf{U}_{\text{px}} \in \mathbb{R}^{M \times P}$, $\mathbf{U}_{\text{i}} \in \mathbb{R}^{I \times R}$, and $\mathbf{U}_{\text{p}} \in \mathbb{R}^{J \times L}$ form an orthonormal basis for the pixel, illumination, and person mode, respectively. The interaction between the different modes is expressed by the core tensor $\mathcal{S} \in \mathbb{R}^{P \times R \times L}$.

Each row of $\mathbf{U}_{\mathrm{p}}$, denoted by $\mathbf{c}_p^{\mathrm{T}}$, can be interpreted as the coefficients for person p and each row of $\mathbf{U}_{\mathrm{i}}$, denoted by $\mathbf{c}_i^{\mathrm{T}}$, can be interpreted as the coefficients for illumination i.

10.2.2 Kronecker product equation

Each mode-1 fiber of $\mathcal{D}$ in (10.2) corresponds to an image and can be modeled by a KPE as follows. Consider a vectorized image $\mathbf{d} \in \mathbb{R}^M$ for a *particular* person p and illumination i:

$$\mathbf{d} = (\mathcal{S} \cdot_1 \mathbf{U}_{\mathrm{px}}) \cdot_2 \mathbf{c}_i^{\mathrm{T}} \cdot_3 \mathbf{c}_p^{\mathrm{T}}, \tag{10.3}$$

$$\mathbf{d} = \mathbf{U}_{\mathrm{px}} \mathbf{S}_{(1)} (\mathbf{c}_p \otimes \mathbf{c}_i). \tag{10.4}$$

Expression (10.4) is the mode-1 unfolding of (10.3) and is a KPE: each $\mathbf{d}$ is a linear combination of the columns of $\mathbf{U}_{\mathrm{px}} \mathbf{S}_{(1)}$ with Kronecker product constrained coefficients $(\mathbf{c}_p \otimes \mathbf{c}_i)$.

Consider a set of facial images of the same person under Q different illuminations, leading to a set of coupled KPEs that share the coefficient vector in the illumination mode:

$$\mathbf{d}^{(q)} = \mathbf{U}_{\mathrm{px}} \mathbf{S}_{(1)} (\mathbf{c}_p \otimes \mathbf{c}_i^{(q)}) \quad \text{for} \quad q = 1, \ldots, Q. \tag{10.5}$$

By stacking all vectorized images $\mathbf{d}^{(q)}$ and illumination coefficient vectors $\mathbf{c}_i^{(q)}$ into a matrix $\mathbf{D} \in \mathbb{R}^{M \times Q}$ and $\mathbf{C}_i \in \mathbb{R}^{R \times Q}$, respectively, we obtain

$$\mathbf{D} = \mathbf{U}_{\mathrm{px}} \mathbf{S}_{(1)} (\mathbf{c}_p \otimes \mathbf{C}_i).$$

10.2.3 Face recognition

We explain how to recognize a person in a (set of) facial image(s) under a new illumination condition using (c)KPEs. First, we construct a tensor $\mathcal{D}$ by stacking a set of facial images of different persons under different illuminations in the way explained in subsection 10.2.1. Second, we compute the truncated MLSVD of $\mathcal{D}$, obtaining factor matrices $\mathbf{U}_{\mathrm{px}}$, $\mathbf{U}_{\mathrm{i}}$, and $\mathbf{U}_{\mathrm{p}}$, and core tensor $\mathcal{S}$. Every (vectorized) image of $\mathcal{D}$ can then be expressed as a KPE as explained in subsection 10.2.2. Next, consider a new, unlabeled facial image $\mathbf{d}^{(\mathrm{new})}$ of a known person. In order to recognize the person in the image, we solve the following KPE using the algorithms from [40]:

$$\mathbf{d}^{(\mathrm{new})} = \mathbf{U}_{\mathrm{px}} \mathbf{S}_{(1)} \left(\mathbf{c}_p^{(\mathrm{new})} \otimes \mathbf{c}_i^{(\mathrm{new})} \right), \tag{10.6}$$

obtaining estimates $\hat{\mathbf{c}}_p^{(\mathrm{new})}$ and $\hat{\mathbf{c}}_i^{(\mathrm{new})}$ for the coefficient vectors. We compare $\hat{\mathbf{c}}_p^{(\mathrm{new})}$ with the rows of $\mathbf{U}_{\mathrm{p}}$ using the Frobenius norm of the difference (after fixing scaling and sign invariance). One can then recognize the person in

the image by assigning the label corresponding to the closest match. In other words, the estimated coefficient vector for the person mode $\hat{\mathbf{c}}_p^{(\text{new})}$ acts as a feature vector and $\mathbf{U}_\text{p}$ acts as a database. More robust results can be obtained by using images under multiple illumination conditions and coupling the KPEs as in (10.5).

In contrast to our method, the tensor-based approach in [296] solves (10.6) by fixing the illumination coefficients to a particular illumination. More specifically, the approach solves (10.6) by taking $\mathbf{c}_i^{(\text{new})}$ equal to a row of $\mathbf{U}_\text{i}$, reducing (10.6) to a linear system of equations for *each* illumination condition. Every estimate is then compared with $\mathbf{U}_\text{p}$ in a similar way as explained above. This approach is especially tedious when considering many modes because a linear system has to be solved for every possible combination. Our method, on the other hand, computes the LS solution of (10.6) by explicitly exploiting the Kronecker product structure of the coefficients.

10.3 Numerical experiments

We illustrate our KPE-based method for the extended Yale Face Database B[1] which contains cropped facial images of $J = 37$ persons under 64 illumination conditions. Some of the illuminations are missing for several persons and are therefore removed entirely from the dataset, obtaining $I = 57$ conditions. Each image of size 51×58 pixels is vectorized into a vector of length $M = 2958$. Hence, the resulting tensor $\mathcal{D}$ has size $2958 \times 37 \times 57$.

We use a nonlinear LS algorithm with random initialization in order to solve (c)KPEs [40]. All computations are done with Tensorlab [305]. We compute the MLSVD with a randomized algorithm called `mlsvd_rsi` [306]. We use $R = 15$, $L = J$, and $P = 1000 \ll M$ which we determined via cross validation. We project the given image onto the column space of $\mathbf{U}_\text{px}$ in order to reduce computation time. In order to accommodate for scaling and sign invariance, the rows of $\mathbf{U}_\text{p}$ and the estimated coefficient vectors are normalized as $\text{sign}(c_1)\frac{\mathbf{c}}{\|\mathbf{c}\|}$. As explained in subsection 10.2.3, the normalized rows of $\mathbf{U}_\text{p}$ act as database, denoted by $\mathbf{U}_\text{db}$, and the normalized coefficient vector for the person mode acts as a feature vector.

10.3.1 Proof-of-concept

Although the facial image in Figure 10.1 is almost completely dark, our method correctly recognizes the person in the image. In this case, we constructed the MLSVD model in (10.2) using all facial images of all persons under every illumination condition. Hence, the coefficient vectors $\mathbf{c}_p$ and $\mathbf{c}_i$ are perfectly reconstructed and a correct match is found. The reconstructed

[1] The extended Yale Face Database B can be downloaded from `http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html` (visited March 14, 2018.

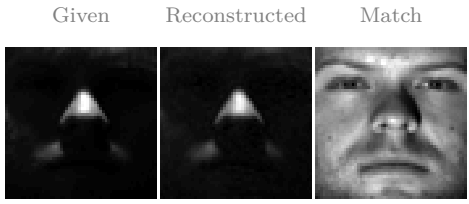


Figure 10.1: Classification of a person that is included in the dataset. *Note that we can identify the person even though the picture is almost completely dark.*

Table 10.1: By reformulating face recognition as a Kronecker product equation, higher performance (%) can be obtained in comparison to conventional techniques such as Eigenfaces as well as the tensor-based approach in [296]. *Lower recognition time (s) is achieved in comparison to the method of [296].*

	Eigenfaces [285]	Vasilescu [296]	KPE
Accuracy	93.3	93.5	**95.7**
Precision	90.6	94.4	**96.6**
Recall	88.4	90.9	**95.8**
Time of PCA/MLSVD	**2.97**	3.29	3.29
Time of recognition	**0.004**	0.135	0.097

image in Figure 10.1 can be obtained by recomputing the vectorized image using the estimated coefficient vectors.

10.3.2 Performance

Our method obtains higher performance than conventional techniques as we effectively exploit the multilinear structure of facial images by reformulating the recognition task as a KPE. Although our method is slower than the matrix-based method, it is slightly faster than the tensor-based method from [296] for this dataset. In Table 10.1 we report the median performance and time across 50 trials for our method, Eigenfaces, and the tensor-based method from [296]. In particular, we report the accuracy as well as the precision and recall using macro-averaging as explained in [254]. In each trial, 75% of the illuminations conditions are selected randomly as training set and 25% as test set for each person. For Eigenfaces, we unfold the data tensor and apply principal component analysis (PCA): we have $\mathbf{D}_{(1)} = \mathbf{B}\mathbf{C}^{\mathrm{T}}$ with $\mathbf{B} \in \mathbb{R}^{M \times T}$ and $\mathbf{C} \in \mathbb{R}^{IJ \times T}$ using $T = J = 37$.

10.3.3 Improving performance through coupling

More robust recognition can be achieved by using multiple images of the same person under different illuminations. In Table 10.2 we report the me-

Table 10.2: Higher performance (%) can be achieved by using multiple images under different illuminations. *Our cKPE-based method outperforms Eigenfaces using majority voting.*

	Eigenfaces [285]			cKPE-based method		
# illuminations	1	2	3	1	2	3
Accuracy	92.7	93.3	96.3	**95.8**	**97.1**	**97.3**
Precision	89.8	91.2	97.9	**97.0**	**99.3**	**99.9**
Recall	87.7	87.8	97.5	**96.2**	**99.2**	**99.9**

dian performance across 15 trials when using $Q = \{1, 2, 3\}$ randomly chosen illuminations. We compare our cKPE-based method to Eigenfaces using majority voting. For the latter, we assign the label of the person with the lowest index in the case of a tie. We use the same experiment settings as in subsection 10.3.2 and solve a cKPE with Q randomly chosen illuminations which we repeat 25 times for each trial and each person. Clearly, higher accuracy can be achieved by using multiple images for both approaches. Our cKPE-based method achieves higher accuracy than Eigenfaces.

10.3.4 Updating the database with a new person

Given an MLSVD model, the KPE-based method allows one to update the database $\mathbf{U}_{\text{db}}$ with a new person using *only a few* images instead of an image for each illumination. For example, consider an MLSVD model as in (10.2) which we have constructed using the facial images of all but one person under every illumination. The retained person is initially not included in the database $\mathbf{U}_{\text{db}}$, but can be recognized as follows. By solving (10.6) for a particular image, we obtain a feature vector $\hat{\mathbf{c}}_p^{(\text{new})}$ which we can add as a new row to the extended database $\mathbf{U}_{\text{db}}$ (with known label). In order to recognize the person in a new image under a different illumination, we can proceed as before, i.e., we solve a KPE and compare the obtained feature vector with the extended database. This strategy works well if the given image can be well approximated by the original MLSVD model. In practice, one can improve the recognition by extending the database using multiple illuminations and solving a cKPE to obtain a new row for the database.

We illustrate the approach for the person depicted in Figure 10.2 (left). In this example, we choose a neutral illumination to update the database. The current MLSVD captures the new person reasonably well as can be seen from the reconstructed image on the right. The person is correctly recognized in a new image with a different illumination as illustrated in Figure 10.3.

By using multiple illuminations to update the database, one can again further improve the performance. In Table 10.3 we report the performance when updating the database with the first, third, or 31st person in the extended Yale Face Database B using all other persons to construct the model.

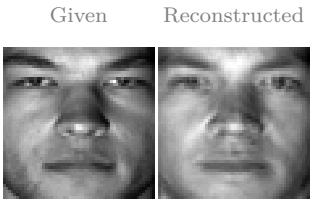


Figure 10.2: The MLSVD model captures the new person reasonably well.

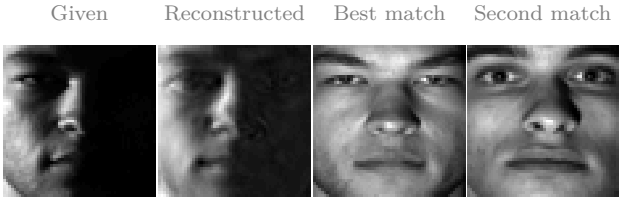


Figure 10.3: Although we update the database with a new person using only one illumination condition, the KPE-based method recognizes that person in a new image under a different illumination condition.

In other words, the data is divided into a training set of 36 persons and a test set of 1 person. In this experiment, we use $P = 1000 \ll M$. When using one, two, or three images, we use illumination setting 1, $\{1, 10\}$, and $\{1, 10, 55\}$, respectively. Illumination 1, 10, and 55 correspond to the neutral illumination and a left and right illumination of the face, respectively. Clearly, the accuracy improves by updating the model using multiple illuminations as can be seen for several persons in Table 10.3. Also, one can see that the median performance over all persons in the dataset improves by using additional illuminations.

Table 10.3: When updating the database with a new person, our method can achieve higher accuracy (%) by fusing multiple images under different illumination conditions instead of using only one image of the new person.

Person	One illumination	Two illuminations	Three illuminations
16	51.8	56.4	**59.3**
25	64.3	72.7	**75.9**
28	58.9	63.6	**70.4**
All	61.8	66.2	**68.1**

10.4 Conclusion

In this chapter, we proposed a new tensor-based technique for face recognition that exploits the multidimensional nature of a collection of facial images under different conditions such as illumination, pose, and expression. First, we construct a tensor by stacking the images along several modes that each relate to a variation in the image. Our method models the obtained tensor by a multilinear SVD, describing each of the modes with a factor matrix and the interaction between the modes with a tensor. By reformulating the recognition task as the computation of a KPE, we can explicitly exploit the multilinear structure of the problem, obtaining a feature vector that enables higher performance than conventional methods. We illustrated our method for the extended Yale Face Database B, obtaining better performance than Eigenfaces and another tensor-based technique. Our method performs well when using only a single image and the performance can be improved further by coupling a few images with different illuminations. Remarkably, our method also allows one to update the database with a new person using only a few images instead of an image for each combination of conditions. In future work, one can probably improve the performance by using neural networks or support vector machines in combination with KPE-computed feature vectors instead of using Euclidian distance-based comparisons. Additionally, one can take into account the nonnegative nature of the data.

11 Conclusion

Extracting useful information such as sources or patterns from a compressed measurement of large-scale datasets using tensor decompositions is the main goal of this thesis. This way, the complexity of the algorithms can be reduced significantly both during the computation of the decomposition and when using this decomposition in further analyses. By exploiting incompleteness, we show that the so-called curse of dimensionality can be alleviated or broken, depending on the decomposition at hand (Chapter 3). In this thesis, we have developed algorithms that deal with sparsely sampled or incomplete tensors with optional linear constraints (Chapter 4), that randomly sample blocks (Chapter 5), that exploit the structure or efficient representation of a tensor (Chapter 6) and that update a tensor when new slices arrive frequently (Chapter 7). Moreover, we have defined a more general problem involving tensors that are given implicitly as the solution of a linear system (Chapter 8). Most of these algorithms are optimization-based and fit in a larger data fusion framework which is discussed in Chapter 2.

We have illustrated these algorithms for a variety of large-scale applications such as modeling the melting temperature of an alloy ($\mathcal{O}\left(10^{18}\right)$ entries), classification of hazardous gasses (12.5 GB) or blind source separation of exponential polynomials (approximately 1 TB). Moreover, we have discussed two applications in more detail. In Chapter 9, we have validated the use of a low-rank CPD to model the Gibbs free energy of an alloy and have shown that the curse of dimensionality can be broken, and in Chapter 10, we have shown how face recognition can be cast as a (coupled) LS-CPD problem which leads to an increased recognition rate.

In the remainder of this chapter, we discuss and compare the algorithms derived in this thesis in section 11.1, we give an overview of all contributions in section 11.2, and outline a perspective for future research in section 11.3.

11.1 Comparison of methods

In this thesis five types of techniques have been derived to decompose large-scale tensors into rank-1 terms. As each of these techniques has its own use cases, advantages and disadvantages, a brief overview of the most important properties is given here. The following points are taken into account: type of tensor or situation in which the method is useful, accuracy compared to the accuracy that would be achieved by an algorithm using the full tensor, speed in terms of computational (per-iteration) complexity, scale of the problems (i.e., which factors present possible limitations), and whether the method can be used easily in a general data fusion framework implementing constrained and coupled matrix/tensor decompositions. When appropriate, additional remarks are made.

Chapter 4: incomplete tensors and linear constraints

Two types of algorithms have been derived: the CPDI algorithm for incomplete tensors and the CPDLI algorithm for incomplete tensors with additional linear constraints, i.e., each factor can be written as $\mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)}$ with $\mathbf{B}^{(n)}$ a known $I \times D$ matrix[1]. The latter algorithm has a data-dependent (DD) and a data-independent (DI) variant.

- *Type and situation.* The tensor has missing entries, e.g., due to measurement errors, unfeasible situations or preprocessing, or very few entries are sampled to reduce the cost. In the case additional (linear) constraints such as smoothness, polynomial structure or dictionary constraints are present, the CPDLI variant is used. If the dominant mode-n subspaces are available, the CPDLI algorithm can be used as well.

- *Accuracy.* The number of sampled tensor entries M that affect each variable, influences the accuracy. In practice, the results from the central limit theorem can be applied: the standard deviation of the estimator, and therefore the accuracy, decreases inversely proportional to $\sqrt{M}$, i.e., as $1/\sqrt{M}$. Depending on the fraction I/D, the number of variables can be much lower for the CPDLI variant, leading to an improved accuracy for the same M.

- *Complexity.* The complexity is mainly governed by the product of the rank and the number of sampled entries. As the inexact CG solver is the bottleneck, good preconditioning strategies such as the proposed statistical preconditioner are important. The CPDLI DI algorithm can only be used for low orders and a small number of coefficients, given

[1] We take $I_n = I$ and $D_n = D$ for all n for simplicity.

the exponential dependency on the order. The DD variant is a factor D more expensive than CPDI.

- *Scale.* The limiting factor is the computation and storage of the Jacobian matrices in RAM, which scales linearly in the number of known entries and the rank. Note that the tensor dimensions indirectly influence the number of samples required and that many samples may be required if one dimension is large.
- *General use.* CPDI and CPDLI can be integrated trivially in a larger data fusion framework.
- *Other notes.* Techniques such as (cross) validation can be used easily by subdividing the data into a training and a test set, resulting in two incomplete tensors. Tensorlab 4.0 provides additional support for validation metrics.

Chapter 5: randomized block sampling

The RBS algorithm samples a random block every iteration to compute an update and discards this block afterwards.

- *Types and situation.* The only requirement is that a dense or sparse tensor can be sampled, which excludes incomplete tensors. An explicit construction of the tensor is not required, as long as the blocks can be generated when required. Especially when data accesses are expensive, RBS can be beneficial as only a fraction of the data accesses are required.
- *Accuracy.* While the accuracy would be limited by the block size used (following the same reasoning as for CPDI), a properly chosen step restriction schedule can compensate this drop in accuracy almost completely, as long as the block size is sufficiently high, e.g., when most dimensions are larger than the chosen rank.
- *Complexity.* The complexity is determined only by the block size and the rank. The same matrix-free Gramian approximation as for full tensors can be used. Typically the number of required iterations increases if the block size decreases.
- *Scale.* In theory, the only limiting factor is the block size and the rank as the blocks should fit in RAM. In practice, generating a sample block can be the bottleneck due to random disk accesses or expensive data generation routines.
- *General use.* Integrating RBS in a data fusion framework is not trivial in general, as specialized sampling routines are required (in the

case of coupling) or the locality principle may not be valid for some constraints, e.g., symmetry, polynomial constraints, or column-wise normalization. Element-wise constraints, e.g., nonnegativity, are an exception (see Chapter 2).

- *Other notes.* Parameter tuning can be required to find suitable step restriction schedules.

Chapter 6: structured tensor framework

The structured tensor framework allows a tensor that is given in an efficient representation, i.e., with fewer parameters than the number of entries, to be decomposed. We show that only four simple representation-dependent operations are required. This framework is also defined for the LL1 decomposition, the LMLRA and the most general BTD.

