KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

# Projection methods for parametrized and multiparameter eigenvalue problems

**Koen Ruymbeek**

Supervisors:
Prof. dr. ir. Karl Meerbergen
Prof. dr. ir. Wim Michiels

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

April 2021

# Projection methods for parametrized and multiparameter eigenvalue problems

**Koen RUYMBEEK**

Examination committee:
Prof. dr. ir. Bob Puers, chair
Prof. dr. ir. Karl Meerbergen, supervisor
Prof. dr. ir. Wim Michiels, supervisor
Prof. dr. Raf Vandebril
Prof. dr. ir. Geert Lombaert
Prof. dr. Bor Plestenjak
(University of Ljubljana)
Prof. dr. Howard Elman
(University of Maryland)

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

April 2021

© 2021 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Koen Ruymbeek, Celestijnenlaan 200A box 2402, 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

What you read in the next chapters is the result of my four years of research in the NUMA group at the Computer Science department in Leuven. Although I started my Phd concerned about not knowing anyone in Leuven and only a little about the subject, I must say that I had a great time in Leuven. Therefore I want to thank in this preface some people that contributed to this in one way or another.

I still remember the moment I was in my room at my parents' home doubting about whether to stay at UAntwerpen or to go to the KU Leuven. In the end I decided to go to Leuven and I did not regret it. First of all I want to thank my supervisors Karl and Wim for the guidance through my Phd track, for their enthusiasm about what I was doing and for the patience when I was struggling with my work or when I was a bit too stubborn in following my own ideas. Ever since I started, I always felt that you had confidence in me as a researcher and that it would at the end turn out in something good. Certainly after my first half year, when my first project did not result in any research output, it helped me a lot that you kept faith in me.

Although I would like to thank all members of NUMA for the entertaining coffee breaks and the great time in a stimulating environment, I want to thank some people in particular. First of all it was a pleasure to work at office 01.184. Thank you Nick for the enjoyable talks in the office and for your enthusiasm about tensors, which in the end was of use in two of my papers. Thank you Bruno for a bit guiding me during the first days of my Phd. Thank you Dong for your "normalness". During the last one year and a half, I got totally new office mates. Thanks to Charlotte for running in the University Trail and thanks to Thijs and Simon for the many table tennis games (especially for the ones I won) and for the many nice chats in the office. Also the table tennis games with Dries, Bruno and Daan were memorable during the first years and were the perfect alternation if I was stuck in my research. Also thanks to the research

group of Wim for letting me participating in the weekly team-lunch and to Luca and Pieter in particular for participating in the Bike and Run-events from Leuven to Louvain-La-Neuve. Sorry that at end of the race I was so tired that I was not of any use anymore but it was nevertheless a great experience.

Also the people from the Systems group were of great help to me when there was something wrong with my computer. Sometimes it looked like magic when I saw you fixing it. The secretary and especially Margot was of great help for the arrangement of flights, train tickets and hotels. Outside building 200A but inside Leuven, there are people which were of great support for me. Thanks to all members of the football team with which I spend many Tuesday evenings on the football pitch. I remember evenings with only four people but still playing like we could win the World Cup. I really looked forward to it every week. Finally, thanks to the members of Club Run for the enjoyable runs during lunch breaks.

Het lijkt het soms voor mij dat er buiten mijn onderzoek niets anders bestaat, gelukkig zijn er nog mensen rondom mij die mij tonen dat er nog een heel wereld buiten dat bestaat. Dank u Tim en Nikolaj om mee te doen aan enkele Mathjams, om mee naar concerten te gaan en zoveel meer. Ook dank aan Team Legendary, mijn zaalvoetbalploeg die mij wekelijks een moment gaf om mij wat af te reageren. Als laatste maar zeker niet minder gemeend, wil ik mijn dankbaarheid uiten aan mijn dichte familie. Dankuwel mama, papa, Bert (en Laura en Louise) en Maureen om mij bij te staan met raad en daad en mij het leven zo aangenaam te maken hoewel jullie waarschijnlijk nog altijd niet weten met wat ik nu precies dagdagelijks bezig ben. Speciale dank aan mijn ouders om mij steeds te steunen en mij de vrijheid te geven om te doen wat ik graag doe. Jullie hebben mij van kindsbeen af de nodige kwaliteiten aangeleerd en jullie bijdrage in wat ik heb bereikt, valt niet te onderschatten.