- *Type and situation.* A large variety of tensors can be used: sparse or factored tensors, large-scale tensors that are compressed first, e.g., using a (randomized) MLSVD or a TT approximation, or tensors that are the result of implicit tensorization, e.g., Hankelization, Löwnerization or outer product structures.

- *Accuracy.* As noise, model errors or regularization terms limit the achievable accuracy in many applications, no additional loss of accuracy is expected. If the exact decomposition exists and is required, a refinement step using the explicitly constructed tensor may be required.

- *Complexity.* The complexity depends linearly on the rank and on the number of parameters in the representation, rather than the number of entries in the tensor. Matrix-free Gramian implementations or Gramian-vector products can again be used.

- *Scale.* The limiting factor is the number of parameters in the efficient representation. In the case compression is used as a first step, this compression is likely to be the bottleneck, although, in practice, this cost can often be amortized over multiple runs with different parameters.

- *General use.* The structured tensor framework can be integrated trivially into a larger data fusion framework.

- *Other notes.* Adding new efficient representations to the structured tensor framework only requires the implementation of the norm, inner product and matricized tensor times Khatri–Rao and/or Kronecker product.

Chapter 7: updating and tracking

The updating methods allow a tensor to be processed one slice at a time. Using windowing strategies, nonstationary CPDs can be tracked.

- *Type and situation.* Updating methods are useful when data arrives frequently and at a fast rate, when a CPD needs to be tracked, or for large-scale tensors that are handled subtensor by subtensor.
- *Accuracy.* The accuracy loss is negligible for large-scale tensors. For tracking purposes, the definition of accuracy is difficult and the error depends on the type and length of window that is used.
- *Complexity.* The complexity is governed mainly by the rank and the dimensions of the new slices. In some cases, factor matrices representing the data that has been processed already may become larger than the new data slices if no windowing techniques are used. Matrix-free Gramian implementations or Gramian-vector products can be used.
- *Scale.* The limiting factor is the storage of the factor matrices and the new slices, which is often far lower than the amount of storage required for the complete tensor.
- *General use.* Updating can be included in a general data fusion framework, albeit not trivially, as a solver that includes updating support is required since the variables may change every update. Also, the additional overhead introduced by a general framework can reduce the processing rate for new slices considerably. Column-wise constraints on the factor to be updated may be difficult to fulfill.

Chapter 8: implicitly defined tensors

The LS-CPD methods decompose a tensor that can be computed as the (reshaped) solution $\mathbf{x}$ of linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

- *Type and situation.* Multilinear systems of equations and tensors that are only available (implicitly) as the solution of a linear system.
- *Accuracy.* The accuracy cannot be compared easily as the LS-CPD algorithms solve a broader problem. Compared to the naive approach, i.e., solving the system first and then computing a tensor decomposition, only the LS-CPD approach is able to recover the decomposition if the system does not have full column rank (under some conditions).
- *Complexity.* The complexity depends on the number of equations and the structure of the matrix $\mathbf{A}$. If $\mathbf{A}$ is structured, exploiting this structure when computing contractions is important to reduce the (otherwise high) complexity.

- *Scale.* The matrix $\mathbf{A}$ is the limiting factor in most practical cases as it has as many columns as the number of entries in the tensor.
- *General use.* An LS-CPD can be integrated trivially in a general data fusion framework.
- *Other notes.* While being more expensive than tensor decompositions, the LS-CPD is far more general allowing new classes of problems to be solved.

11.2 Overview of contributions

For each chapter, a brief overview of the contributions is given. A number of chapters rely on results obtained in collaboration with other team members and departments. For these chapters, we clearly indicate the contributions of each collaborator.

Chapter 2

- *Broadly accessible overview of tensor-based optimization.* We give a clear tutorial on the implementation of optimization-based algorithms for tensors, building upon an inexact Gauss–Newton framework presented in part in [260], [262]. We focus on large-scale implementations and preconditioning, and explain how certain classes of constraints and coupling can be integrated elegantly in the framework.
- *Extension to other divergences.* To accommodate other types of statistical assumptions on the data, other divergences such as Kullback–Leibler or Itakura–Saito are often more appropriate. We show that the gradient and Gramian can be computed using simple modifications of the standard algorithm, by making similar assumptions as for the Gauss–Newton method. The structure can be exploited in a large-scale setting, and efficient routines such as MTKRPROD can be reused.
- *Inclusion of box constraints.* We show that bound or box constraints can be implemented using an active set method with minimal changes to the gradient and Gramian of the unconstrained problem. An important advantage of this method is that zero variables can still be updated, e.g., in the case of nonnegative tensor factorizations, while variables cannot escape from zero values when other types of nonnegativity constraints are used.
- *Overview of uniqueness results for coupled decompositions.* A brief overview with pointers to uniqueness results for coupled matrix and/or tensor decompositions is given.

- *Overview of state-of-the-art algorithms for large-scale tensors.* Relevant alternative techniques to compute a CPD of a large-scale tensor are discussed.

Chapter 3

In this chapter, we partially report results originating from the master's thesis by O. Debals and the doctoral candidate under daily supervision of L. Sorber. The materials science results have been obtained during a collaboration as a part of an IWT feasibility study together with InsPyro NV, a KU Leuven spinoff.

- *Introduction of scientific computing concepts.* We introduce scientific computing concepts such as tensor trains, the hierarchical Tucker decomposition and pseudoskeleton approximations into the signal processing community and illustrate how these concepts can be used for multidimensional harmonic retrieval problems.

- *Breaking the curse of dimensionality via tensor decompositions.* We show that the curse of dimensionality can be alleviated (in the case of an LMLRA) or even broken (in the case of a CPD or a TT) when this decomposition is used instead of a tensor for further analysis and processing. To alleviate or break this curse during the computation step as well, we give an overview of optimization-based techniques using very few sampled entries (for the CPD) and adaptive sampling techniques based on cross approximation or the pseudoskeleton approximation (for the LMLRA and TT).

- *Modeling melting temperature of a multicomponent alloy.* We show how a CPD can be used to model the melting temperature of an alloy with ten compounds. As measuring and storing all $\mathcal{O}\left(10^{18}\right)$ entries of this ninth-order tensor is unfeasible, we show that an accurate low-rank CPD can be constructed using only 100 000 samples. Computing the 4 500 variables in the CPD requires a couple of minutes on a laptop. The data has been provided by InsPyro NV.

Chapter 4

Parts of the results for incomplete tensors without constraints (CPDI) originate from work performed during the master's thesis of O. Debals and the doctoral candidate. The materials science results were obtained during a collaboration with Yuan Yuan and N. Moelans.

- *Development of a CPD algorithm for incomplete tensors.* A GN type algorithm with an exact formulation for Gramian of the Jacobian is

derived, and a new preconditioner based on the distribution of the known entries is constructed. We illustrate the superiority of this new algorithm compared to state-of-the-art methods for the interesting and common cases, i.e., when very few entries are known or when the missing entries are structured. We give experimental evidence that only few samples per variable are required, hence that the curse of dimensionality can effectively be broken.

- *Data-dependent and data-independent algorithms for incomplete tensors and linear constraints.* In large-scale applications one often imposes smoothness constraints or uses dictionaries, which can be modeled as linear constraints on the factor matrices. We show that this constraint can be exploited elegantly to significantly reduce the computational cost. The data-independent version uses a projection of the data, allowing large speedups when relatively many entries are given and the number of variables is low.

- *Materials science application.* To model the smooth Gibbs free energy data from a multicomponent alloy in its liquid phase, a CPD with polynomial constraints on the factor matrices is computed. We show that 200 samples are sufficient to accurately represent this data using a rank-6 CPD in which each factor vector is a polynomial with a maximal degree of four; hence, only 90 variables are used. The materials science data and background were contributed by Yuan Yuan and N. Moelans.

Chapter 5

- *Randomized block sampling algorithm for CPD.* Combining concepts from randomized block coordinate descent and stochastic gradient descent, a randomized algorithm for the computation of the CPD is derived. By sampling subtensors, or blocks, instead of some entries, efficient full tensor kernels for the gradient and the Gramian can be reused and fast convergence can be achieved under certain conditions. Variants based on ALS and GN are derived, and illustrated for the decomposition of tensors up to 8 TB. The number of data accesses is reduced significantly, which is important in the case data accesses are expensive, and in some cases, the algorithm converges before all entries are sampled.

- *Step restriction schedule.* By explicitly controlling the step size for ALS algorithms or the trust region radius in the case of GN, a search-then-converge step restriction schedule is implemented. We show that this restriction can be seen as a variance reduction technique and that the error on the recovered factor matrices can be reduced significantly.

This way the error can be reduced to almost the same level as if the full tensor is used instead of small blocks.

- *Stopping criterion based on Cramér–Rao bound.* An efficient way to estimate a lower bound on the variance of the variables based on the CRB is presented. This estimate can be used as an alternative stopping criterion as the function value and the step length can be unreliable in the case of RBS.

- *Classification of hazardous gasses.* We show that a CPD can be used to extract useful features to classify hazardous gasses based on time series from a sensor array. Given the large size (12.5 GB) and the limited amount of RAM (16 GB), loading the tensor takes more than ten minutes and full tensor algorithms run out of memory immediately, while the RBS method decomposes the tensor in a few minutes using only a negligible amount of RAM.

Chapter 6

In this chapter, we report the results of a collaboration with O. Debals.

- *Structured tensor framework.* By rewriting the cost function and the gradient, the typical subtraction of a tensor and the model in the least squares formulation can be avoided and few core operations are exposed: norms, inner products and matricized tensor times Khatri–Rao or Kronecker products. We show that the Gramian required for the GN algorithm is unaffected. In these core operations the structure of a tensor, e.g., sparsity, LMLRA structure or Hankel structure, can be exploited. This way, the complexity of ALS, quasi-Newton and GN algorithms becomes a function of the number of parameters required to represent the tensor efficiently, instead of a function of the number of entries.

- *Structure exploiting core operations.* We show for five types of efficient representations (CPD, LMLRA, tensor train, Hankelization and Löwnerization) how the structure can be exploited to achieve the desired complexity reduction. This is illustrated for all core operations such that both the CPD and the BTD can be computed. The implementations for implicit Hankelization and implicit Löwnerizations were contributed by O. Debals.

- *Concept of implicit tensorization.* As tensorization techniques map lower-order vector and matrix data to higher-order tensors, much redundant information is created and applications are often limited in size due to memory limits. By implementing core operations that exploit the structure created by tensorization, tensor decompositions can

be computed with a vector or matrix complexity and the explicit construction of the tensor is avoided. We illustrate this implicit tensorization technique for blind source separation of exponential polynomials with up to 500 000 samples for which the explicitly Hankelized tensor would require 953 GB of memory.

- *Accuracy analysis.* The effect of exploiting the structure on the accuracy is limited in the case noise or model errors are present or when regularization is added. However, in the case a highly accurate solution is required (and can be achieved), additional iterations using the full tensors may be required as the accuracy may be limited to half the machine precision. This is an effect that is almost always overlooked in other papers using similar techniques.

- *Constraints and coupling.* We show that imposing constraints and coupling multiple datasets in a data fusion setting is trivial as long as the objective function and gradient make use of the structure exploiting core operations. We illustrate for the nonnegative tensor factorization that MLSVD-based compression, which is a common preprocessing step for the unconstrained CPD, can be combined with nonnegativity constraints. For high-order tensors, a TT approximation of the tensor proves effective as compression step, which is illustrated for GB-size tensors.

Chapter 7

In this chapter, we report the results of a collaboration with M. Vandecappelle. The doctoral candidate has contributed the structured tensor approach (Chapter 6) and the efficient implementation, which has been generalized by M. Vandecappelle.

- *Memory efficient updating algorithm.* A GN based algorithm for updating a CPD is derived. Instead of storing the tensor of slices that have been processed, only the factor matrices of its CPD are kept. When a new slice arrives, the factor matrices and the slice are used as an efficient representation for the complete tensor. Our algorithm exploits this representation to achieve a low complexity in terms of memory and computations. The complexity results were contributed by M. Vandecappelle.

- *Tracking dynamic systems.* For nonstationary systems that output low-rank tensor data, the CPD can change over time. Using rectangular and/or exponential windows, we take the 'age' of data into account. By assuming the factors change relatively slowly, a fast algebraic initialization is derived. Thanks to the combination of this algebraic

initialization with the structure exploiting algorithm and windowing strategies, new slices can arrive and be processed at significantly higher rates. This result was contributed mainly by M. Vandecappelle.

Chapter 8

In this chapter, we report the results of a collaboration with M. Boussé, I. Domanov and O. Debals. The doctoral candidate has contributed significantly in the formulation of the LS-CPD concept, the optimization-based algorithms and the face recognition application.

- *Interpretation of systems with Kronecker product constraints as multilinear systems of equations and implicitly given tensor decompositions.* We show that the system $\mathbf{Ax} = \mathbf{b}$ in which $\mathbf{x}$ can be written as a sum of R Kronecker products can be interpreted as a multilinear system of equations if $R = 1$. By formulating the problem as a tensor decomposition of an implicitly given tensor, instead of solving the system and then decomposing the reshaped solution, systems without full column rank can be solved which means the number of required equations for a unique solution is reduced significantly.

- *Generic uniqueness conditions.* Given a system $\mathbf{Ax} = \mathbf{b}$ with a random matrix $\mathbf{A}$, and $\mathbf{x}$ a vectorized CPD with random factor matrices, we prove that this CPD can be recovered uniquely with probability one if the number of equations (number of rows in $\mathbf{A}$) is strictly greater than the number of free variables in the CPD. This result was contributed by I. Domanov.

- *Algebraic algorithm for the recovery of rank-1 tensors.* If $\mathbf{x}$ is a rank-1 tensor, we present an algorithm to compute the factor matrices of the corresponding CPD, even in the case $\mathbf{A}$ does not have full column rank provided some specific conditions hold. This result was contributed by I. Domanov.

- *Optimization-based algorithm.* A GN type algorithm is derived to compute the factorization of $\mathcal{X} = \text{unvec}(\mathbf{x})$ while solving the system of equations simultaneously, i.e., without solving for $\mathbf{x}$ first. This can be seen as a tensor that is given implicitly as a solution of a system of equations. While the algorithm is derived for general $\mathbf{A}$, the complexity can be reduced significantly by exploiting the structure in $\mathbf{A}$. The generalization to $R > 1$ was contributed by M. Boussé.

- *Face recognition using TensorFaces.* We show that face recognition using a tensor constructed with face images from different persons under varying illumination conditions can be formulated as an LS-CPD and

that using the presented LS-CPD approach improves the recognition rate. See also Chapter 10.

- *Algorithm for computing tensors with given multilinear singular values.* By writing the orthogonality constraints and the constraints imposing the multilinear singular values as linear system with a Kronecker product constraint on the solution, the LS-CPD algorithm can be used to compute a tensor with prescribed multilinear singular values. Given the size of $\mathbf{A}$, which is approximately $J^2/2 \times J^2$ with J the number of entries in the tensor, the exploitation of the sparsity of $\mathbf{A}$ is crucial. The problem was formulated by I. Domanov; the structure exploiting implementation was developed by the doctoral candidate.

- *Constant modulus algorithm.* We show that blind system identification with constant modulus assumptions on the source signals can be seen as an LS-CPD problem and that the generic LS-CPD algorithm without exploiting structure already is as effective as, and almost as efficient as state-of-the-art methods. This application was contributed by O. Debals.

Chapter 9

In this chapter, we report the results of a collaboration with Y. Coutinho and N. Moelans. The doctoral candidate was responsible for the efficient conversion of the CALPHAD data to the tensor model.

- *Coupled decomposition with polynomial constraints.* To exploit the smoothness of the Gibbs free energy and its first and second-order partial derivatives w.r.t. the fractions of each compound, we propose a linearly constrained CPD as a model for each of the ten tensors. By taking the derivatives into account, the datasets can be coupled, hence reducing the number of coefficient matrices to be estimated to three rather than 30, while improving accuracy. To compute this coupled decomposition the CPDLI algorithm (Chapter 4) is generalized. The required data was provided by Y. Coutinho and N. Moelans.

- *Breaking the curse of dimensionality using incompleteness.* We illustrate that for multicomponent alloys, the number of required data points increases exponentially. By exploiting the smoothness and the low-rank structure of the CPD, only few entries are required, which allows the curse of dimensionality to be broken.

- *Validation through simulation.* We show that the low-rank models which are computed using only the potential data (first-order derivatives of the Gibbs free energy), represent the ten datasets accurately. We verify that the chosen rank and degree of the polynomials, which

are determined using validation data, coincide with the parameters chosen when using the data in the simulation of the spinodal decomposition of an Ag–Cu–Ni–Sn alloy. This validation step was contributed by Y. Coutinho and N. Moelans.

Chapter 10

In this chapter, we report the results of a collaboration with M. Boussé, and O. Debals. The doctoral candidate has contributed the framework for face recognition with Kronecker product equations[2] (KPE).

- *Improved face recognition via multilinear systems.* We show that face recognition using a tensor with vectorized face images of different persons under varying illuminations, i.e., the TensorFaces approach [296], can be solved more efficiently and accurately using a KPE after using multilinear compression of the tensor. Moreover, the formulation allows new persons to be added to the database using a single image under one arbitrarily chosen illumination setting. The performance is illustrated using the Extended Yale B dataset. Preliminary results have been obtained during an intense collaboration between O. Debals and the doctoral candidate, and have been extended during a collaboration with M. Boussé.
- *Extension to coupled KPEs.* Instead of using a single image to recognize a person, multiple images with varying illumination conditions can be used to improve robustness. We show that this leads to coupled KPEs, which can be computed via a generalization of the LS-CPD algorithm from Chapter 8. This extension was contributed by M. Boussé.

Appendix A

In this appendix, the new features contributed to Tensorlab 3.0 by O. Debals and the doctoral candidate are discussed.

- *Large-scale tensor decompositions.* New algorithms for incomplete tensors (Chapter 3), based on randomized block sampling (Chapter 5), or exploiting efficient representations (Chapter 6) are made available. Moreover, a new algorithm for computing the MLSVD using randomized SVDs and subspace iteration is presented.
- *More efficient handling of coupling and symmetry.* To accelerate data fusion problems, a new solver for (possibly symmetric) coupled matrix/tensor factorizations is implemented. This new solver has a small-scale algorithm, reduces the number of computations and has a better preconditioner.

[2] A KPE is equivalent to an LS-CPD with $R = 1$.

- *(De)tensorization techniques.* New methods for (explicit and implicit) Hankelization, Löwnerization, decimation and segmentation have been created. To extract the underlying signals again, detensorization techniques are implemented. Apart from full tensors, these techniques accept implicitly-given tensors as well, such that the explicit creation of the full tensors is avoided. These techniques were contributed by O. Debals.

- *Improved domain specific language (DSL) for structured data fusion (SDF) problems.* More efficient solvers have been added to the SDF framework to speed up common computations. To make the DSL more user-friendly, the language has become more lenient and syntax errors are indicated using a new parser.

11.3 Perspectives

Over the last ten years, research on tensor decompositions and algorithms spurred a lot of interest resulting in new developments such as the block term decomposition, tensor trains and hierarchical Tucker, randomized tensor algorithms, distributed and parallel approaches, the use of inexact solvers for Gauss–Newton and so on. In this thesis, we discussed a number of methods to handle large-scale tensors. However, despite all this progress, a number of promising paths are still to be investigated and can result in a variety of new applications. In the remainder of this conclusion, we outline a few of these potential lines of research.

- *Global (in)equality constraints.* Combining information from various sources and applying prior knowledge are key ingredients in data fusion. In the case of tensor decompositions, constraints are often imposed on factors. However, in many cases, it would be more interesting to apply constraints on the global decomposition, e.g., to limit the angle between two rank-1 terms, or to impose nonnegativity on tensors without requiring every entry in a factor to be nonnegative. The synergies between these global constraints and optimization over manifolds are worth investigating.

- *Alternative statistical assumptions.* The least-squares formulation is ubiquitous in literature and often leads to elegant algorithms. Its inherent assumptions, namely that the residuals are normally distributed with the same variance, may not make sense for audio or count data or when some slices or more reliable than others, e.g., thanks to better sensors. In the latter case, introducing weights can be appropriate. In the former case, other divergences such as β-divergences can be used. While alternating algorithms have been proposed in statistics,

for nonnegative matrix factorizations or for count data, it is still an open problem how to exploit (approximate) second-order information and all available multilinear structure.

- *The general block term decomposition.* Although the BTD has been proposed ten years ago [77], its use is still limited compared to the CPD despite its enormous potential in signal processing applications. An impediment for the use of general BTDs is its computational difficulty. However, for special variants such as the decomposition in multilinear rank $(L_r, L_r, 1)$ new algorithms have become available [260], [305]. Progress can be made by developing better algorithms for increasingly difficult variants, e.g., for multilinear rank $(M_r, N_r, \cdot)$ terms, and by combining with results from optimization over manifolds.

- *Linear systems with tensor-constrained solutions, machine learning and updating.* The theory and algorithms for linear systems with CPD constrained solutions can obviously be generalized to other tensor decompositions. In scientific computing, for example, tensor trains are already investigated as constraints in combination with TT approximations on the matrix and the right-hand-side. In the case of the MLSVD, a challenge is to construct algorithms relying solely on numerical linear algebra tools. From the application point-of-view, a promising research direction is to look into multilinear alternatives to nonlinear machine learning techniques. Combined with tensor updating, powerful new methods can be derived.

- *Revisiting null space problems.* An important special case of linear systems with low (multilinear) rank solutions are the so-called null space problems in which the tensor to be decomposed is implicitly determined by the null space of a matrix. To avoid the high cost of computing this null space, which currently limits the study of this type of problems, a single step approach can again be used. Efficient algorithms can result in breakthroughs for computing roots of systems of polynomials or difficult tensor decompositions and for higher-dimensional system theory.

Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization

A

ABSTRACT | We give an overview of recent developments in numerical optimization-based computation of tensor decompositions that have led to the release of Tensorlab 3.0 in March 2016 (www.tensorlab.net). By careful exploitation of tensor product structure in methods such as quasi-Newton and nonlinear least squares, good convergence is combined with fast computation. A modular approach extends the computation to coupled factorizations and structured factors. In the case of large datasets, different compact representations (polyadic, Tucker, ...) may be obtained by stochastic optimization, randomization and compressed sensing, among others. Careful exploitation of the representation structure allows us to scale the algorithms for constrained/coupled factorizations to large problem sizes. The discussion is illustrated with application examples.

This chapter is based on N. Vervliet, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization", in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov. 2016, pp. 1733–1738. DOI: 10.1109/ACSSC.2016.7869679. The figures have been updated for consistency.

A.1 Introduction

Central to multilinear algebra are tensors, or multi-way arrays of numerical values, and their many types of decompositions such as the canonical polyadic decomposition (CPD), the block term decomposition (BTD) or the multilinear singular value decomposition (MLSVD). Similar to their matrix counterparts, these decompositions can be used to analyze data, compress data, make predictions and much more. The multilinear structures allow more complex relations to be modeled, as has been shown in countless applications not only in signal processing [65], [73], [243], but, among others, also in data analytics and machine learning [4], [193], [250].

Tensorlab [305] is a MATLAB toolbox with as main purpose to provide user-friendly access to a variety of state-of-the-art numerical algorithms and tools for tensor computations. In March 2016, the third version of Tensorlab has been released. This chapter gives a birds-eye overview of some new techniques that have been made available. The overview is by no means exhaustive: a full overview can be found at `www.tensorlab.net`. A number of demos illustrating good Tensorlab practice can be accessed at `www.tensorlab.net/demos`.

We continue this section by explaining the history and philosophy of Tensorlab and by fixing the notations. Appendix A.2 discusses the SDF framework from Tensorlab, while Appendix A.3 explains the concept of tensorization. Appendix A.4 introduces a new algorithm for coupled matrix/tensor factorizations in Tensorlab 3.0. Large-scale approaches are discussed in Appendix A.5, with a focus on compression, incompleteness, randomizations and efficient representations.

A.1.1 History and philosophy

The first version of Tensorlab provided state-of-the-art algorithms for the computation of CPDs, BTDs or low multilinear rank approximations (LMRLA) as well as a large number of convenience methods involving tensors. These algorithms are based on the complex optimization toolbox (COT) [258], [259], allowing decompositions of real and complex datasets and/or variables. In optimization problems, real-valued functions with complex arguments are often split into the real part and the imaginary part, and both problems are solved separately. In contrast, the complex Taylor series expansion can be used to generalize standard real-valued optimization algorithms for complex arguments and data, thereby exploiting inherent structure present in derivatives which would otherwise be ignored [8], [258]. COT leverages this structure and provides generalizations of many standard optimization algorithms.

The alternating least squares (ALS) algorithm is undoubtedly the most popular algorithm for tensor decompositions, mainly because of its simplic-

ity. While it effectively exploits multilinear structures and often provides good results quickly, it is numerically not very sophisticated and it has no proven convergence [260], [287]. In Tensorlab, the main focus lies on more advanced optimization algorithms such as nonlinear least squares (NLS) methods, thereby benefiting from the many good results in numerical optimization, including convergence guarantees. The number of iterations needed is often lower because of the quadratic convergence. The asymptotic cost per iteration of NLS can be reduced to the cost of ALS, although with some larger constants [260]. To achieve this low cost, the multilinear structure is exploited and a preconditioned iterative solver is used to determine the step direction. In particular, in NLS algorithms the system

$$\mathbf{H}\mathbf{p} = -\mathbf{g} \tag{A.1}$$

is solved in every iteration, in which $\mathbf{H}$ is the Gramian of the Jacobian and $\mathbf{g}$ is the gradient. As computing the pseudoinverse of $\mathbf{H}$ is too expensive, the conjugate gradients (CG) method is used. CG requires only the matrix-vector products $\mathbf{H}\mathbf{p}$ to iteratively solve (A.1). In many tensor decomposition algorithms the multilinear structure can be exploited when computing these products. To reduce the number of CG iterations needed, preconditioning is used, i.e., instead of (A.1) the system

$$\mathbf{M}^{-1}\mathbf{H}\mathbf{p} = -\mathbf{M}^{-1}\mathbf{g} \tag{A.2}$$

is solved, in which the preconditioner $\mathbf{M}$ is an easily invertible matrix chosen such that (A.2) is easier to solve. (More technically, the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered than those of $\mathbf{H}$.) For tensor problems, a block-Jacobi preconditioner, i.e., a block diagonal approximation to $\mathbf{H}$, is often an effective choice [260]. The combination of low per-iteration cost with quadratic convergence of NLS type methods leads to a fast algorithm. In practice, the algorithms also seem more robust for ill-conditioned problems [260].

Since its official launch in February 2013, Tensorlab has seen two more releases. In January 2014, Tensorlab 2.0 was revealed, including the structured data fusion (SDF) framework as its major feature. SDF allows structured and coupled decompositions of multiple full, sparse or incomplete matrices or tensors. This was inspired by the success of specific dedicated algorithms, each exploiting a particular type of constraint on the factor matrices. SDF allows the user to choose different decompositions, constraints and regularizations and combine these to their liking using SDF's own domain specific language [262]. By leveraging the chain rule for derivatives, parametric constraints can be handled easily: over 40 constraints are included, such as nonnegativity, Toeplitz, polynomial, Kronecker, Vandermonde and matrix multiplication. Different types of regularization can be used to model soft constraints as well.

The most recent release from March 2016, Tensorlab 3.0, introduces tensorization and structured tensors, extends and improves the SDF framework while making it more user-friendly, introduces a number of large-scale algorithms and a new algorithmic family, improves coupled matrix/tensor factorizations, and much more. In the following sections, we discuss a number of these new features in more detail.

A.1.2 Notation

An Nth order tensor $\mathcal{T}$ can be factorized in various ways. The (canonical) polyadic decomposition (CPD) writes the tensor as a (minimal) number of rank-1 terms, each of which is the outer product, denoted by $\otimes$, of N nonzero vectors $\mathbf{a}_r^{(n)}$:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} \stackrel{\text{def}}{=} \left[\!\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right],$$

in which the factor matrix $\mathbf{A}^{(n)}$ contains the vectors $\mathbf{a}_r^{(n)}$ as its columns. The higher-order SVD (HOSVD) or multilinear SVD (MLSVD) can be written as the mode-n tensor-matrix product $\cdot_n$ of a core tensor $\mathcal{S}$ and N factor matrices $\mathbf{U}^{(n)}$:

$$\mathcal{T} = \mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot \cdots \cdot_N \mathbf{U}^{(N)}.$$

The block term decomposition (BTD) writes a tensor as a sum of low-multilinear rank terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathcal{S}^{(r)} \cdot_1 \mathbf{U}^{(r,1)} \cdot \cdots \cdot_N \mathbf{U}^{(r,N)}.$$

A special variant of the BTD is the decomposition into a sum of multilinear rank-$(L_r, L_r, 1)$ terms (LL1):

$$\mathcal{T} = \sum_{r=1}^{R} (\mathbf{A}_r \mathbf{B}_r^{\mathrm{T}}) \otimes \mathbf{c}_r.$$

An overview of these decompositions is given in Figure A.1.

The mode-n unfolding of a tensor $\mathcal{T}$ is denoted by $\mathbf{T}_{(n)}$ and concatenates the mode-n vectors as columns in the matrix $\mathbf{T}_{(n)}$. The element-wise product or Hadamard product, the transpose and the Hermitian transpose are denoted by $*$, $\cdot^{\mathrm{T}}$ and $\cdot^{\mathrm{H}}$, respectively.

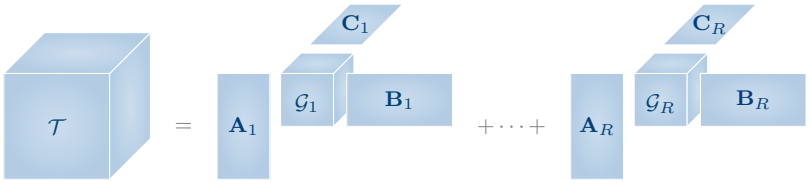


Figure A.1: Block term decomposition of a tensor $\mathcal{T}$ in terms with multilinear ranks (L_r, M_r, N_r). *If $R = 1$, an LMLRA is obtained. If $M_r = L_r$ and $N_r = 1$, thus if the rth core tensor has size $(L_r, L_r, 1)$, a BTD in multilinear rank-$(L_r, L_r, 1)$ terms is obtained. If $L_r = M_r = N_r = 1$, a CPD is obtained.*

A.2 Structured data fusion

Structured data fusion (SDF) is a framework for rapid prototyping of analysis and knowledge discovery in one or more multidimensional datasets in the form of tensors. Figure A.2 gives a schematic overview. These tensors can be complex, incomplete, sparse and/or structured. Each tensor is decomposed using one of the tensor decompositions that are included in Tensorlab. The factor matrices are possibly shared between the different datasets, meaning that the tensors are coupled. They can also be equal within a tensor decomposition, indicating the presence of symmetry. Furthermore, besides the choice of factorizations, regularization terms can be added as well, based on L_0, L_1 or L_2 norms. Regularization can be used to prevent overfitting but also to implement soft constraints.

In a lot of applications, prior knowledge is available on the factor matrices indicating some kind of structure such as orthogonality or nonnegativity. More than 40 structures are readily available in Tensorlab to constrain the factor matrices. Besides the administered structures, a user can design its own constraints as well by providing the mapping and its first-order derivative information. It is worthwhile to note that the constraints are implemented with parametric transformations of underlying optimization variables, rather than with penalty terms. For example, an orthogonal factor matrix of size $I \times R$ requires only $R(I-(R-1)/2)$ variables, while a Vandermonde matrix of size $I \times R$ requires only I generating variables. Hence, the solution space is reduced to a restricted search space, and the constraints are imposed exactly rather than only approximately. The chain rule is then internally used to cope with the composition of the tensor decomposition model and the various transformations/constraints, and to solve for the underlying variables.

The type of tensor decomposition, the coupling and the structure imposed on the factors can all be chosen independently of the solver and its options. Two different popular classes of algorithms are available to solve SDF problems in Tensorlab: quasi-Newton (qN) methods and nonlinear least squares (NLS) methods, implemented in `sdf_minf` and `sdf_nls`, respectively.

Within the qN methods, both limited memory BFGS (L-BFGS, subdivided in line search and trust region approaches) and nonlinear conjugate gradient (NCG) methods can be selected, while Gauss–Newton (CG-Steihaug and dogleg trust region approaches) and Levenberg–Marquardt algorithms are implemented within the NLS class.

In Tensorlab 3.0, the SDF framework has been updated in several respects. Two new solvers for symmetric and/or coupled CPDs are introduced (as discussed in Appendix A.4), as well as three new factorization types and various updated and new transformations. Besides a focus on content, there has also been a focus on user-friendliness. Using a new language parser (`sdf_check`), it is easier to formulate SDF models and to investigate them. It also helps finding errors in the model. Furthermore, the domain specific language has been made more lenient to allow more flexible model formulation, e.g., by automatically converting arrays to cells and adding braces, wherever necessary.

The handling of incomplete and sparse tensors has also improved from Tensorlab 3.0 on. Note that with the surge of big data applications in mind, the Tensorlab algorithms have a linear time complexity in the number of known/nonzero elements of the data tensor. The SDF features regarding incomplete tensors have shown its value in various applications before, such as in movie recommendation and user participation predictions [262] as well as in the design of alloys and in multidimensional harmonic retrieval [304]. This is further discussed in Appendix A.5.2.

A.3 Tensorization

Many powerful tensor tools have been developed throughout the years for analyzing multiway data. When no tensor data is available and only a matrix is given, tensor tools may still be used after first transforming the matrix data to tensor data. This transformation is called tensorization, and many different mappings are possible. The tensorization step is conceptually an important step by itself. Many results concerning tensorization have appeared in the literature in a disparate manner, but have not been discussed as such, e.g., [298].

After the tensorization step, one often computes a tensor decomposition. This is especially the case in blind signal separation, where the first tensorization step implements assumptions on the source signals while the second decomposition step realizes the actual separation of the sources [84].

Tensorlab 3.0 contains a number of tensorization techniques [84]. Hankelization (Hankel-based mapping) and Löwnerization (Löwner-based mapping) can be used when dealing with approximations by exponentials/sinusoids and rational functions, respectively. Segmentation and decimation are based on folding matrix data, which is, e.g., useful when dealing with large-scale data

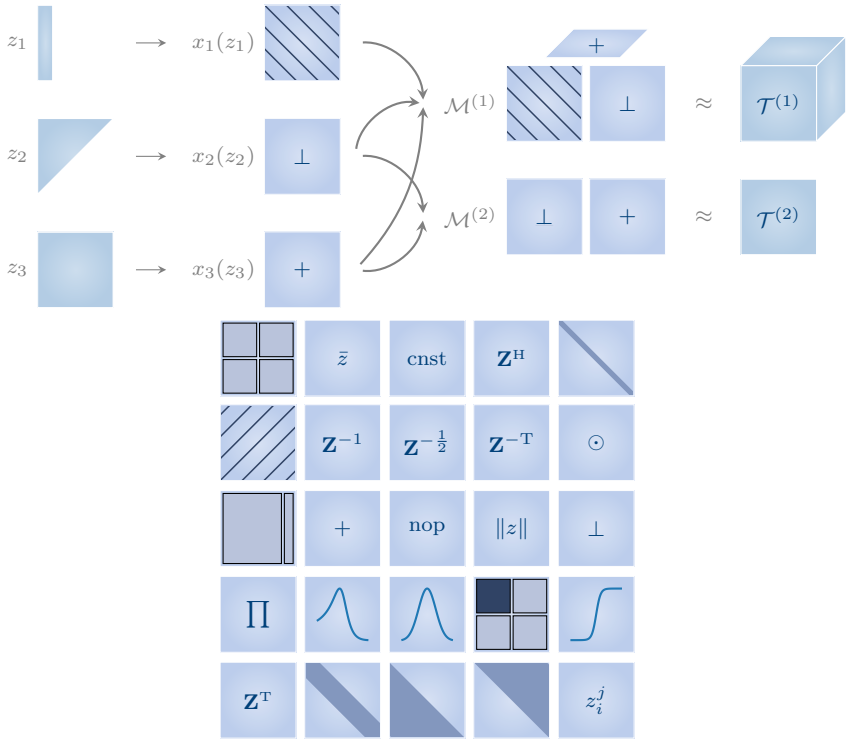


Figure A.2: A schematic of structured data fusion. *The vector z_1, upper triangular matrix z_2 (representing a sequence of Householder reflectors) and full matrix z_3 are transformed into a Toeplitz, orthogonal and nonnegative matrix, respectively. The resulting factors are then used to jointly factorize two coupled datasets $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$. More than 40 structures can be imposed on factor matrices. 25 examples are shown schematically at the bottom. (The top part is adapted from [262].)*

[36]. Also higher-order and lagged second-order statistics have been included.

Corresponding detensorization techniques have been included where possible. They can be useful, for example, to extract source estimates from the terms in the tensor decomposition. By providing a (noisy) Hankel matrix or tensor for example, the command `dehankelize` returns the averaged anti-diagonals or anti-diagonal slices, respectively.

Tensorization typically involves including redundant information in the higher-order tensor. The number of elements in the obtained tensor can grow quickly, in line with the curse of dimensionality which states that the number of elements in a tensor increases exponentially with the number of dimensions, and so do the computational and memory requirements. To cope with this curse, Tensorlab 3.0 can use efficient representations of the higher-order tensors resulting from the tensorization. The efficiency of these representations can then be exploited in the decomposition algorithms, as discussed in Section A.5.4.

A.4 Coupled matrix/tensor factorization

Joint decomposition of multiple datasets into rank-1 terms is a common problem in data analysis. Often symmetry constraints are used as well. Both coupling and symmetry, at the level of the data and the factorization, are easy to implement using SDF. In this section, we discuss how the new, specialized coupled and symmetric CPD (CCPD) solver improves convergence and reduces computation time compared to the standard SDF solvers by exploiting both constraints early.

The general SDF solvers `sdf_minf` and `sdf_nls` handle coupling and symmetry by first computing the Gramian of the Jacobian $\mathbf{H}$ and the gradient $\mathbf{g}$ as if no constraints were imposed (see Equation (A.1)). $\mathbf{G}$ and $\mathbf{g}$ are then contracted to $\mathbf{H}_c$ and $\mathbf{g}_c$ which, in this case, boils down to summing the proper blocks, as indicated in Figure A.3. The result is a smaller system which is cheaper to solve.

Figure A.3 shows that many blocks in $\mathbf{H}$ are repeated because of symmetry. The `ccpd_nls` function takes this into account directly: each unique block is multiplied by the number of occurrences instead of summing all blocks after computing them. In the case of the gradient, symmetry in the data $\mathcal{T}$ and the decomposition, e.g., $[\![\mathbf{A}, \mathbf{A}, \mathbf{B}]\!]$, has to be considered. In the example, the decomposition is symmetric in the first two modes as the factor matrices are identical. The gradients w.r.t. the first and second mode are only identical if $\mathcal{T}$ is symmetric in the first two modes as well. If this is the case, computing the gradient w.r.t. the second mode factor is unnecessary. Otherwise, there is no computational gain possible. Detecting symmetry is therefore an important task in the CCPD solvers.

For a regular CPD, a block-Jacobi preconditioner has shown to be effec-

Table A.1: Compared to SDF, CCPD requires less time and fewer iterations to converge when computing (A.3). *Increasing the number of CG iterations improves convergence and reduces computation time. All numbers are medians over 50 experiments. Both algorithms use the options* `TolX = eps` *and* `TolFun = eps^2`*, with* `eps` *the machine precision.*

	25 CG Iter.		75 CG Iter.	
	SDF	CCPD	SDF	CCPD
Time (s)	70.3	6.9	19.4	6.2
Iterations	170.0	45.5	39.5	29.5
Time/iteration (s)	0.40	0.15	0.47	0.20

tive and efficient to reduce the cost of solving (A.1) because of the Kronecker structure present in the blocks [260]. The `ccpd_nls` algorithm uses a similar preconditioner that exploits symmetry and coupling while keeping the Kronecker structure, in contrast to the nonpreconditioned `sdf_nls` algorithm.