Koen Ruymbeek
Sint-Niklaas, januari 2021

# Abstract

*If we knew what it was*
*we were doing, it would*
*not be called research, would it?*
*(Albert Einstein)*

Matrix eigenvalue problems are common in computational science and engineering. In this thesis we focus on generalisations of the eigenvalue problem, namely parametrized and multiparameter eigenvalue problems. The matrices involved are large, so calculating eigenvalues is computationally expensive. Therefore, we develop techniques that exploit the structure and properties of the underlying problem.

In each of the proposed methods in this thesis, a subspace is iteratively built and, instead of solving large eigenvalue problems, we repeatedly solve eigenvalue problems projected on a small subspace. As these projected eigenvalue problems are smaller, they are computationally tractable. The question then is how we can build this subspace as efficiently as possible. In each of the three problems considered in this thesis, the way in which they are built up is different.

The first two algorithms that we propose deal with parameterized eigenvalue problems. Here the matrix pencil depends on parameters and we want to determine a global approximation of an extreme eigenvalue (largest or smallest in absolute value or with largest real part) over the parameter space. We discretise the parameter space into a set and we approximate the wanted eigenvalue in all points of this set.

In the first algorithm we focus on matrix pencils whose matrices are symmetric and at least one of the two matrices is positive definite. The idea here is that every vector in the subspace that we build is computed from information from one point in the parameter space. This point is taken from a large set of samples in which the approximation of the eigenvalue is the worst according to

its residual. We can therefore say that we only use local information to build our subspace.

This changes in the second algorithm we propose. Here we also solve parametrized eigenvalue problems but now we work only with standard eigenvalue problems and the matrix does not need to be symmetric. We extend an existing Krylov algorithm in order to perform iterations for multiple parameter points simultaneously. The used subspace arises from information coming from all parameter points.

The last problem we deal with concerns multiparameter eigenvalue problems. These are equivalent to solving a potentially very large generalized eigenvalue problem. The proposed projection method is based on the Tensor-Train representation of the problem. The resulting subspace is represented in Tensor-Train format. In one iteration, we update one core of the tensor representing the desired eigenvectors, while leaving the other cores fixed.

# Beknopte samenvatting

*Als we wisten wat we*
*aan het doen waren, zou het toch*
*geen onderzoek worden genoemd?*
*(Albert Einstein)*

Matrix eigenwaarde problemen komen vaak voor in wetenschappelijk rekenen. In deze thesis focussen we ons op veralgemeningen van het eigenwaarde probleem, namelijk geparametriseerde en multiparameter eigenwaarde problemen. De betrokken matrices zijn groot, wat maakt dat eigenwaarden berekenen rekenkundig duur is. Daarom hebben we baat bij het maken van technieken die de eigenschappen van het onderliggend probleem ten volle benutten.

In elk van de methodes in deze thesis wordt er iteratief een deelruimte opgebouwd en lossen we in plaats van grote eigenwaarde problemen, de eigenwaarde problemen geprojecteerd op een kleine deelruimte op. Deze geprojecteerde eigenwaarde problemen zijn veel kleiner en aldus computationeel goedkoper. De vraag is dan hoe je deze deelruimte zo efficiënt mogelijk opstelt. In elk van de drie problemen die in deze thesis worden beschouwd, is de manier waarop deze worden opgesteld verschillend.

De eerste twee algoritmes die we voorstellen, handelen over geparametriseerde eigenwaarde problemen. Hier hangt de matrix/het matrix pencil af van parameters en willen we een globale benadering van een extreme eigenwaarde (grootste/kleinste in absolute waarde of met grootste reële deel) bepalen. We discretiseren de parameterruimte en we benaderen de gezochte eigenwaarde in alle punten van deze eindige verzameling.