To illustrate the performance gain of the new algorithm, consider the following coupled and symmetric problem:

$$\min_{\mathbf{M},\boldsymbol{\kappa}} \left|\left|\mathbf{C}^{(2)} - \mathbf{M}\mathbf{M}^{\mathrm{T}}\right|\right|^2 + \left|\left|\mathcal{C}^{(4)} - [\![\mathbf{M},\mathbf{M},\mathbf{M},\mathbf{M},\boldsymbol{\kappa}]\!]\right|\right|^2 \tag{A.3}$$

in which $\mathbf{C}^{(2)}$ and $\mathcal{C}^{(4)}$ are constructed using $\mathbf{M} \in \mathbb{R}^{50\times 25}$ and $\boldsymbol{\kappa} \in \mathbb{R}^{1\times 25}$ drawn from a normal distribution. In Table A.1 the SDF and the NLS algorithms are compared[1]. It is clear that exploiting all symmetry reduces the time per iteration. The block-Jacobi preconditioner used to solve (A.1) improves convergence considerably as can be seen from the reduced number of iterations. The combination of all improvements reduces the total computation time significantly.

A.5 Large-scale tensor decompositions

There exist many strategies for handling large-scale tensors: parallelization of operations, parallel decompositions, incompleteness, compression, exploitation of sparsity and so on. Here, we discuss four techniques readily available in Tensorlab: MLSVD computation using randomized matrix algebra, the use of incomplete tensors and randomized block sampling for polyadic decompositions, and the use of structured tensors.

[1] The timings for both algorithms benefited from a modified version of `mtkrprod` which is not yet released.

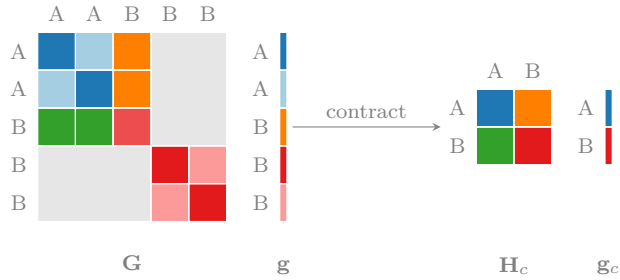


Figure A.3: For a joint decomposition of $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{B}]\!]$ and $\mathbf{M} = \mathbf{B}\mathbf{B}^{\mathrm{T}}$, the CCPD algorithm directly computes the contracted Gramian and gradient, while `sdf_nls` computes all blocks separately. *All blocks with the same color are identical and all blocks with the same hue are summed during contraction. (The gradient blocks are only identical if the tensor/matrix is symmetric.)*

A.5.1 Randomized compression

Using randomized matrix algebra, we derive a fast yet precise algorithm for computing an approximate multilinear singular value decomposition of a tensor $\mathcal{T}$. The standard way to compute an MLSVD uses the matrix SVD to compute the left singular vectors $\mathbf{U}^{(n)}$, $n = 1, 2, 3$, of the different unfoldings $\mathbf{T}_{(n)}$ of the tensor, and computes the core tensor $\mathcal{S}$ as $\mathcal{T} \cdot_1 \mathbf{U}^{(1)^{\mathrm{T}}} \cdot_2 \mathbf{U}^{(2)^{\mathrm{T}}} \cdot_3 \mathbf{U}^{(3)^{\mathrm{T}}}$ [78]. In very recent literature, one has replaced the SVD by a randomized variant from [135]. Here we present a variant that combines a sequential truncation strategy [294] with randomized SVDs and Q subspace iterations [135]. The full algorithm is described in Algorithm A.1.

As example, we create 400 random third-order tensors of size $I_1 \times I_2 \times I_3$ with I_n uniformly distributed in $[100; 400]$ and with multilinear ranks (R_1, R_2, R_3) with R_n distributed uniformly in $[10; 50]$, $n = 1, 2, 3$. The compression size is $(\tilde{R}_1, \tilde{R}_2, \tilde{R}_3)$ with $\tilde{R}_n$ distributed uniformly in $[10; 40]$. The oversampling parameter P is 5 and the number of subspace iterations Q is 2. The relative Frobenius norm error is maximally 4.2% higher in the case of the randomized algorithm `mlsvd_rsi` compared to the standard algorithm `mlsvd`, while the speedup is a factor 3 for small tensors and a factor 25 for larger tensors. If the used compression size is equal to or larger than the multilinear rank of the tensor, the mean relative errors are $1.3 \cdot 10^{-14}$ and $0.5 \cdot 10^{-14}$ for the standard and the randomized algorithm, respectively.

A.5.2 Incomplete tensors

Incomplete tensors occur for two main reasons. First, one can be unable to know some entries, for example, because a sensor breaks down, or because some entries correspond to physically impossible situations, e.g., negative

Algorithm A.1: Computation of MLSVD using randomization and subspace iteration. *(Implemented as* `mlsvd_rsi`*.)*

Input: Nth-order tensor $\mathcal{T}$ of size $I_1 \times \cdots \times I_N$, compression size $R_1 \times \cdots \times R_N$, oversampling parameter P and number of subspace iterations Q.
Output: Factor matrices $\mathbf{U}^{(n)}$, $n = 1, \ldots, N$ and core tensor $\mathcal{S}$ such that $\mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot \cdots \cdot_N \mathbf{U}^{(N)} \approx \mathcal{T}$.
Set size $s_n \leftarrow I_n$, $n = 1, \ldots, N$ and $\mathcal{Y} \leftarrow \mathcal{T}$
for $n = 1 \ldots, N$ **do**
    Let $\mathbf{\Omega}$ be a random matrix of size $\prod_{k \neq n} s_k \times R_n + P$
    $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}\mathbf{\Omega}$
    for $q = 1, \ldots, Q$ **do**
        $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}^{\mathrm{T}}\mathbf{Q}$
        $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}\mathbf{Q}$
    end for
    $\mathbf{USV}^{\mathrm{T}} \xleftarrow{SVD} \mathbf{Q}^{\mathrm{T}}\mathbf{Y}_{(n)}$
    $\mathbf{U}^{(n)} \leftarrow \mathbf{QU}(:, 1 : J_n)$ $s_n \leftarrow R_n + P$
    $\mathcal{Y} \leftarrow \text{reshape}(\mathbf{SV}^{\mathrm{T}}, s_1, \ldots, s_N)$
end for
$\mathcal{S} \leftarrow \mathcal{Y}(1 : R_1, \ldots, 1 : R_N)$

concentrations [304]. In the second case, all elements could be known, but computing or storing all entries is too costly, hence some elements are deliberately omitted. For example, for a rank-R CPD of an Nth order tensor $\mathcal{T}$ of size $I \times \cdots \times I$, the number of entries is I^N, while the number of variables is only NIR. Hence, the number of entries scales exponentially in the order, while the number of variables scales only linearly. This enables the use of very sparse sampling schemes [304].

Here, we restrict the discussion to the computation of a CPD of an incomplete tensor. Three main techniques can be found in literature [304]. First, unknown elements can be imputed, e.g., by replacing all unknown values with the mean value or with zero. Second, in an expectation-maximization scheme, the unknown values are imputed each iteration with the current best guess from the model. Third, the unknown elements can be ignored altogether. In this last approach, the objective function for a CPD becomes

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} ||\mathcal{W} * (\mathcal{T} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!])||_{\mathrm{F}}^2 , \tag{A.4}$$

in which $\mathcal{W}$ is a binary observation tensor. Various optimization schemes have been used to minimize objective (A.4) [6], [262], [279], [302].

Two NLS type algorithms are available in Tensorlab. The first technique scales the Gramian by the fraction of known values, but ignores the structure of the missing data [262]. While this approach is very fast, the result may not

be accurate in some cases. If the number of known entries is extremely small, this algorithm may fail [302]. The second technique uses the exact Gramian of the Jacobian, i.e, the structure of the missing data is exploited. Second-order convergence can be achieved, but each iteration is relatively expensive. This is often compensated for, however, as the number of iterations needed for convergence is reduced significantly. As shown in [302], leveraging the exact Gramian can sometimes be crucial in order to find a reasonable solution.

A.5.3 Randomized block sampling

A third technique involves full tensors which may not fit into memory entirely, or for which the computation cost per iteration would be excessive. In [300] a technique called Randomized Block Sampling (RBS) was presented to compute the CPD of large-scale tensors. This method combines block coordinate descent techniques with stochastic optimization as follows. Every iteration, a random subtensor or *block* is sampled from the full tensor. Using this block, one optimization step is performed. Due to the structure of a CPD, only a limited amount of variables are affected in each step. This means multiple steps from multiple blocks can be computed in parallel, as long as the affected variables do not overlap. As only small blocks are used, there is no need to load the full tensor. Blocks can also be generated on-the-fly obfuscating the need to construct a tensor beforehand. Thanks to a simple step restriction schedule, the underlying CP structure can be recovered almost as accurately as if the full tensor were decomposed, even if only a fraction of the data is used.

A.5.4 Efficient representation of structured tensors

Tensors are not always given as a multiway array of numbers or as a list of nonzeros or known entries. The tensor can, for example, be given in the Tucker format as a result of randomized compression, as a tensor train approximation [211] to solution of a partial differential equation, or in a Hankel format after tensorization (see Appendix A.3). As discussed in [303], the efficient representation of a tensor $\mathcal{T}$ can be exploited by rewriting the objective function

$$\min \|\mathcal{T} - \hat{\mathcal{T}}\|_{\mathrm{F}}^2 \;=\; \min \|\mathcal{T}\|_{\mathrm{F}}^2 - 2\langle \mathcal{T}, \hat{\mathcal{T}} \rangle + \|\hat{\mathcal{T}}\|_{\mathrm{F}}^2, \tag{A.5}$$

in which $\hat{\mathcal{T}}$ can be a CPD, an LMLRA, an LL1 or a BTD and $\langle \cdot, \cdot \rangle$ is the inner product. The gradients can be rewritten in a similar way. All norms and inner products at the right-hand side of (A.5), and all matricized tensor times Khatri–Rao or Kronecker products needed for the gradients can be computed efficiently by exploiting the structure of $\mathcal{T}$ and $\hat{\mathcal{T}}$. This technique can lead to speedups in many tensor decomposition algorithms, including

ALS, quasi-Newton and NLS algorithms.

Exploiting the structure of tensors by rewriting the objective function as (A.5) does not change the optimization variables. This has as consequence that constraints on factor matrices, symmetry or coupling can be handled trivially. Consider, for example, a nonnegative CPD of a large-scale tensor. Without constraints, the tensor $\mathcal{T}$ is often compressed first to reduce the computational complexity using $\mathcal{S} = \mathcal{T} \bullet_1 \mathbf{U}^{(1)^{\mathrm{T}}} \bullet \cdots \bullet_N \mathbf{U}^{(N)^{\mathrm{T}}}$. The compressed tensor $\mathcal{S}$ is then decomposed as $\left[\!\left[\hat{\mathbf{A}}^{(1)}, \ldots, \hat{\mathbf{A}}^{(N)}\right]\!\right]$. The CPD $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ of $\mathcal{T}$ can be recovered using $\mathbf{A}^{(n)} = \mathbf{U}^{(n)}\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. This technique cannot be used to compute the nonnegative CPD, as non-negativity is not preserved by compression, i.e., $\hat{\mathbf{A}}^{(n)} \geq 0$ does not imply $\mathbf{A}^{(n)} \geq 0$ in which $\geq$ holds entry-wise. However, using the structured format, the optimization variables are the full factor matrices $\mathbf{A}^{(n)}$ instead of the compressed ones $\hat{\mathbf{A}}^{(n)}$. Hence standard nonnegativity techniques can be used, while still exploiting the structure.

A.6 Conclusion

In this chapter, we have elaborated on a number of features that have been introduced by the third release of Tensorlab in March 2016. First of all, new factorizations and constraints are available in the Structured Data Fusion (SDF) framework. A new tool improves the user-friendliness of SDF by finding model errors early. A number of tensorization and detensorization methods have also been added, allowing the transformation of lower-order to higher-order data, and vice versa. By carefully exploiting the symmetry and coupling structure in different stages including the preconditioning stage, new solvers for coupled matrix/tensor factorizations have been enabled. Furthermore, a number of new large-scale approaches have been discussed in this chapter, such as randomized block sampling and decompositions of large structured tensors using efficient representations.

Supplementary materials: canonical polyadic decomposition of incomplete tensors with linearly constrained factors

B

ABSTRACT | Detailed derivations of the CPDLI DD and CPDLI DI algorithms from Chapter 4 are presented in Appendix B.1 and the proposed preconditioners are discussed in Appendix B.2. The parameters used for the experiments are discussed in Appendix B.3.

This appendix provides supplementary material for Chapter 4 [302].

B.1 Derivation of cpdli

The expressions for CPDLI DD and CPDLI DI are derived in Appendix B.1.2 and Appendix B.1.3, respectively. To keep the multilinear structure clear, element-wise expressions are avoided. An overview of useful identities regarding multilinear algebra is given in Appendix B.1.1.

B.1.1 Identities

To derive expressions for the CPD of incomplete tensors, the following identities involving Kronecker, (row-wise) Khatri–Rao and Hadamard products are useful:

$$(\mathbf{A} \odot \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} \odot^{\mathrm{T}} \mathbf{B}^{\mathrm{T}}, \tag{B.1}$$
$$(\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \odot \mathbf{D}) = \mathbf{AB} \odot \mathbf{CD}, \tag{B.2}$$
$$(\mathbf{A} \odot^{\mathrm{T}} \mathbf{C})(\mathbf{B} \otimes \mathbf{D}) = \mathbf{AB} \odot^{\mathrm{T}} \mathbf{CD}, \tag{B.3}$$
$$(\mathbf{A} \odot^{\mathrm{T}} \mathbf{B})(\mathbf{C} \odot \mathbf{D}) = \mathbf{AC} * \mathbf{BD}. \tag{B.4}$$

As explained in subsection 4.1.1, we assume matrix multiplication takes precedence over Kronecker, (row-wise) Khatri–Rao and Hadamard products to reduce the number of parentheses needed. The well-known result for the product of a Kronecker matrix with a vectorized matrix can be extended using permutation matrices $\mathbf{P}^{(n)}$:

$$\begin{aligned}(\mathbf{A} \otimes \mathbf{B})\mathrm{vec}\,(\mathbf{C}) &= \mathrm{vec}\,(\mathbf{BCA}^{\mathrm{T}}),\\ (\mathbf{C} \odot \mathbf{B} \otimes \mathbf{I})\mathrm{vec}\,(\mathbf{A}) &= \mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket),\\ \mathbf{P}^{(2)^{\mathrm{T}}}(\mathbf{C} \odot \mathbf{A} \otimes \mathbf{I})\mathrm{vec}\,(\mathbf{B}) &= \mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket).\end{aligned} \tag{B.5}$$

The Matrix Cookbook [222] provides some useful rules for (vectorized) matrix derivatives:

$$\begin{aligned}\partial \mathrm{vec}\,(\mathbf{AB}) &= (\mathbf{I} \otimes \mathbf{A})\partial \mathrm{vec}\,(\mathbf{B}) + (\mathbf{B}^{\mathrm{T}} \otimes \mathbf{I})\partial \mathrm{vec}\,(\mathbf{A})\\ \partial \mathbf{AB} &= (\partial \mathbf{A})\mathbf{B} + \mathbf{A}(\partial \mathbf{B}).\end{aligned}$$

B.1.2 Derivation of the data-dependent algorithm

The expressions for CPDLI DD (subsection 4.2.1) are derived here. The objective value and Gramian-vector product are already explained in the main text and are therefore omitted.

Gradient

While the computation of the function value is straightforward, the gradient expression is more involved. To compute the gradient $\mathbf{G}^{(n)}$ w.r.t. $\mathbf{C}^{(n)}$, the

matrix unfolding of the residual tensor

$$\mathbf{F}_{(n)} = \mathbf{S}_{(n)} * \left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}} - \mathbf{T}_{(n)}\right) \qquad \text{where} \qquad \mathbf{V}^{(n)} = \bigodot_{k\neq n} \mathbf{A}^{(k)}$$

is used. First, the derivatives w.r.t. $\mathbf{A}^{(n)}$ are computed, after which the chain rule is used to compute the derivative w.r.t. $\mathbf{C}^{(n)}$:

$$\begin{aligned}
\frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} &= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \frac{1}{2} \left|\left|\mathbf{F}_{(n)}\right|\right|_{\mathrm{F}}^2 \\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{S}_{(n)} * \left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}} - \mathbf{T}_{(n)}\right)\right) && \text{(B.6)} \\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \mathrm{diag}\left(\mathrm{vec}\left(\mathbf{S}_{(n)}\right)\right) \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}}\right) && \text{(B.7)} \\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}}\right) \\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \left(\mathbf{V}^{(n)} \otimes \mathbf{I}\right) \\
&= \mathrm{vec}\left(\bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}\right)^{\mathrm{T}}.
\end{aligned}$$

The Hadamard product is written as a matrix multiplication to obtain (B.6). As $\mathbf{F}_{(n)}$ already contains a Hadamard product with $\mathbf{S}_{(n)}$, the diagonal matrix can be omitted in (B.7). The result is reshaped into an $I_n \times R$ matrix $\mathbf{G}^{(n)}$:

$$\mathbf{G}_A^{(n)} = \bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}.$$

The derivative w.r.t. $\mathbf{C}^{(n)}$ is found by applying the chain rule:

$$\begin{aligned}
\mathrm{vec}\left(\mathbf{G}^{(n)}\right)^{\mathrm{T}} = \frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} &= \frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \frac{\partial \mathbf{A}^{(n)}}{\partial \mathbf{C}^{(n)}} \\
&= \mathrm{vec}\left(\mathbf{G}_A^{(n)}\right)^{\mathrm{T}} (\mathbf{I} \otimes \mathbf{B}) \\
&= \mathrm{vec}\left(\mathbf{B}^{(1)^{\mathrm{T}}}\bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}\right)^{\mathrm{T}}.
\end{aligned}$$

Next, the structure of $\mathbf{V}^{(n)}$ is exploited by using its definition in terms of $\mathbf{A}^{(n)}$ and identity (B.2):

$$\begin{aligned}
\mathbf{G}^{(n)} &= \mathbf{B}^{(1)^{\mathrm{T}}}\bar{\mathbf{F}}_{(n)} \left(\bigotimes_{k\neq n} \mathbf{B}^{(k)}\right) \left(\bigodot_{k\neq n} \mathbf{C}^{(k)}\right) \\
&= \bar{\mathbf{Z}}_{(n)} \left(\bigodot_{k\neq n} \mathbf{C}^{(k)}\right)
\end{aligned} \qquad \text{(B.8)}$$

in which the (conjugated) auxiliary tensor $\bar{\mathcal{Z}} = \bar{\mathcal{F}} \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}$ is used.

Gramian

The Gramian of the Jacobian $\mathbf{H}$ is computed as $\mathbf{J}^{\mathrm{H}}\mathbf{J}$ in which $\mathbf{J}$ is the Jacobian of the objective function (4.4). The Jacobian matrix is divided w.r.t. the factor matrices, such that $\mathbf{J} = \begin{bmatrix}\mathbf{J}^{(1)} & \cdots & \mathbf{J}^{(N)}\end{bmatrix}$. Consequently, we have $N \times N$ blocks in the Gramian, i.e., $\mathbf{H}^{(m,n)} = \mathbf{J}^{(m)^{\mathrm{H}}}\mathbf{J}^{(n)}$. The Jacobian $\mathbf{J}^{(n)}$ is the derivative of the vectorized residual $\mathcal{F}$ w.r.t. the vectorized factor matrix $\mathbf{C}^{(n)}$ and is given by

$$\begin{aligned}
\mathbf{J}^{(n)} &= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \mathrm{vec}\left(\mathcal{F}\right) \\
&= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{A}^{[1]} * \cdots * \mathbf{A}^{[N]}\right) \mathbf{1} \\
&= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]}\mathbf{I} * \mathbf{B}^{[n]}\mathbf{C}^{(n)}\right) \mathbf{1} \\
&= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \left(\mathbf{I} \odot \mathbf{C}^{(n)}\right) \mathbf{1} \\
&= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \mathrm{vec}\left(\mathbf{C}^{(n)}\right) \\
&= \mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]},
\end{aligned}$$

where we have used the definition of $\mathbf{V}^{[n]} = *_{k \neq n} \mathbf{A}^{[n]}$, followed by identities (B.4) and (B.5). Each Gramian block is then given by

$$\mathbf{H}^{(m,n)} = \left(\mathbf{V}^{[m]^{\mathrm{H}}} \odot \mathbf{B}^{[m]^{\mathrm{H}}}\right) \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right). \tag{B.9}$$

This expression is further simplified in the data-independent formulation.

B.1.3 Derivation of the data-independent algorithm

The derivation of CPDLI DI (subsection 4.2.2) is discussed here.

Objective function

The norm in the objective function f is expanded as

$$\min f = \frac{1}{2} f_1 - \mathrm{Re}\left(f_2\right) + \frac{1}{2} f_3.$$

The first term is computed as $f_1 = ||\mathcal{S} * \mathcal{T}||^2 = \hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}}$ with $\hat{\mathbf{t}} \in \mathbb{K}^{N_{\mathrm{ke}}}$ containing the known entries of $\mathcal{T}$. The second term f_2 is given by

$$\begin{aligned} f_2 &= \left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right], \mathcal{T} \right\rangle_{\mathcal{S}} \\ &= \hat{\mathbf{t}}^{\mathrm{H}} \left(\mathbf{A}^{[N]} * \cdots * \mathbf{A}^{[1]} \right) \mathbf{1} \\ &= \hat{\mathbf{t}}^{\mathrm{H}} \left(\mathbf{B}^{[N]}\mathbf{C}^{(n)} * \cdots * \mathbf{B}^{[1]}\mathbf{C}^{(1)} \right) \mathbf{1} \\ &= \hat{\mathbf{t}}^{\mathrm{H}} \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right) \left(\bigodot_n \mathbf{C}^{(n)} \right) \mathbf{1} \\ &= \mathrm{vec}\left(\left(\mathcal{S} * \bar{\mathcal{T}} \right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \right)^{\mathrm{T}} \mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right). \end{aligned}$$

We subsequently applied the definitions of the weighted inner product and extended matrices, the definition of $\mathbf{A}^{[n]}$, identity (B.4) and finally the definition of tensor-matrix products and of a vectorized CPD. Finally, the third term is derived as

$$\begin{aligned} f_3 &= \left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right], \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] \right\rangle_{\mathcal{S}} \\ &= \mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right)^{\mathrm{H}} \left[\left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{H}} \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right) \right] \mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right) \\ &= \mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right)^{\mathrm{H}} \mathbf{D}\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right), \end{aligned}$$

in which we applied the definition of $\mathbf{A}^{(n)}$, identity (B.4), and the definition of $\mathbf{D}$.

Gradient

To derive the expression for the gradient in the data-independent algorithm, we expand the gradient result (B.8) using the definition of the residual tensor $\mathcal{F} = \mathcal{S} * \left(\left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] - \mathcal{T} \right)$:

$$\mathbf{G}^{(n)} = \mathbf{G}_1^{(n)} - \mathbf{G}_2^{(n)}.$$

To compute the first part $\mathbf{G}_1^{(n)} = \bar{\mathbf{Z}}_{(n)} \odot_n \mathbf{C}^{(n)}$ we use $\bar{\mathcal{Z}}$ which is given by

$$\begin{aligned} \mathrm{vec}\left(\bar{\mathcal{Z}}\right) &= \mathrm{vec}\left(\left(\mathcal{S} * \left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)} \right]\!\right] \right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \right) \\ &= \left(\bigotimes_n \mathbf{B}^{(n)} \right)^{\mathrm{T}} \mathrm{diag}\left(\mathrm{vec}\left(\mathcal{S} \right) \right) \mathrm{vec}\left(\left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)} \right]\!\right] \right) \\ &= \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{T}} \left(\bar{\mathbf{B}}^{[N]}\bar{\mathbf{C}}^{(n)} * \cdots * \bar{\mathbf{B}}^{[1]}\bar{\mathbf{C}}^{(1)} \right) \mathbf{1} \end{aligned} \tag{B.10}$$

$$
\begin{aligned}
&= \left(\bigodot_n{}^{\mathrm{T}} \mathbf{B}^{[n]}\right)^{\mathrm{T}} \left(\bigodot_n{}^{\mathrm{T}} \bar{\mathbf{B}}^{[n]}\right) \left(\bigodot_n \bar{\mathbf{C}}^{(n)}\right) \mathbf{1} \\
&= \bar{\mathbf{D}} \mathrm{vec}\left(\left[\!\left[\bar{\mathbf{C}}^{(1)}, \ldots, \bar{\mathbf{C}}^{(N)}\right]\!\right]\right).
\end{aligned}
$$

We first applied the Kronecker definition of tensor-matrix products. In (B.10) the zero rows created by $\mathcal{S}$ are ignored and the definitions of $\mathbf{A}^{(n)}$ and a vectorized CPD are used. Finally, we again apply identity (B.4) and the definitions of $\mathbf{D}$ and a vectorized CPD.

The second part $\mathbf{G}_2^{(n)}$ is computed as

$$
\begin{aligned}
\mathbf{G}_2^{(n)} &= \left(\bar{\mathcal{T}}_B\right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right) \\
\bar{\mathcal{T}}_B &= \left(\mathcal{S} * \bar{\mathcal{T}}\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}.
\end{aligned}
$$

Combining $\mathbf{G}_1^{(n)}$ and $\mathbf{G}_2^{(n)}$ leads to

$$
\mathbf{G}^{(n)} = \left(\bar{\mathcal{Z}} - \bar{\mathcal{T}}_B\right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right).
$$

Gramian

Equation (B.9) is the starting point for the derivation of each block in the Gramian:

$$
\begin{aligned}
\mathbf{H}^{(m,n)} &= \left(\mathbf{V}^{[m]^{\mathrm{H}}} \odot \mathbf{B}^{[m]^{\mathrm{H}}}\right) \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right) \\
&= \left(\ast_{k \neq m} \mathbf{B}^{[k]} \mathbf{C}^{(k)} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \left(\ast_{k \neq n} \mathbf{B}^{[k]} \mathbf{C}^{(k)} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \\
&= \left[\left(\bigodot_{k \neq m}{}^{\mathrm{T}} \mathbf{B}^{[k]}\right) \left(\bigodot_{k \neq m} \mathbf{C}^{(k)}\right) \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right]^{\mathrm{H}} \cdot \\
&\qquad \left[\left(\bigodot_{k \neq n}{}^{\mathrm{T}} \mathbf{B}^{[k]}\right) \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right) \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right] \qquad \text{(B.11)} \\
&= \left[\left(\bigodot_{k \neq m}{}^{\mathrm{T}} \mathbf{B}^{[k]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right) \left(\bigodot_{k \neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)\right]^{\mathrm{H}} \cdot \qquad \text{(B.12)} \\
&\qquad \left[\left(\bigodot_{k \neq n}{}^{\mathrm{T}} \mathbf{B}^{[k]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)\right] \\
&= \left(\bigodot_{k \neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \left(\bigodot_k{}^{\mathrm{T}} \mathbf{B}^{[k]}\right)^{\mathrm{H}} \left(\bigodot_k{}^{\mathrm{T}} \mathbf{B}^{[k]}\right) \mathbf{P}^{(n)^{\mathrm{T}}} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right) \\
&= \left(\bigodot_{k \neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \mathbf{D} \mathbf{P}^{(n)^{\mathrm{T}}} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right) \qquad \text{(B.13)}
\end{aligned}
$$

First, we applied identity (B.1) and the definition of $\mathbf{V}^{[n]}$, followed by (B.4). From (B.11) to (B.12) identity (B.3) is used, after which we use the properties of permutation matrices. Finally, the definition of $\mathbf{D}$ is used to arrive at the KRMTKRPROD.

Gramian vector product

As shown in the main text, the Gramian-vector products $\mathbf{Y}^{(m)}$ can be computed by first summing over n:

$$\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) = \sum_{n=1}^{N} \mathbf{H}^{(m,n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right).$$

After substituting the definition of $\mathbf{H}^{(m,n)}$ in (B.13) and applying identity (B.5), we find

$$\begin{aligned}\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) &= \sum_{n=1}^{N} \left(\bigodot_{k\neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \mathbf{D} \mathbf{P}^{(n)^{\mathrm{T}}} \left(\bigodot_{k\neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right) \operatorname{vec}\left(\mathbf{X}^{(n)}\right) \\ &= \left(\bigodot_{k\neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \mathbf{D} \sum_{n=1}^{N} \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right).\end{aligned}$$

Again, an auxiliary tensor $\mathcal{Z}$

$$\operatorname{vec}(\mathcal{Z}) = \mathbf{D} \sum_{n=1}^{N} \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right),$$

is used to simplify the notations:

$$\begin{aligned}\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) &= \left(\bigodot_{k\neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \operatorname{vec}(\mathcal{Z}) \\ \mathbf{Y}^{(m)} &= \mathbf{Z}_{(m)} \left(\bigodot_{k\neq m} \bar{\mathbf{C}}^{(k)}\right).\end{aligned}$$

B.2 Preconditioner

To prove the expressions for the preconditioner for the computation of a CPD of an incomplete tensor without (Equation (4.21)) and with (Equation (4.23)) linear constraints, we start from the block diagonal approximation of the Gramian if all entries are known. First, linear constraints are ignored. Each diagonal block is given as $\mathbf{H}^{(n,n)} = \mathbf{W} \otimes \mathbf{I}_{I_n}$ [260]. Element-wise, each

nonzero entry is therefore given by

$$h^{(n,n)}_{(i_n,r),(i_n,s)} = \sum_{i_1=1}^{I_1} \cdots \sum_{i_{n-1}=1}^{I_{n-1}} \sum_{i_{n+1}=1}^{I_{n+1}} \cdots \sum_{i_N=1}^{I_N} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s} = \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}_{i_n}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}, \tag{B.14}$$

in which the set $\pi^{(n)}_{i_n}$ contains all multi-indices in the i_nth mode-n slice of order $N-1$. As all entries are known $\pi^{(n)}_{i_n} = \pi^{(n)}$ for $i_n = 1,\ldots,I_n$ and

$$\pi^{(n)} = \Big\{ (i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N) \;\Big|\; i_k = 1,\ldots,I_k \text{ and } k = 1,\ldots,n-1,n+1,\ldots,N \Big\}.$$

In the case of missing entries, the number and position of missing entries can vary in each slice. Therefore, the sets $\pi^{(n)}_{i_n}$ in (B.14) are replaced by the sets $\hat{\pi}^{(n)}_{i_n}$ containing the multi-indices of the known entries in each slice:

$$\hat{\pi}^{(n)}_{i_n} = \Big\{ (i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N) \;\Big|\; (i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N) \in \pi^{(n)}; s_{i_1,\ldots,i_N} = 1 \Big\}.$$

Using index sets $\hat{\pi}^{(n)}_{i_n}$, (B.14) boils down to element-wise expression for the diagonal blocks of the Gramian for incomplete tensors.

The set $\hat{\pi}^{(n)}_{i_n}$ contains $Q^{(n)}_{i_n}$ elements and corresponds to one particular distribution of entries. Let $\Pi^{(n)}_{i_n}$ be the set of all possible partitions of $Q^{(n)}_{i_n}$ known entries, i.e.,

$$\Pi^{(n)}_{i_n} = \Big\{ \hat{\pi}^{(n)} \Big| \, |\hat{\pi}^{(n)}| = Q^{(n)}_{i_n} \Big\}.$$

If all partitions $\hat{\pi}^{(n)}$ in $\Pi^{(n)}_{i_n}$ are equally likely, the expected value of an entry $h^{(n,n)}_{(i_n,r),(i_n,s)}$ over $\Pi^{(n)}_{i_n}$ is given by

$$\mathbb{E}\left[h^{(n,n)}_{(i_n,r),(i_n,s)}\right] = \frac{1}{|\Pi^{(n)}_{i_n}|} \sum_{\hat{\pi}^{(n)}\in\Pi^{(n)}_{i_n}} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\hat{\pi}^{(n)}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}.$$

It can be verified that each product $\prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}$ appears $|\Pi^{(n)}_{i_n}| \frac{Q^{(n)}_{i_n}}{\prod_{k\neq n} I_k}$

times, hence

$$\mathbb{E}\left[h^{(n,n)}_{(i_n,r),(i_n,s)}\right] = \frac{N^{(n)}_{i_n}}{\prod_{k\neq n} I_k} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}.$$

The fraction $\frac{Q^{(n)}_{i_n}}{\prod_{k\neq n} I_k}$ is exactly the fraction of known entries $f^{(n)}_{i_n} = \mathbf{f}^{(n)}(i_n)$ in the i_nth mode-n slice. The expected value for the block can then be written as

$$\mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \operatorname{diag}(\mathbf{f}^{(n)}),$$

which concludes the derivation of (4.21).

In the case of linear constraints, each Gramian entry is given by

$$h^{(n,n)}_{(d_1,r),(d_2,s)} = \sum_{i_n=1}^{I_n} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}} \overline{b}^{(n)}_{i_n,d_1} b^{(n)}_{i_n,d_2} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}$$

Using a similar derivation, the expected value of the block diagonal of the Gramian with respect to all possible partitions is given by

$$\mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \left(\mathbf{B}^{(n)^{\mathrm{H}}} \operatorname{diag}(\mathbf{f}^{(n)}) \mathbf{B}^{(n)}\right),$$

which is expression (4.23).

B.3 Experiment parameters

B.3.1 CPD of incomplete tensors

The `cpd_nls` implementation from Tensorlab [305] is used for the NLS-based algorithms CPD and CPDI (using the `UseCPDI = true` option). In the case of CPDI, the statistical preconditioner $\mathbf{M}_{\text{INC,STAT}}$ presented here is used, as well as an optimized C implementation to construct the sparse Jacobian matrices. Both algorithms use at most `CGMaxIter = 40` CG iterations. The MINF algorithm [305] is implemented using `cpd_minf` with the `minf_ncg` solver which uses NCG with modified Hestenes–Stiefel updates and Moré–Thuente line search. The CP WOPT algorithm [6], [17] uses NCG with the same settings. The Tensorlab routines use `TolFun = 1e-24` and `TolFun = 1e-11` as stopping criterion for the high and low accuracy experiments, respectively. Other parameters are set such that they do not influence convergence. CP WOPT uses `StopTol = 1e-11` and `RelFuncTol = 1e-11` as stopping criterion.

B.3.2 Materials science application

Our implementation of MVR ALS from [27] uses the backslash operator to solve the linear system in contrast to the CG approach suggested in [27], and computes the function value every iteration. The SDF implementation from Tensorlab [305] uses `struct_matvec` to implement the linear constraint and the `cpdi` kernel presented in this chapter, but without any preconditioner. The CPDLI algorithms use the small-scale version, i.e., the Gramian $\mathbf{J}^{\mathrm{H}}\mathbf{J}$ is constructed and inverted instead of using CG. SDF and CPDI perform at most `CGMaxIter = 150` CG iterations, while the CPDLI algorithms use at most `CGMaxIter = 15` iterations. The CPDI algorithm take advantage of the statistical block-Jacobi preconditioner $\mathbf{M}_{\mathrm{INC,STAT}}$.

Bibliography

[1] K. Abed–Meraim, W. Qiu, and Y. Hua, "Blind system identification", *Proc. IEEE*, vol. 85, no. 8, pp. 1310–1322, Aug. 1997.

[2] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations with missing data", in *SIAM International Conference on Data Mining*, 2010, pp. 701–712.

[3] E. Acar, T. G. Kolda, and D. M. Dunlavy, *All-at-once optimization for coupled matrix and tensor factorizations*, May 2011. arXiv: 1105.3422.

[4] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey", *IEEE Trans. Knowl. Data Eng*, vol. 21, no. 1, pp. 6–20, 2009.

[5] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions", *J. Chemometrics*, vol. 25, no. 2, pp. 67–86, Jan. 2011. DOI: 10.1002/cem.1335.

[6] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data", *Chemometr. Intell. Lab.*, vol. 106, no. 1, pp. 41–56, Mar. 2011. DOI: 10.1016/j.chemolab.2010.08.004.

[7] 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. DOI: 10.1186/1471-2105-15-239.

[8] T. Adalı, P. J. Schreier, and L. L. Scharf, "Complex-valued signal processing: The proper way to deal with impropriety", *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5101–125, Nov. 2011. DOI: 10.1109/tsp.2011.2162954.

[9] T. Adalı, Y. Levin-Schwartz, and V. D. Calhoun, "Multimodal data fusion using source separation: Application to medical imaging", *Proc. IEEE*, vol. 103, no. 9, pp. 1494–1506, Sep. 2015. DOI: 10.1109/jproc.2015.2461601.

[10] S. Allen and J. Cahn, "Ground state structures in ordered binary alloys with second neighbor interactions", *Acta Metallurgica*, vol. 20, no. 3, pp. 423–433, Mar. 1972. DOI: 10.1016/0001-6160(72)90037-5.

[11] A. de Almeida, G. Favier, and J. C. M. Mota, "A constrained factor decomposition with application to MIMO antenna systems", *IEEE Trans. Signal Process.*, vol. 56, no. 6, pp. 2429–2442, Jun. 2008. DOI: 10.1109/TSP.2008.917026.

[12] B. K. Alsberg and O. M. Kvalheim, "Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication", *Chemometr. Intell. Lab.*, vol. 24, no. 1, pp. 31–42, Jun. 1994. DOI: 10.1016/0169-7439(94)00013-1.

[13] J. Andersson and J. Ågren, "Models for numerical treatment of multicomponent diffusion in simple phases", *Journal of Applied Physics*, vol. 72, no. 4, pp. 1350–1355, Aug. 1992. DOI: 10.1063/1.351745.

[14] J.-O. Andersson, T. Helander, L. Höglund, P. Shi, and B. Sundman, "Thermo-Calc & DICTRA, computational tools for materials science", *Calphad*, vol. 26, no. 2, pp. 273–312, Jun. 2002. DOI: 10.1016/s0364-5916(02)00037-8.

[15] R. Badeau and R. Boyer, "Fast multilinear singular value decomposition for structured tensors", *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1008–1021, Jan. 2008. DOI: 10.1137/060655936.

[16] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors", *SIAM J. Sci. Comput.*, vol. 30, no. 1, pp. 205–231, Jan. 2008. DOI: 10.1137/060676489.

[17] B. W. Bader, T. G. Kolda, *et al.*, *Matlab tensor toolbox version 2.6*, Available online at http://www.sandia.gov/~tgkolda/TensorToolbox/, Feb. 2015.

[18] E.-W. Bai and Y. Liu, "On the least squares solutions of a system of bilinear equations", in *Proceedings of the 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference (CDC-ECC 2005, Seville, Spain)*, Dec. 2005, pp. 1197–1202. DOI: 10.1109/CDC.2005.1582321.

[19] J. Ballani and L. Grasedyck, "A projection method to solve linear systems in tensor format", *Numer. Linear Algebra Appl.*, vol. 20, pp. 27–43, Jan. 2013. DOI: 10.1002/nla.1818.

[20] G. Ballard, N. Knight, and K. Rouse, *Communication lower bounds for matricized tensor times Khatri–Rao product*, Oct. 2017. arXiv: 1708.07401.

[21] G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, and et al., "Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models", *Proc. IEEE*, vol. 93, no. 2, pp. 276–292, Feb. 2005. DOI: 10.1109/jproc.2004.840311.

[22] M. Bebendorf and R. Kriemann, "Fast parallel solution of boundary integral equations and related problems", *Computing and Visualization in Science*, vol. 8, no. 3-4, pp. 121–135, Nov. 2005. DOI: 10.1007/s00791-005-0001-x.

[23] M. Bebendorf and S. Rjasanow, "Adaptive low-rank approximation of collocation matrices", *Computing*, vol. 70, no. 1, pp. 1–24, Feb. 2003. DOI: 10.1007/s00607-002-1469-6.

[24] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods", in *Proceedings of the 1988 connectionist models summer school*, San Matteo, CA: Morgan Kaufmann, 1988, pp. 29–37.

[25] M. Benzi, "Preconditioning techniques for large linear systems: A survey", *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, Nov. 2002. DOI: 10.1006/jcph.2002.7176.

[26] G. Beylkin and M. J. Mohlenkamp, "Numerical operator calculus in higher dimensions", *Proceedings of the National Academy of Sciences*, vol. 99, no. 16, pp. 10 246–10 251, Jul. 2002. DOI: 10.1073/pnas.112329799.

[27] G. Beylkin, J. Garcke, and M. J. Mohlenkamp, "Multivariate regression and machine learning with sums of separable functions", *SIAM J. Sci. Comput.*, vol. 31, no. 3, pp. 1840–1857, Jan. 2009. DOI: 10.1137/070710524.