In het eerste algoritme richten we ons specifiek op matrix pencils waarvan de matrices symmetrisch zijn en minstens één van de twee moet positief definiet zijn. Hier is het idee dat elke vector in de deelruimte die we opbouwen, berekent is aan de hand van informatie van één punt in de parameterruimte. Dit punt is het punt in dewelke de benadering van de eigenwaarde het slechtst is op dat

moment volgens het residu. We kunnen dus stellen dat we enkel lokale info gebruiken om onze deelruimte op te stellen.

Dit verandert in het tweede algoritme dat we voorstellen. Hier lossen we ook geparametriseerde eigenwaarde problemen op maar nu werken we enkel met standaard eigenwaarde problemen waarbij de matrix wel asymetrisch mag zijn. Hier nemen we een bestaande deelruimte-methode om eigenwaarden te bepalen en breiden we deze methode uit zodat we tegelijk de methode voor heel veel punten kunnen toepassen. De deelruimte die we gebruiken, vloeit voort uit informatie die van alle parameterpunten afkomstig is.

Het laatste probleem dat we behandelen is het oplossen van multiparameter eigenwaarde problemen. Dit soort problemen zijn equivalent met het oplossen van een potentieel heel groot gegeneraliseerd eigenwaardeprobleem. De deelruimte-methode die hiervoor gemaakt is, is gebaseerd op de Tensor-Trein representatie van het probleem. De deelruimte die hieruit voortvloeit wordt ook voorgesteld binnen het raamwerk van Tensor-Treinen. In een iteratie doen we een update van één core van de tensor die de eigenvectoren voorstelt, terwijl de andere cores vast worden gehouden.

# List of Symbols

| | |
|---|---|
| $\mathbb{N}$ | Natural numbers (zero included) |
| $\mathbb{R}$ | Real numbers |
| $\mathbb{R}_+$ | Non-negative real numbers |
| $\mathbb{C}$ | Complex numbers |
| | |
| $a, b, \ldots$ | Scalars |
| $\overline{a}$ | Complex conjugate of $a$ |
| $\mathbf{a}, \mathbf{b}, \ldots$ | Vectors |
| $\mathbf{A}, \mathbf{B}, \ldots$ | Matrices |
| $\mathbf{A}(i, j)$ | $(i, j)$th element of the matrix $\mathbf{A}$ |
| $(\mathbf{A}, \mathbf{B})$ | matrix pencil |
| $\mathcal{A}, \mathcal{B}, \ldots$ | Tensors (except $\mathcal{V}$ which denotes a subspace) |
| $\mathrm{Re}(a)$ | Real part of $a \in \mathbb{C}$ |
| $\mathrm{Im}(a)$ | Complex part of $a \in \mathbb{C}$ |
| | |
| $< ., . >$ | Euclidean inner product |
| $\|\mathbf{a}\|$ | Euclidean norm |
| $\|\mathbf{a}\|_\infty$ | $\infty$-norm |

| | |
|---|---|
| $\mathbf{A}^T$ | Transpose of $\mathbf{A}$ |
| $\mathbf{A}^*$ | Conjugate transpose of $\mathbf{A}$ |
| $\mathbf{A}^{-1}$ | Inverse of $\mathbf{A}$ |
| $\text{diag}(a_1, \ldots, a_n)$ | Diagonal matrix with $a_1, \ldots, a_n$ on the diagonal |
| $\det(\mathbf{A}))$ | Determinant of a matrix $\mathbf{A}$ |
| $\Lambda(\mathbf{A})$ | Spectrum of $\mathbf{A}$ |
| $\text{Ker}(\mathbf{A})$ | Kernel of $\mathbf{A}$ |
| $\text{Range}(\mathbf{A})$ | Range of the matrix $\mathbf{A}$ |
| | |
| $. \times .$ | Cartesian product |
| $. \otimes .$ | Kronecker product |
| $\cdot_H \cdot$ | Hadamard product |
| $(\mathbf{A}_1, \ldots, \mathbf{A}_d) . \mathbf{A}$ | Multilinear multiplication |
| | |
| $\mathcal{V}$ | Linear Subspace of $\mathbb{R}^n$ |
| $\mathbf{V}$ | Orthonormal matrix whose columns spans the subspace $\mathcal{V}$ |
| $\mathbf{X}^{\mathcal{V}}$ | Matrix such that $\mathbf{V}\mathbf{X}^{\mathcal{V}} = \mathbf{X}$ |
| $\text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_m)$ | Linear subspace spanned by the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ |
| | |
| $\mathbf{e}_m$ | The $m$th canonical vector, $m$th column of $\mathbf{I}$ |
| $\mathbf{1}_m$ | Vector with all ones of length $m$ |
| $\mathbf{I}, \mathbf{I}_m$ | Identity matrix of unspecified size, of size $m \times m$ |
| $\mathbf{0}_{m,n}$ | Zero matrix of size $m \times n$ |
| | |
| $\mathcal{O}(.)$ | Big O |