[28] G. Beylkin and M. J. Mohlenkamp, "Algorithms for numerical analysis in high dimensions", *SIAM J. Sci. Comput.*, vol. 26, no. 6, pp. 2133–2159, Jul. 2005. DOI: 10.1137/040604959.

[29] H. N. Bharath, D. Sima, N. Sauwen, U. Himmelreich, L. De Lathauwer, and S. Van Huffel, "Nonnegative canonical polyadic decomposition for tissue-type differentiation in gliomas", *IEEE J. Biomed. Health Inform.*, vol. 21, no. 4, pp. 1124–1132, Jul. 2017. DOI: 10.1109/JBHI.2016.2583539.

[30] A. Bloom, *The republic of Plato*. Basic Books, 1968, Second edition.

[31] B.-Z. Bobrovsky, E. Mayer-Wolf, and M. Zakai, "Some classes of global Cramér–Rao bounds", *Ann. Statist.*, vol. 15, no. 4, pp. 1421–1438, 1987.

[32] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent", *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, 2009.

[33] B. Böttger, J. Eiken, and M. Apel, "Multi-ternary extrapolation scheme for efficient coupling of thermodynamic data to a multi-phase-field model", *Computational Materials Science*, vol. 108, pp. 283–292, Oct. 2015. DOI: 10.1016/j.commatsci.2015.03.003.

[34] L. Bottou, "Stochastic gradient tricks", in *Neural Networks, Tricks of the Trade, Reloaded*, ser. Lecture Notes in Computer Science (LNCS 7700), G. Montavon, G. B. Orr, and K.-R. Müller, Eds., Springer, 2012, pp. 430–445.

[35] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning", in *Optimization for Machine Learning*, S. Sra, S. Nowozin, and S. J. Wright, Eds., MIT Press, 2011, pp. 351–368.

[36] M. Boussé, O. Debals, and L. De Lathauwer, "A tensor-based method for large-scale blind source separation using segmentation", *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 346–358, Jan. 2017. DOI: 10.1109/TSP.2016.2617858.

[37] ——, "Tensor-based large-scale blind system identification using segmentation", *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5770–5784, Nov. 2017. DOI: 10.1109/TSP.2017.2736505.

[38] M. Boussé, G. Goovaerts, N. Vervliet, O. Debals, S. Van Huffel, and L. De Lathauwer, "Irregular heartbeat classification using Kronecker product equations", in *39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2017)*, Jul. 2017, pp. 438–441. DOI: 10.1109/EMBC.2017.8036856.

[39] M. Boussé, N. Vervliet, O. Debals, and L. De Lathauwer, "Face recognition as a Kronecker product equation", in *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec. 2017, pp. 276–280.

[40] M. Boussé, N. Vervliet, I. Domanov, O. Debals, and L. De Lathauwer, "Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications", Technical Report 17-01, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017.

[41] M. Boussé, I. Domanov, and L. De Lathauwer, "Linear systems with a multilinear singular value decomposition constrained solution", ESAT-STADIUS, KU Leuven, Belgium, Tech. Rep., 2017.

[42] M. Brazell, N. Li, C. Navasca, and C. Tamon, "Solving multilinear systems via tensor inversion", *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 2, pp. 542–570, May 2013. DOI: 10.1137/100804577.

[43] R. Bro, "PARAFAC. Tutorial and applications", *Chemometr. Intell. Lab.*, vol. 38, no. 2, pp. 149–171, 1997. DOI: 10.1016/S0169-7439(97)00032-4.

[44] R. Bro, R. A. Harshman, N. D. Sidiropoulos, and M. E. Lundy, "Modeling multi-way data with linearly dependent loadings", *J. Chemometrics*, vol. 23, no. 7-8, pp. 324–340, 2009. DOI: 10.1002/cem.1206.

[45] R. Bro, "Multi-way analysis in the food industry: Models, algorithms, and applications", PhD thesis, University of Amsterdam, 1998.

[46] R. Bro and C. A. Andersson, "Improving the speed of multiway algorithms: Part II: Compression", *Chemometr. Intell. Lab.*, vol. 42, no. 1–2, pp. 105–113, 1998. DOI: 10.1016/S0169-7439(98)00011-2.

[47] R. Bro and S. De Jong, "A fast non-negativity-constrained least squares algorithm", *J. Chemometrics*, vol. 11, no. 5, pp. 393–401, Sep. 1997. DOI: 10.1002/(sici)1099-128x(199709/10)11:5<393::aid-cem483>3.3.co;2-c.

[48] R. Bro, R. A. Harshman, N. D. Sidiropoulos, and M. E. Lundy, "Modeling multi-way data with linearly dependent loadings", *J. Chemometrics*, vol. 23, no. 7-8, pp. 324–340, Jul. 2009. DOI: 10.1002/cem.1206.

[49] R. Cabral Farias, J. E. Cohen, and P. Comon, "Exploring multimodal data fusion through joint decompositions with flexible couplings", *IEEE Trans. Signal Process.*, vol. 64, no. 18, pp. 4830–4844, Sep. 2016. DOI: 10.1109/tsp.2016.2576425.

[50] J. W. Cahn and J. E. Hilliard, "Free energy of a nonuniform system. I. Interfacial free energy", *The Selected Works of John W. Cahn*, pp. 29–38, Oct. 2013. DOI: 10.1002/9781118788295.ch4.

[51] C. F. Caiafa and A. Cichocki, "Generalizing the column–row matrix decomposition to multi-way arrays", *Linear Algebra Appl.*, vol. 433, no. 3, pp. 557–573, Sep. 2010. DOI: 10.1016/j.laa.2010.03.020.

[52] ——, "Multidimensional compressed sensing and their applications", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 6, pp. 355–380, Oct. 2013. DOI: 10.1002/widm.1108.

[53] J. A. Calvin and E. F. Valeev, "TiledArray: A general-purpose scalable block-sparse tensor framework", URL: https://github.com/valeevgroup/tiledarray.

[54] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling", *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 21–30, Mar. 2008. DOI: 10.1109/msp.2007.914731.

[55] C. Cardon, R. Le Tellier, and M. Plapp, "Modelling of liquid phase segregation in the Uranium–Oxygen binary system", *Calphad*, vol. 52, pp. 47–56, Mar. 2016. DOI: 10.1016/j.calphad.2015.10.005.

[56] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart–Young" decomposition", *Psychometrika*, vol. 35, no. 3, pp. 283–319, Sep. 1970. DOI: 10.1007/bf02310791.

[57] J. D. Carroll, 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, Mar. 1980. DOI: 10.1007/bf02293596.

[58] CERN, "Processing: What to record?", URL: https://home.cern/about/computing/processing-what-record (visited Mar. 20, 2018).

[59] ——, "The Higgs boson", URL: https://home.cern/topics/higgs-boson (visited Mar. 20, 2018).

[60] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics", *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 32–43, Sep. 2014. DOI: 10.1109/MSP.2014.2329397.

[61] P. T. Chavda and S. Solanki, "Illumination invariant face recognition based on PCA (eigenface)", *International Journal of Engineering Development and Research*, vol. 2, no. 2, pp. 2155–2162, Jun. 2014.

[62] 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, Dec. 2012. DOI: 10.1137/110859063.

[63] D. Choi, J.-G. Jang, and U. Kang, *Fast, accurate, and scalable method for sparse coupled matrix-tensor factorization*, Dec. 2017. arXiv: 1708.08640.

[64] J. H. Choi and S. V. N. Vishwanathan, "DFacTo: Distributed factorization of tensors", in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14, Cambridge, MA, USA: MIT Press, 2014, pp. 1296–1304.

[65] A. Cichocki, D. Mandic, A.-H. Phan, C. Caiafa, G. Zhou, Q. Zhao, and L. De Lathauwer, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis", *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015. DOI: 10.1109/msp.2013.2297439.

[66] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S.-I. Amari, "Non-negative tensor factorization using alpha and beta divergences", *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, 2007. DOI: 10.1109/icassp.2007.367106.

[67] A. Cichocki, R. Zdunek, A.-H. Phan, and S.-I. Amari, *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation*. Chichester, U.K: John Wiley, 2009.

[68] L. Clarke, X. Zheng-Bradley, R. Smith, E. Kulesha, C. Xiao, I. Toneva, B. Vaughan, D. Preuss, R. Leinonen, M. Shumway, S. Sherry, P. Flicek, and T. 1. G. P. Consortium, "The 1000 genomes project: Data management and community access", *Nature Methods*, vol. 9, 459 EP, Apr. 2012. DOI: 10.1038/nmeth.1974.

[69] J. E. Cohen, R. Cabral Farias, and P. Comon, "Fast decomposition of large nonnegative tensors", *IEEE Signal Process. Lett.*, vol. 22, no. 7, pp. 862–866, Jul. 2015. DOI: 10.1109/lsp.2014.2374838.

[70] S. Cohen and C. Tomasi, "Systems of bilinear equations", Technical Report, 1997.

[71] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Academic press, 2010.

[72] P. Comon, X. Luciani, and A. L. F. de Almeida, "Tensor decompositions, alternating least squares and other tales", *J. Chemometrics*, vol. 23, no. 7-8, pp. 393–405, Jul. 2009. DOI: 10.1002/cem.1236.

[73] P. Comon, "Tensors: A brief introduction", *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 44–53, May 2014. DOI: 10.1109/msp.2014.2298533.

[74] Y. Coutinho, N. Vervliet, L. De Lathauwer, and N. Moelans, "Efficient use of CALPHAD based data in phase-field spinodal decomposition simulations for a quaternary system through decomposed thermodynamic tensor models", Technical Report 18–51, ESAT-STADIUS, KU Leuven, Belgium, 2018.

[75] C. Darken and J. Moody, "Towards faster stochastic gradient descent", in *NIPS*, 1991, pp. 1009–1016.

[76] 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, Sep. 2006. DOI: 10.1137/040608830.

[77] ——, "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, Sep. 2008. DOI: 10.1137/070690729.

[78] 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, Jul. 2000. DOI: 10.1137/S0895479896305696.

[79] ——, "On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors", *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1324–1342, Jul. 2000. DOI: 10.1137/S0895479898346995.

[80] L. De Lathauwer, "Blind separation of exponential polynomials and the decomposition of a tensor in rank-$(L_r, L_r, 1)$ terms", *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 4, pp. 1451–1474, 2011. DOI: 10.1137/100805510.

[81] L. De Lathauwer and E. Kofidis, "Coupled matrix-tensor factorizations — The case of partially shared factors", Technical Report 17-171, ESAT-STADIUS, KU Leuven, Belgium. Accepted for publication in 2017 51th Asilomar Conference on Signals, Systems and Computers, 2017.

[82] 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, Jan. 2008. DOI: 10.1137/070690730.

[83] L. De Lathauwer and J. Vandewalle, "Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \ldots, R_N)$ reduction in multilinear algebra", *Linear Algebra Appl.*, vol. 391, pp. 31–55, Nov. 2004. DOI: 10.1016/j.laa.2004.01.016.

[84] O. Debals and L. De Lathauwer, "Stochastic and deterministic tensorization for blind signal separation", in *Latent Variable Analysis and Signal Separation*, ser. Lecture Notes in Computer Science, vol. 9237, Springer Berlin / Heidelberg, 2015, pp. 3–13.

[85] ——, "The concept of tensorization", Technical Report 17–99, ESAT-STADIUS, KU Leuven, Belgium, 2017.

[86] O. Debals, L. De Lathauwer, and M. Van Barel, "About higher-order Löwner tensors", Technical Report 17–98, ESAT-STADIUS, KU Leuven, Belgium, 2017.

[87] O. Debals, M. Sohail, and L. De Lathauwer, "Analytical multi-modulus algorithms based on coupled canonical polyadic decompositions", Technical Report 16–150, ESAT-STADIUS, KU Leuven, Belgium, 2016.

[88] O. Debals, F. Van Eeghem, N. Vervliet, and L. De Lathauwer, "Tensorlab demos — Release 3.0", Technical Report 16–68, ESAT-STADIUS, KU Leuven, Belgium, 2016.

[89] O. Debals and N. Vervliet, "Efficiënte tensorgebaseerde methoden voor modellering en signaalscheiding", (Dutch), Master's thesis, KU Leuven, 2013.

[90] 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, Apr. 2016. DOI: 10.1109/tsp.2015.2500179.

[91] S. Diamond and S. Boyd, "Matrix-free convex optimization modeling", *Springer Optimization and Its Applications*, pp. 221–264, 2016. DOI: 10.1007/978-3-319-42056-1_7.

[92] W. Ding and Y. Wei, "Solving multilinear systems with $\mathcal{M}$-tensors", *J. Sci. Comput.*, vol. 68, no. 2, pp. 689–715, Aug. 2016. DOI: 10.1007/s10915-015-0156-7.

[93] W. Ding, L. Qi, and Y. Wei, "Fast Hankel tensor-vector product and its application to exponential data fitting", *Numer. Linear Algebra Appl.*, vol. 22, no. 5, pp. 814–832, Feb. 2015. DOI: 10.1002/nla.1970.

[94] A. T. Dinsdale, "SGTE data for pure elements", *Calphad*, vol. 15, no. 4, pp. 317–425, Oct. 1991. DOI: 10.1016/0364-5916(91)90030-n.

[95] 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, Jul. 2013. DOI: 10.1137/120877234.

[96] ——, "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, Jul. 2013. DOI: 10.1137/120877258.

[97] ——, "Generic uniqueness conditions for the canonical polyadic decomposition and INDSCAL", *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 4, pp. 1567–1589, Nov. 2015. DOI: 10.1137/140970276.

[98] ——, "Canonical polyadic decomposition of third-order tensors: Relaxed uniqueness conditions and algebraic algorithm", *Linear Algebra Appl.*, vol. 513, pp. 342–375, Jan. 2017. DOI: 10.1016/j.laa.2016.10.019.

[99] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition", *SIAM J. Matrix Anal. Appl.*, vol. 35, no. 2, pp. 636–660, May 2014. DOI: 10.1137/130916084.

[100] I. Domanov, A. Stegeman, and L. De Lathauwer, "On the largest multilinear singular values of higher-order tensors", *SIAM J. Matrix Anal. Appl.*, vol. 38, no. 4, pp. 1434–1453, Jan. 2017. DOI: 10.1137/16m110770x.

[101] D. Donoho, "Compressed sensing", *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006. DOI: 10.1109/tit.2006.871582.

[102] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk, "Single-pixel imaging via compressive sampling", *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 83–91, Mar. 2008. DOI: 10.1109/msp.2007.914730.

[103] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank", *Psychometrika*, vol. 1, no. 3, pp. 211–218, Sep. 1936. DOI: 10.1007/bf02288367.

[104] J. Eiken, B. Böttger, and I. Steinbach, "Multiphase-field approach for multicomponent alloys with extrapolation scheme for numerical application", *Physical Review E*, vol. 73, no. 6, Jun. 2006. DOI: 10.1103/physreve.73.066122.

[105] B. Ermiş, E. Acar, and A. T. Cemgil, "Link prediction in heterogeneous data via generalized coupled tensor factorization", *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 203–236, Dec. 2013. DOI: 10.1007/s10618-013-0341-y.

[106] M. Espig, W. Hackbusch, T. Rohwedder, and R. Schneider, "Variational calculus with sums of elementary tensors of fixed rank", *Numerische Mathematik*, vol. 122, no. 3, pp. 469–488, Nov. 2012.

[107] M. Espig and W. Hackbusch, "A regularized newton method for the efficient approximation of tensors represented in the canonical tensor format", *Numerische Mathematik*, vol. 122, no. 3, pp. 489–525, May 2012. DOI: 10.1007/s00211-012-0465-9.

[108] Facebook Research, "Tensor comprehensions: A domain specific language to express machine learning workloads.", URL: https://facebookresearch.github.io/TensorComprehensions/ (visited Mar. 21, 2018).

[109] ——, "Zstandard — Real-time data compression algorithm", URL: http://facebook.github.io/zstd/ (visited Mar. 20, 2018).

[110] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization", *Inverse Problems*, vol. 27, no. 2, p. 025 010, Jan. 2011.

[111] J. Garcke, "Sparse grids in a nutshell", *Sparse Grids and Applications*, pp. 57–80, 2012. DOI: 10.1007/978-3-642-31703-3_3.

[112] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points — Online stochastic gradient for tensor decomposition", in *Proceedings of The 28th Conference on Learning Theory*, 2015, pp. 797–842.

[113] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent", in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2011, pp. 69–77. DOI: 10.1145/2020408.2020426.

[114] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 6, pp. 643–660, Jun. 2001. DOI: 10.1109/34.927464.

[115] R. S. Ghiass and E. Fatemizadeh, "Multi-view face detection and recognition under varying illumination conditions by designing an illumination effect cancelling filter", in *New Trends in Audio and Video / Signal Processing Algorithms, Architectures, Arrangements, and Applications SPA 2008*, Sep. 2008, pp. 27–32.

[116] N. Gillis, "The why and how of nonnegative matrix factorization", in *Regularization, Optimization, Kernels, and Support Vector Machines*, ser. Machine Learning and Pattern Recognition, J. A. K. Suykens, M. Signoretto, and A. Argyriou, Eds., Chapman & Hall / CRC, 2014, ch. 12, pp. 257–291.

[117] I. Gohberg and V. Olshevsky, "Complexity of multiplication with vectors for structured matrices", *Linear Algebra Appl.*, vol. 202, pp. 163–192, Apr. 1994. DOI: 10.1016/0024-3795(94)90189-9.

[118] ——, "Fast algorithms with preprocessing for matrix-vector multiplication problems", *Journal of Complexity*, vol. 10, no. 4, pp. 411–427, Dec. 1994. DOI: 10.1006/jcom.1994.1021.

[119] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 2012.

[120] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, "A theory of pseudoskeleton approximations", *Linear Algebra Appl.*, vol. 261, no. 1, pp. 1–21, 1997. DOI: 10.1016/S0024-3795(96)00301-1.

[121] S. A. Goreinov, N. L. Zamarashkin, and E. E. Tyrtyshnikov, "Pseudo-skeleton approximations by matrices of maximal volume", *Mathematical Notes*, vol. 62, no. 4, pp. 515–519, 1997. DOI: 10.1007/BF02358985.

[122] U. Grafe, B. Böttger, J. Tiaden, and S. G. Fries, "Coupling of multicomponent thermodynamic databases to a phase field model: Application to solidification and solid state transformations of superalloys", *Scripta Materialia*, vol. 42, no. 12, pp. 1179–1186, Jun. 2000. DOI: 10.1016/s1359-6462(00)00355-9.

[123] L. Grasedyck, "Polynomial approximation in hierarchical Tucker format by vector tensorization", in, Preprint 43, DFG/SPP1324, RWTH Aachen, Apr. 2010.

[124] L. Grasedyck, "Hierarchical singular value decomposition of tensors", *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 2029–2054, Jan. 2010. DOI: 10.1137/090764189.

[125] L. Grasedyck and S. Krämer, *Stable ALS approximation in the TT-format for rank-adaptive tensor completion*. arXiv: 1701.08045.

[126] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques", *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, Aug. 2013. DOI: 10.1002/gamm.201310004.

[127] K. Grönhagen, J. Ågren, and M. Odén, "Phase-field modelling of spinodal decomposition in TiAlN including the effect of metal vacancies", *Scripta Materialia*, vol. 95, pp. 42–45, Jan. 2015. DOI: 10.1016/j.scriptamat.2014.09.027.

[128] Y. Guan and N. Moelans, "Influence of the solubility range of intermetallic compounds on their growth behavior in hetero-junctions", *Journal of Alloys and Compounds*, vol. 635, pp. 289–299, Jun. 2015. DOI: 10.1016/j.jallcom.2015.02.028.

[129] W. Hackbusch, "Tensor spaces and numerical tensor calculus", *Springer Series in Computational Mathematics*, 2012. DOI: 10.1007/978-3-642-28027-6.

[130] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Hierarchical Kronecker tensor-product approximations", *J. Numer. Math.*, vol. 13, no. 2, Jan. 2005. DOI: 10.1515/1569395054012767.

[131] W. Hackbusch, D. Kressner, and A. Uschmajew, "Perturbation of higher-order singular values", in, INS Preprint No. 1616, 2016.

[132] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation", *J. Fourier Anal. Appl.*, vol. 15, no. 5, pp. 706–722, Oct. 2009. DOI: 10.1007/s00041-009-9094-9.

[133] W. Hackbusch and A. Uschmajew, "On the interconnection between the higher-order singular values", *Numerische Mathematik*, vol. 135, no. 3, pp. 875–894, Mar. 2017. DOI: 10.1007/s00211-016-0819-9.

[134] W. Hackbusch, "Hierarchical matrices: Algorithms and analysis", *Springer Series in Computational Mathematics*, 2015. DOI: 10.1007/978-3-662-47324-5.

[135] 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. DOI: 10.1137/090771806.

[136] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations", *Optimization Methods and Software*, vol. 30, no. 5, pp. 1002–1029, Apr. 2015. DOI: 10.1080/10556788.2015.1009977.

[137] N. Hao, M. E. Kilmer, K. Braman, and R. C. Hoover, "Facial recognition using tensor-tensor decompositions", *SIAM Journal on Imaging Sciences*, vol. 6, no. 1, pp. 437–463, Jan. 2013. DOI: 10.1137/110842570.

[138] R. A. Harshman and M. E. Lundy, "PARAFAC: Parallel factor analysis", *Computational Statistics & Data Analysis*, vol. 18, no. 1, pp. 390–72, Aug. 1994. DOI: 10.1016/0167-9473(94)90132-5.

[139] R. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis", *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[140] X. He, D. Cai, and P. Niyogi, "Tensor subspace analysis", in *Advances in Neural Information Processing Systems*, Y. Weiss, B. Schölkopf, and J. C. Platt, Eds., vol. 18, MIT Press, 2006, pp. 499–506.

[141] J. Heulens, B. Blanpain, and N. Moelans, "A phase field model for isothermal crystallization of oxide melts", *Acta Materialia*, vol. 59, no. 5, pp. 2156–2165, Mar. 2011. DOI: 10.1016/j.actamat.2010.12.016.

[142] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products", *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, Apr. 1927. DOI: 10.1002/sapm192761164.

[143] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the Tensor Train format", *SIAM J. Sci. Comput.*, vol. 34, no. 2, A683–A713, Jan. 2012. DOI: 10.1137/100818893.

[144] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, Cambridge, 2013.

[145] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient algorithmic framework for constrained matrix and tensor factorization", *IEEE Trans. Signal Process.*, vol. 64, no. 19, pp. 5052–5065, 2016. DOI: 10.1109/TSP.2016.2576427.

[146] K. Z. Ibrahim, S. W. Williams, E. Epifanovsky, and A. I. Krylov, "Analysis and tuning of libtensor framework on multicore architectures", in *2014 21st International Conference on High Performance Computing (HiPC)*, Dec. 2014, pp. 1–10. DOI: 10.1109/HiPC.2014.7116881.

[147] M. Ishteva, P.-A. Absil, S. Van Huffel, and L. De Lathauwer, "Best low multilinear rank approximation of higher-order tensors, based on the Riemannian trust-region scheme", *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 1, pp. 115–135, Jan. 2011. DOI: 10.1137/090764827.

[148] M. Ishteva, L. De Lathauwer, P.-A. Absil, and S. Van Huffel, "The best rank-(R_1, R_2, R_3) approximation of tensors by means of a geometric Newton method", *AIP Conference Proceedings*, vol. 1048, no. 1, pp. 274–277, 2008. DOI: 10.1063/1.2990911.

[149] B. Jeon, I. Jeon, L. Sael, and U. Kang, "Scout: Scalable coupled matrix-tensor factorization - algorithm and discoveries", *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, May 2016. DOI: 10.1109/icde.2016.7498292.

[150] I. Jeon, E. E. Papalexakis, C. Faloutsos, L. Sael, and U. Kang, "Mining billion-scale tensors: Algorithms and discoveries", *The VLDB Journal*, vol. 25, no. 4, pp. 519–544, Mar. 2016. DOI: 10.1007/s00778-016-0427-4.

[151] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "HaTen2: Billion-scale tensor decompositions", in *2015 IEEE 31st International Conference on Data Engineering*, IEEE, Apr. 2015, pp. 1047–1058. DOI: 10.1109/icde.2015.7113355.

[152] C. R. Johnson, H. Šmigoc, and D. Yang, "Solution theory for systems of bilinear equations", *Linear and Multilinear Algebra*, vol. 62, no. 12, pp. 1553–1566, 2014.

[153] A. M. Jokisaari and K. Thornton, "General method for incorporating CALPHAD free energies of mixing into phase field models: Application to the α-zirconium/δ-hydride system", *Calphad*, vol. 51, pp. 334–343, Dec. 2015. DOI: 10.1016/j.calphad.2015.10.011.

[154] A. Jokisaari, P. Voorhees, J. Guyer, J. Warren, and O. Heinonen, "Benchmark problems for numerical implementations of phase field models", *Computational Materials Science*, vol. 126, pp. 139–151, Jan. 2017. DOI: 10.1016/j.commatsci.2016.09.022.

[155] N. Jouppi, "Google supercharges machine learning tasks with tpu custom chip", URL: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html, May 2017.

[156] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling tensor analysis up by 100 times - algorithms and discoveries", *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, 2012. DOI: 10.1145/2339530.2339583.

[157] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format", *Parallel Comput.*, vol. 57, pp. 222–234, Sep. 2016. DOI: 10.1016/j.parco.2015.10.002.

[158] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*, ACM, 2015, 77:1–77:11. DOI: 10.1145/2807591.2807624.

[159] ——, "High performance parallel algorithms for the Tucker decomposition of sparse tensors", in *2016 45th International Conference on Parallel Processing (ICPP)*, IEEE, Aug. 2016, pp. 103–112. DOI: 10.1109/icpp.2016.19.

[160] C. Kelley, *Iterative Methods for Optimization*. SIAM, 1999.

[161] B. N. Khoromskij, "Structured rank-$(r_1, \ldots, r_D)$ decomposition of function-related tensors in $\mathbb{R}^D$", *Comput. Methods Appl. Math.*, vol. 6, no. 2, 2006. DOI: 10.2478/cmam-2006-0010.

[162] ——, "$\mathcal{O}(d \log N)$-quantics approximation of $N - d$ tensors in high-dimensional numerical modeling", *Constr. Approx.*, vol. 34, no. 2, pp. 257–280, Oct. 2011. DOI: 10.1007/s00365-011-9131-1.

[163] B. N. Khoromskij and V. Khoromskaia, "Low rank Tucker-type tensor approximation to classical potentials", *Central European Journal of Mathematics*, vol. 5, no. 3, pp. 523–550, Sep. 2007. DOI: 10.2478/s11533-007-0018-0.

[164] ——, "Multigrid accelerated tensor approximation of function related multidimensional arrays", *SIAM J. Sci. Comput.*, vol. 31, no. 4, pp. 3002–3026, Jan. 2009. DOI: 10.1137/080730408.

[165] B. N. Khoromskij, "Tensors-structured numerical methods in scientific computing: Survey on recent advances", *Chemometr. Intell. Lab.*, vol. 110, no. 1, pp. 1–19, Jan. 2012. DOI: 10.1016/j.chemolab.2011.09.001.

[166] H. A. L. Kiers and R. A. Harshman, "Relating two proposed methods for speedup of algorithms for fitting two- and three-way principal component and related multilinear models", *Chemometr. Intell. Lab.*, vol. 36, no. 1, pp. 31–40, Feb. 1997. DOI: 10.1016/s0169-7439(96)00074-3.

[167] Y.-D. Kim, A. Cichocki, and S. Choi, "Nonnegative Tucker decomposition with alpha-divergence", *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, Mar. 2008. DOI: 10.1109/icassp.2008.4517988.

[168] T. Kitashima, "Coupling of the phase-field and calphad methods for predicting multicomponent, solid-state phase transformations", *Philosophical Magazine*, vol. 88, no. 11, pp. 1615–1637, Apr. 2008. DOI: 10.1080/14786430802243857.

[169] T. G. Kolda, "Orthogonal tensor decompositions", *SIAM J. Matrix Anal. Appl.*, vol. 23, no. 1, pp. 243–255, Jul. 2001. DOI: 10.1137/S0895479800368354.

[170] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Aug. 2009. DOI: 10.1137/07070111x.

[171] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining", *2008 Eighth IEEE International Conference on Data Mining*, Dec. 2008. DOI: 10.1109/icdm.2008.89.

[172] K. Konakli and B. Sudret, "Polynomial meta-models with canonical low-rank approximations: Numerical insights and comparison to sparse polynomial chaos expansions", *J. Comput. Phys.*, vol. 321, pp. 1144–1169, Sep. 2016. DOI: 10.1016/j.jcp.2016.06.005.

[173] T. Koyama, K. Hashimoto, and H. Onodera, "Phase-field simulation of phase transformation in Fe–Cu–Mn–Ni quaternary alloy", *MATERIALS TRANSACTIONS*, vol. 47, no. 11, pp. 2765–2772, 2006. DOI: 10.2320/matertrans.47.2765.

[174] T. Koyama and H. Onodera, "Computer simulation of phase decomposition in Fe–Cu–Mn–Ni quaternary alloy based on the phase-field method", *Materials Transactions*, vol. 46, no. 6, pp. 1187–1192, 2005. DOI: 10.2320/matertrans.46.1187.

[175] W. P. Krijnen, T. K. Dijkstra, and A. Stegeman, "On the non-existence of optimal solutions and the occurrence of "degeneracy" in the CANDECOMP/PARAFAC model", *Psychometrika*, vol. 73, no. 3, pp. 431–439, Jan. 2008. DOI: 10.1007/s11336-008-9056-1.

[176] P. M. Kroonenberg and J. de Leeuw, "Principal component analysis of three-mode data by means of alternating least squares algorithms", *Psychometrika*, vol. 45, no. 1, pp. 69–97, Mar. 1980. DOI: 10.1007/bf02293599.

[177] A. Kroupa, "Modelling of phase diagrams and thermodynamic properties using Calphad method — Development of thermodynamic databases", *Computational Materials Science*, vol. 66, pp. 3–13, Jan. 2013. DOI: 10.1016/j.commatsci.2012.02.003.

[178] A. Kroupa, A. T. Dinsdale, A. Watson, J. Vrestal, J. Vízdal, and A. Zemanova, "The development of the COST 531 lead-free solders thermodynamic database", *JOM*, vol. 59, no. 7, pp. 20–25, Jul. 2007. DOI: 10.1007/s11837-007-0084-6.

[179] 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. DOI: 10.1016/0024-3795(77)90069-6.

[180] B. Krzanich, "Data is the new oil in the future of automated driving", URL: https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-driving/ (visited Mar. 20, 2018), Nov. 2016.

[181] D. Lahat, T. Adalı, and C. Jutten, "Multimodal data fusion: An overview of methods, challenges, and prospects", *Proc. IEEE*, vol. 103, no. 9, pp. 1449–1477, Sep. 2015. DOI: 10.1109/jproc.2015.2460697.

[182] H. Larsson and L. Höglund, "A scheme for more efficient usage of CALPHAD data in simulations", *Calphad*, vol. 50, pp. 1–5, Sep. 2015. DOI: 10.1016/j.calphad.2015.04.007.

[183] 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. DOI: 10.1137/0614071.

[184] F. Li, B. Wu, L. Xu, C. Shi, and J. Shi, "A fast distributed stochastic gradient descent algorithm for matrix factorization", in *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2014, pp. 77–87.

[185] Q. Li, D. Schonfeld, and S. Friedland, "Generalized tensor compressive sensing", in *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, Jul. 2013, pp. 1–6. DOI: 10.1109/ICME.2013.6607560.

[186] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization", *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017. DOI: 10.1109/icassp.2017.7953287.

[187] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers", *IEEE Trans. Signal Process.*, vol. 63, no. 20, pp. 5450–5463, Oct. 2015. DOI: 10.1109/tsp.2015.2454476.

[188] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, Jan. 2013. DOI: 10.1109/tpami.2012.39.

[189] S. Liu and G. Trenkler, "Hadamard, Khatri–Rao, Kronecker and other matrix products", *Int. J. Inf. Syst. Sci*, vol. 4, no. 1, pp. 160–177, 2008.

[190] X. Liu and N. D. Sidiropoulos, "Cramér–Rao lower bounds for low-rank decomposition of multidimensional arrays", *IEEE Trans. Signal Process.*, vol. 49, no. 9, pp. 2074–2086, Sep. 2001. DOI: 10.1109/78.942635.

[191] X. Liu, N. D. Sidiropoulos, and T. Jiang, "Multidimensional harmonic retrieval with applications in MIMO wireless channel sounding", in *Space-Time Processing for MIMO Communications*, A. Gershman and N. Sidiropoulos, Eds., John Wiley & Sons, Ltd, 2005.

[192] L. Ljung, *System identification: Theory for the user*, second edition. Prentice hall, 1999.

[193] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data", *Pattern Recognition*, vol. 44, no. 7, pp. 1540–1551, 2011.

[194] H. L. Lukas, S. G. Fries, and B. Sundman, *Computational thermodynamics: The Calphad method*. Cambridge university press, 2007.

[195] M. Mahoney, "Randomized algorithms for matrices and data", *Foundations and Trends in Machine Learning*, vol. 3, no. 2, pp. 123–224, 2010. DOI: 10.1561/2200000035.

[196] M. Mahoney, M. Maggioni, and P. Drineas, "Tensor-CUR decompositions for tensor-based data", *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 957–987, 2008. DOI: 10.1137/060665336.

[197] M. Mardani, G. Mateos, and G. B. Giannakis, "Subspace learning and imputation for streaming big data matrices and tensors", *IEEE Trans. Signal Process.*, vol. 63, no. 10, pp. 2663–2677, May 2015. DOI: 10.1109/tsp.2015.2417491.

[198] D. Matthews, *High-performance tensor contraction without transposition*, Jul. 2017. arXiv: 1607.00291.

[199] N. Moelans, B. Blanpain, and P. Wollants, "An introduction to phase-field modeling of microstructure evolution", *Calphad*, vol. 32, no. 2, pp. 268–294, Jun. 2008. DOI: 10.1016/j.calphad.2007.11.003.

[200] M. J. Mohlenkamp, "Musings on multilinear fitting", *Linear Algebra Appl.*, vol. 438, no. 2, pp. 834–852, Jan. 2013. DOI: 10.1016/j.laa.2011.04.019.

[201] M. Moonen and B. De Moor, *SVD and signal processing, III: Algorithms, architectures and applications*. Elsevier, 1995.

[202] Y.-M. Muggianu, M. Gambino, and J.-P. Bros, "Enthalpies of formation of liquid alloys bismuth-gallium-tin at 723k. Choice of an analytical representation of integral and partial thermodynamic functions of mixing for this ternary-system", *Journal de Chimie Physique et de Physico-Chimie Biologique*, vol. 72, no. 1, pp. 83–88, 1975. DOI: 10.1051/jcp/1975720083.

[203] T. Müller, K. Kruppa, G. Lichtenberg, and N. Réhault, "Fault detection with qualitative models reduced by tensor decomposition methods", *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 416–421, 2015. DOI: 10.1016/j.ifacol.2015.09.562.

[204] NASA, "MODIS web - Specifications", URL: https://modis.gsfc.nasa.gov/about/specifications.php (visited March 22, 2018).

[205] A. Nedić and D. Bertsekas, "Convergence rate of incremental subgradient algorithms", in *Stochastic optimization: algorithms and applications*, Springer, 2001, pp. 223–264.

[206] P. Nelson, "Just one autonomous car will use 4,000 GB of data/day", URL: https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html (visited Mar. 20, 2018), Dec. 2016.

[207] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems", *SIAM J. Optim.*, vol. 22, no. 2, pp. 341–362, 2012. DOI: 10.1137/100802001.

[208] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor", *IEEE Trans. Signal Process.*, vol. 57, no. 6, pp. 2299–2310, Jun. 2009. DOI: 10.1109/TSP.2009.2016885.

[209] J. Nocedal and S. J. Wright, *Numerical Optimization*, Second edition. New York: Springer, 2006.

[210] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states", *Ann. Physics*, vol. 349, pp. 117–158, 2014. DOI: 10.1016/j.aop.2014.06.013.

[211] I. V. Oseledets, "Tensor-train decomposition", *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Sep. 2011. DOI: 10.1137/090752286.

[212] I. V. Oseledets and S. Dolgov, "Solution of linear systems and matrix inversion in the TT-format", *SIAM J. Sci. Comput.*, vol. 34, no. 5, A2718–A2739, Jan. 2012. DOI: 10.1137/110833142.

[213] I. V. Oseledets, S. Dolgov, V. Kazeev, D. V. Savostyanov, O. Lebedeva, P. Zhlobich, T. Mach, and L. Song, *TT-Toolbox*, Available online at https://github.com/oseledets/TT-Toolbox.

[214] I. V. Oseledets, D. V. Savostianov, and E. E. Tyrtyshnikov, "Tucker dimensionality reduction of three-dimensional arrays in linear time", *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 939–956, 2008. DOI: 10.1137/060655894.

[215] I. V. Oseledets, D. V. Savostyanov, and E. E. Tyrtyshnikov, "Linear algebra for tensor problems", *Computing*, vol. 85, no. 3, pp. 169–188, Jun. 2009. DOI: 10.1007/s00607-009-0047-6.

[216] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions", *SIAM J. Sci. Comput.*, vol. 31, no. 5, pp. 3744–3759, Jan. 2009. DOI: 10.1137/090748330.

[217] I. V. Oseledets and E. E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays", *Linear Algebra Appl.*, vol. 432, no. 1, pp. 70–88, Jan. 2010. DOI: 10.1016/j.laa.2009.07.024.

[218] P. Paatero, "A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis", *Chemometr. Intell. Lab.*, vol. 38, no. 2, pp. 223–242, Oct. 1997. DOI: 10.1016/s0169-7439(97)00031-2.

[219] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "ParCube: Sparse parallelizable CANDECOMP-PARAFAC tensor decomposition", *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 1, pp. 1–25, Jul. 2015. DOI: 10.1145/2729980.

[220] 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, Jun. 2016. DOI: 10.1002/sam.11315.

[221] J. M. Papy, L. De Lathauwer, and S. Van Huffel, "Exponential data fitting using multilinear algebra: The single-channel and multi-channel case", *Numer. Linear Algebra Appl.*, vol. 12, no. 8, pp. 809–826, 2005. DOI: 10.1002/nla.453.

[222] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Version 2012-11-15, Nov. 2012.

[223] A.-H. Phan and A. Cichocki, "PARAFAC algorithms for large-scale problems", *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, May 2011. DOI: 10.1016/j.neucom.2010.06.030.

[224] 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, Oct. 2013. DOI: 10.1109/tsp.2013.2269903.

[225] A.-H. Phan and A. Cichocki, "Fast and efficient algorithms for nonnegative Tucker decomposition", *Advances in Neural Networks - ISNN 2008*, pp. 772–782, 2008. DOI: 10.1007/978-3-540-87734-9_88.

[226] 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, Jan. 2013. DOI: 10.1137/100808034.

[227] Plato, *Republic*. 381 BC, Available online at http://www.perseus.tufts.edu/hopper/text?doc=Plat.+Rep.+7.514a&fromdoc=Perseus%3Atext%3A1999.01.0167.

[228] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: A novel method to accelerate PARAFAC", *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1128–1147, Jan. 2008. DOI: 10.1137/06065577.

[229] M. J. Reynolds, G. Beylkin, and A. Doostan, "Optimization via separated representations and the canonical tensor decomposition", *J. Comput. Phys.*, vol. 348, pp. 220–230, Nov. 2017. DOI: 10.1016/j.jcp.2017.07.012.

[230] M. J. Reynolds, A. Doostan, and G. Beylkin, "Randomized alternating least squares for canonical tensor decompositions: Application to a PDE with random data", *SIAM J. Sci. Comput.*, vol. 38, no. 5, A2634–A2664, Jan. 2016. DOI: 10.1137/15m1042802.

[231] H. Robbins and S. Monro, "A stochastic approximation method", *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951. DOI: 10.1214/aoms/1177729586.

[232] J.-P. Royer, N. Thirion-Moreau, and P. Comon, "Computing the polyadic decomposition of nonnegative third order tensors", *Signal Processing*, vol. 91, no. 9, pp. 2159–2171, Sep. 2011. DOI: 10.1016/j.sigpro.2011.03.006.

[233] E. Sanchez and B. R. Kowalski, "Generalized rank annihilation factor analysis", *Analytical Chemistry*, vol. 58, no. 2, pp. 496–499, 1986. DOI: 10.1021/ac00293a054.

[234] N. Saunders and A. P. Miodownik, *CALPHAD: Calculation of phase diagrams: A comprehensive guide*, ser. Pergamon materials series. Oxford; New York: Pergamon, 1998.

[235] D. V. Savostyanov, "Fast revealing of mode ranks of tensor in canonical form", *Numer. Math. Theor. Meth. Appl*, vol. 2, no. 4, pp. 439–444, 2009. DOI: 10.4208/nmtma.2009.m9006s.

[236] D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, "Fast truncation of mode ranks for bilinear tensor operations", *Numer. Linear Algebra Appl.*, vol. 19, no. 1, pp. 103–111, Feb. 2011. DOI: 10.1002/nla.765.

[237] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent", *Neural Computation*, vol. 14, no. 7, pp. 1723–1738, 2002. DOI: 10.1162/08997660260028683.

[238] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization", in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 436–443.

[239] D. Schwen, L. Aagesen, J. Peterson, and M. Tonks, "Rapid multiphase-field model development using a modular free energy based approach with automatic differentiation in MOOSE/MARMOT", *Computational Materials Science*, vol. 132, pp. 36–45, May 2017. DOI: 10.1016/j.commatsci.2017.02.017.

[240] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization", in *Data Mining (ICDM), 2014 IEEE International Conference on*, Dec. 2014, pp. 989–994. DOI: 10.1109/ICDM.2014.78.

[241] 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, May 2000. DOI: 10.1002/1099-128X(200005/06)14:3<229::AID-CEM587>3.0.CO;2-N.

[242] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, "Parallel factor analysis in sensor array processing", *IEEE Trans. Signal Process.*, vol. 48, no. 8, pp. 2377–2388, Aug. 2000. DOI: 10.1109/78.852018.

[243] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning", *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jul. 2017. DOI: 10.1109/TSP.2017.2690524.

[244] N. D. Sidiropoulos and A. Kyrillidis, "Multi-way compressed sensing for sparse low-rank tensors", *IEEE Signal Process. Lett.*, vol. 19, no. 11, pp. 757–760, Nov. 2012. DOI: 10.1109/lsp.2012.2210872.

[245] N. D. 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, Sep. 2014. DOI: 10.1109/msp.2014.2329196.

[246] M. Signoretto, Q. Tran Dinh, L. De Lathauwer, and J. A. K. Suykens, "Learning with tensors: A framework based on convex optimization and spectral regularization", *Machine Learning*, vol. 94, no. 3, pp. 303–351, May 2013. DOI: 10.1007/s10994-013-5366-3.

[247] M. Signoretto, R. Van de Plas, B. De Moor, and J. A. K. Suykens, "Tensor versus matrix completion: A comparison with application to spectral data", *IEEE Signal Process. Lett.*, vol. 18, no. 7, pp. 403–406, Jul. 2011. DOI: 10.1109/lsp.2011.2151856.

[248] 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, Sep. 2008. DOI: 10.1137/06066518X.

[249] U. Şimşekli, A. T. Cemgil, and B. Ermiş, "Learning mixed divergences in coupled matrix and tensor factorization models", *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015. DOI: 10.1109/icassp.2015.7178345.

[250] A. K. Smilde, R. Bro, P. Geladi, and J. Wiley, *Multi-way analysis with applications in the chemical sciences*. Wiley Chichester, UK, 2004.

[251] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication", in *2015 IEEE International Parallel and Distributed Processing Symposium*, May 2015, pp. 61–70. DOI: 10.1109/IPDPS.2015.27.

[252] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor", in *Proceedings of the 5th Workshop on Irregular Applications Architectures and Algorithms - IA3 '15*, ACM, 2015, 5:1–5:7. DOI: 10.1145/2833179.2833183.

[253] ——, "A medium-grained algorithm for sparse tensor factorization", in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2016, pp. 902–911. DOI: 10.1109/ipdps.2016.113.

[254] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, Jul. 2009. DOI: 10.1016/j.ipm.2009.03.002.

[255] E. Solomonik, D. Matthews, J. Hammond, and J. Demmel, "Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions", in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, May 2013, pp. 813–824. DOI: 10.1109/IPDPS.2013.112.

[256] A. J. Sommese and C. W. Wampler II, *The numerical solution of systems of polynomials arising in engineering and science*. World Scientific, Hackensack, NJ, 2005.

[257] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization", *Comput. Optim. Appl.*, vol. 63, no. 1, pp. 121–142, May 2015. DOI: 10.1007/s10589-015-9761-5.

[258] L. Sorber, M. Van Barel, and L. De Lathauwer, "Unconstrained optimization of real functions in complex variables", *SIAM J. Optim.*, vol. 22, no. 3, pp. 879–898, Jan. 2012. DOI: 10.1137/110832124.

[259] ——, *Complex optimization toolbox v1.0*, Available online at https://www.esat.kuleuven.be/sista/cot/, Feb. 2013.

[260] ——, "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, Apr. 2013. DOI: 10.1137/120868323.

[261] ——, *Tensorlab v2.0*, Available online at https://www.tensorlab.net, Jan. 2014.

[262] ——, "Structured data fusion", *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, Jun. 2015. DOI: 10.1109/jstsp.2015.2400415.

[263] M. Sørensen and L. De Lathauwer, "Multidimensional harmonic retrieval via coupled canonical polyadic decomposition — Part I: Model and identifiability", *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 517–527, Jan. 2017. DOI: 10.1109/TSP.2016.2614796.

[264] M. Sørensen and L. De Lathauwer, "Tensor decompositions with block-Toeplitz structure and applications in signal processing", *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, Nov. 2011. DOI: 10.1109/acssc.2011.6190040.

[265] ——, "Blind signal separation via tensor decomposition with Vandermonde factor: Canonical polyadic decomposition", *IEEE Trans. Signal Process.*, vol. 61, no. 22, pp. 5507–5519, Nov. 2013. DOI: 10.1109/tsp.2013.2276416.

[266] ——, "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, Jan. 2015. DOI: 10.1137/140956853.

[267] ——, "Fiber sampling approach to canonical polyadic decomposition and tensor completion", Technical Report 15-151, ESAT-STADIUS, KU Leuven, Belgium, 2015.

[268] ——, "Multiple invariance ESPRIT for nonuniform linear arrays: A coupled canonical polyadic decomposition approach", *IEEE Trans. Signal Process.*, vol. 64, no. 14, pp. 3693–3704, Jul. 2016. DOI: 10.1109/tsp.2016.2551686.

[269] ——, "Multidimensional harmonic retrieval via coupled canonical polyadic decomposition — Part II: Algorithm and multirate sampling", *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 528–539, Jan. 2017. DOI: 10.1109/tsp.2016.2614797.

[270] M. Sørensen, L. De Lathauwer, P. Comon, S. Icart, and L. Deneire, "Canonical polyadic decomposition with a columnwise orthonormal factor matrix", *SIAM J. Matrix Anal. Appl.*, vol. 33, no. 4, pp. 1190–1213, Jan. 2012. DOI: 10.1137/110830034.

[271] 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, Jan. 2015. DOI: 10.1137/140956865.

[272] 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, Aug. 2017. DOI: 10.1109/tsp.2017.2706183.

[273] P. Springer and P. Bientinesi, *Design of a high-performance GEMM-like tensor-tensor multiplication*. arXiv: 1607.00145.

[274] 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. 3, 11:1–11:37, Oct. 2008.

[275] *Thermo-Calc software COST 531 database version 3*, www.thermocalc.com. Accessed Jul 2016.

[276] P. Tichavský, A.-H. Phan, and Z. Koldovský, "Cramér–Rao-induced bounds for CANDECOMP/PARAFAC tensor decomposition", *IEEE Trans. Signal Process.*, vol. 61, no. 8, pp. 1986–1997, Apr. 2013. DOI: 10.1109/TSP.2013.2245660.

[277] K. Tiels, M. Schoukens, and J. Schoukens, "Generation of initial estimates for Wiener-Hammerstein models via basis function expansions", *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 481–486, Aug. 2014. DOI: 10.3182/20140824-6-ZA-1003.02292.

[278] 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, Apr. 2006. DOI: 10.1016/j.csda.2004.11.013.

[279] G. Tomasi and R. Bro, "Parafac and missing values", *Chemometr. Intell. Lab.*, vol. 75, no. 2, pp. 163–180, Feb. 2005. DOI: 10.1016/j.chemolab.2004.07.003.

[280] M. R. Tonks, D. Gaston, P. C. Millett, D. Andrs, and P. Talbot, "An object-oriented finite element framework for multiphysics phase field simulations", *Computational Materials Science*, vol. 51, no. 1, pp. 20–29, Jan. 2012. DOI: 10.1016/j.commatsci.2011.07.028.

[281] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*. SIAM, Jan. 1997.

[282] J. Treichler and B. Agee, "A new approach to multipath correction of constant modulus signals", *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, no. 2, pp. 459–472, Apr. 1983. DOI: 10.1109/TASSP.1983.1164062.

[283] S. Tu, R. Boczar, M. Simchowitz, M. Soltanolkotabi, and B. Recht, "Low-rank solutions of linear matrix equations via Procrustes flow", in *Proceedings of the International Conference on Machine Learning (ICML 2016, New York, USA)*, 2016.

[284] L. R. Tucker, "Some mathematical notes on three-mode factor analysis", *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep. 1966. DOI: 10.1007/bf02289464.

[285] M. Turk and A. Pentland, "Eigenfaces for recognition", *J. Cogn. Neurosci.*, vol. 3, no. 1, pp. 71–86, Jan. 1991. DOI: 10.1162/jocn.1991.3.1.71.

[286] E. E. Tyrtyshnikov, "Incomplete cross approximation in the mosaic-skeleton method", *Computing*, vol. 64, pp. 367–380, 2000. DOI: 10.1007/s006070070031.

[287] A. Uschmajew, "Local convergence of the alternating least squares algorithm for canonical tensor approximation", *SIAM J. Matrix Anal. Appl.*, vol. 33, no. 2, pp. 639–652, Jan. 2012. DOI: 10.1137/110843587.

[288] S. Van Eyndhoven, B. Hunyadi, L. De Lathauwer, and S. Van Huffel, "Flexible fusion of electroencephalography and functional magnetic resonance imaging: Revealing neural-hemodynamic coupling through structured matrix-tensor factorization", *2017 25th European Signal Processing Conference (EUSIPCO)*, Aug. 2017. DOI: 10.23919/eusipco.2017.8081162.

[289] S. Van Huffel, H. Chen, C. Decanniere, and P. Vanhecke, "Algorithm for time-domain NMR data fitting based on total least squares", *J. Magn. Reson. A*, vol. 110, no. 2, pp. 228–237, 1994. DOI: 10.1006/jmra.1994.1209.

[290] I. Van Mechelen and A. K. Smilde, "A generic linked-mode decomposition model for data fusion", *Chemometr. Intell. Lab.*, vol. 104, no. 1, pp. 83–94, Nov. 2010. DOI: 10.1016/j.chemolab.2010.04.012.

[291] 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. DOI: 10.23919/EUSIPCO.2017.8081290.

[292] N. Vannieuwenhoven, "Condition numbers for the tensor rank decomposition", *Linear Algebra Appl.*, vol. 535, pp. 35–86, 2017. DOI: 10.1016/j.laa.2017.08.014.

[293] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril, "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, Jan. 2015. DOI: 10.1137/14097968x.

[294] 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, A1027–A1052, Jan. 2012. DOI: 10.1137/110836067.

[295] M. A. O. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces", in *Proceedings of the European Conference on Computer Vision (ECCV '02, Copenhagen, Denmark)*, May 2002, pp. 447–460.

[296] ——, "Multilinear image analysis for facial recognition", in *Object recognition supported by user interaction for service robots*, vol. 2, Aug. 2002, pp. 511–514.

[297] A.-J. van der Veen, "Algebraic methods for deterministic blind beamforming", *Proc. IEEE*, vol. 86, no. 10, pp. 1987–2008, Oct. 1998. DOI: 10.1109/5.720249.

[298] A.-J. van der Veen and A. Paulraj, "An analytical constant modulus algorithm", *IEEE Trans. Signal Process.*, vol. 44, no. 5, pp. 1136–115, May 1996. DOI: 10.1109/78.502327.

[299] A. Vergara, J. Fonollosa, J. Mahiques, M. Trincavelli, N. Rulkov, and R. Huerta, "On the performance of gas sensor arrays in open sampling systems using inhibitory support vector machines", *Sens. Actuators B Chem.*, vol. 185, pp. 462–477, 2013. DOI: 10.1016/j.snb.2013.05.027.

[300] 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, no. 2, pp. 284–295, Mar. 2016. DOI: 10.1109/JSTSP.2015.2503260.

[301] ——, "Numerical optimization based algorithms for data fusion", Technical Report 18-11, ESAT-STADIUS, KU Leuven, Belgium. (Accepted)., 2018.

[302] N. Vervliet, O. Debals, and L. De Lathauwer, "Canonical polyadic decomposition of incomplete tensors with linearly constrained factors", Technical Report 16–172, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017.

[303] ——, "Exploiting efficient representations in tensor decompositions", Technical Report 16–174, ESAT-STADIUS, KU Leuven, Belgium, Oct. 2017.

[304] 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, Sep. 2014. DOI: 10.1109/MSP.2014.2329429.

[305] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, *Tensorlab 3.0*, Available online at https://www.tensorlab.net, Mar. 2016.

[306] N. Vervliet, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization", in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov. 2016, pp. 1733–1738. DOI: 10.1109/ACSSC.2016.7869679.

[307] X. T. Vu, S. Maire, C. Chaux, and N. Thirion-Moreau, "A new stochastic optimization algorithm to decompose large nonnegative tensors", *IEEE Signal Process. Lett.*, vol. 22, no. 10, pp. 1713–1717, Oct. 2015. DOI: 10.1109/LSP.2015.2427456.

[308] B. Widrow and S. Stearns, *Adaptive signal processing*, 1st. Englewood Cliffs, NJ: Prentice Hall, 1985.

[309] Z. Yawen, D. Guangjun, and X. Zhixiang, "Hyperspectral image tensor feature extraction based on fusion of multiple spectral-spatial features", in *Proceedings of the 2016 International Conference on Intelligent Information Processing - ICIIP '16*, ACM, 2016, 43:1–43:8. DOI: 10.1145/3028842.3028885.

[310] K. Y. Yılmaz, A. T. Cemgil, and U. ŞimŞekli, "Generalised coupled tensor factorisation", in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2011, pp. 2151–2159.

[311] T. Yokota, R. Zdunek, A. Cichocki, and Y. Yamashita, "Smooth nonnegative matrix and tensor factorizations for robust multi-way data analysis", *Signal Processing*, vol. 113, pp. 234–249, Aug. 2015. DOI: 10.1016/j.sigpro.2015.02.003.

[312] A. S. Zamzam, V. N. Ioannidis, and N. D. Sidiropoulos, "Coupled graph tensor factorization", *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov. 2016. DOI: 10.1109/acssc.2016.7869683.

[313] V. Zarzoso and P. Comon, "Optimal step-size constant modulus algorithm", *IEEE Trans. Commun.*, vol. 56, no. 1, Jan. 2008. DOI: 10.1109/TCOMM.2008.050484.

[314] 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, Sep. 2015. DOI: 10.1109/tpami.2015.2392756.

[315] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey", *ACM Computing Surveys*, vol. 35, no. 4, pp. 399–458, Dec. 2003. DOI: 10.1145/954339.954342.

[316] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories", *Proceedings of the 18th international conference on World wide web - WWW '09*, 2009. DOI: 10.1145/1526709.1526816.

[317] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of big tensors with low multilinear rank*, 2014. arXiv: 1412.1885.

[318] G. Zhou, A. Cichocki, Q. Zhao, and S. Xie, "Nonnegative matrix and tensor factorizations: An algorithmic perspective", *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 54–65, May 2014. DOI: 10.1109/msp.2014.2298891.

[319] ——, "Efficient nonnegative Tucker decompositions: Algorithms and uniqueness", *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 4990–5003, Dec. 2015. DOI: 10.1109/tip.2015.2478396.

[320] J. Zhu, Z. Liu, V. Vaithyanathan, and L. Chen, "Linking phase-field model to CALPHAD: Application to precipitate shape evolution in Ni-base alloys", *Scripta Materialia*, vol. 46, no. 5, pp. 401–406, Mar. 2002. DOI: 10.1016/s1359-6462(02)00013-1.

List of publications

Journal papers

[1] Y. Coutinho, **N. Vervliet**, L. De Lathauwer, and N. Moelans, "Efficient use of CALPHAD based data in phase-field spinodal decomposition simulations for a quaternary system through decomposed thermodynamic tensor models", Technical Report 18–51, ESAT-STADIUS, KU Leuven, Belgium, 2018.

[2] F. Van Eeghem, O. Debals, **N. Vervliet**, and L. De Lathauwer, "Coupled and incomplete tensors in blind system identification", Technical Report 17–128, ESAT-STADIUS, KU Leuven, Belgium, 2018. Submitted to IEEE Transactions on Signal Processing.

[3] M. Boussé, **N. Vervliet**, I. Domanov, O. Debals, and L. De Lathauwer, "Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications", Technical Report 17-01, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017. Submitted to Numerical Linear Algebra with Applications.

[4] **N. Vervliet**, O. Debals, and L. De Lathauwer, "Canonical polyadic decomposition of incomplete tensors with linearly constrained factors", Technical Report 16–172, ESAT-STADIUS, KU Leuven, Belgium, Apr. 2017. Submitted to SIAM Journal on Scientific Computing.

[5] **N. Vervliet**, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in tensor decompositions", Technical Report 16–174, ESAT-STADIUS, KU Leuven, Belgium, Oct. 2017. Submitted to SIAM Journal on Scientific Computing.

[6] **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, no. 2, pp. 284–295, Mar. 2016. DOI: 10.1109/JSTSP.2015.2503260.

[7] **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, Sep. 2014. DOI: 10.1109/MSP.2014.2329429.

Book chapters

[1] **N. Vervliet** and L. De Lathauwer, "Numerical optimization based algorithms for data fusion", Technical Report 18-11, ESAT-STADIUS, KU Leuven, Belgium. (Accepted)., 2018.

Conference proceedings

[1] M. Boussé, G. Goovaerts, **N. Vervliet**, O. Debals, S. Van Huffel, and L. De Lathauwer, "Irregular heartbeat classification using Kronecker product equations", in *39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2017)*, Jul. 2017, pp. 438–441. DOI: 10.1109/EMBC.2017.8036856.

[2] M. Boussé, **N. Vervliet**, O. Debals, and L. De Lathauwer, "Face recognition as a Kronecker product equation", in *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec. 2017, pp. 276–280.

[3] M. Vandecapelle, M. Boussé, **N. Vervliet**, and L. De Lathauwer, "CPD updating using low-rank weights", Technical Report 17-164, ESAT-STADIUS, KU Leuven, Belgium. Accepted for publication in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017.

[4] 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. DOI: 10.23919/EUSIPCO.2017.8081290.

[5] X.-F. Gong, Q. H. Lin, O. Debals, **N. Vervliet**, and L. De Lathauwer, "Coupled rank-$(L_m, L_n, \cdot)$ block term decomposition by coupled block simultaneous generalized Schur decomposition", in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 2554–2558. DOI: 10.1109/ICASSP.2016.7472138.

[6] **N. Vervliet**, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization", in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov. 2016, pp. 1733–1738. DOI: 10.1109/ACSSC.2016.7869679.

Other publications

[1] O. Debals, F. Van Eeghem, **N. Vervliet**, and L. De Lathauwer, "Tensorlab demos — Release 3.0", Technical Report 16–68, ESAT-STADIUS, KU Leuven, Belgium, 2016.

[2] **N. Vervliet**, O. Debals, and L. De Lathauwer, "Nieuwste versie Tensorlab vereenvoudigt 'big data' analyse", *Nieuwsbrief KU Leuven Campus Kulak Kortrijk*, May 2016, May 17, 2016. URL: http://www.kuleuven-kulak.be/nl/nieuws/nieuwste-versie-tensorlab-vereenvoudigt-2018big-data2019-analyse.

[3] **N. Vervliet**, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, *Tensorlab 3.0*, Available online at https://www.tensorlab.net, Mar. 2016.



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
Nico.Vervliet@kuleuven.be
http://www.esat.kuleuven.be/stadius/