| | |
|---|---|
| $\lambda_1, \ldots, \lambda_n$ | Eigenvalues ordered following some rule. It will be clear from the text from which matrix or matrix pair these are the eigenvalues and which ordering is used |
| $\mathbf{x}_1, \ldots, \mathbf{x}_n$ | Eigenvector associated with $\lambda_i, i = 1, \ldots, n$ |
| $(\lambda_i, \mathbf{x}_i)$ | The $i$th eigenpair of a matrix or matrix pencil |
| $(\mu_1, \ldots, \mu_m)$ | Eigenvalue-tuple of a $m$EP |
| $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ | Eigenvector-tuple of a $m$EP |
| $\mathbf{X}_{\neq k}$ | Frame matrix where we excluded mode $k$ |

# Contents

**Abstract** iii

**Beknopte samenvatting** v

**List of Symbols** ix

**Contents** xi

**List of Figures** xv

**List of Tables** xix

**1 State of the art and objectives** 1
- 1.1 Introduction . . . 1
- 1.2 Eigenvalue problems . . . 2
  - 1.2.1 Applications of eigenvalue problems . . . 4
- 1.3 Parametrized eigenvalue problems . . . 6
  - 1.3.1 Applications of parametrized eigenvalue problems . . . 7
  - 1.3.2 State-of-the-art . . . 10
- 1.4 Multiparameter eigenvalue problem . . . 13
  - 1.4.1 Applications of multiparameter eigenvalue problems . . . 13
  - 1.4.2 State-of-the-art . . . 16
- 1.5 Contributions and overview . . . 17

**2 Preliminaries** 21
- 2.1 Subspaces and projection methods . . . 21
  - 2.1.1 Orthonormalisation . . . 23
  - 2.1.2 Krylov subspaces . . . 24
- 2.2 Tensors . . . 25
  - 2.2.1 Basic concepts and operations . . . 26
  - 2.2.2 Tensor-Train decomposition . . . 28

2.2.3 Operations . . . 31
2.2.4 Comparison with other decompositions . . . 34

**3 Calculating the minimal eigenvalue of symmetric parametrized matrix pairs 35**
3.1 Introduction . . . 35
3.2 Derivatives of eigenvalues and eigenvectors . . . 36
3.3 Algorithm . . . 41
3.3.1 Idea . . . 41
3.3.2 Upper bound of the error . . . 44
3.3.3 Algorithm . . . 46
3.3.4 Saturation assumption . . . 47
3.4 Numerical results . . . 48
3.5 Expected value of the minimum eigenvalue . . . 51
3.5.1 Numerical experiment . . . 52
3.6 Conclusion . . . 53

**4 Tensor-Krylov method for computing eigenvalues of parametrized matrices 55**
4.1 Introduction . . . 55
4.2 Arnoldi method . . . 57
4.2.1 Convergence of the Residual Arnoldi algorithm . . . 59
4.3 Tensor-Arnoldi for one parameter . . . 61
4.4 Error analysis . . . 63
4.5 More than one parameter . . . 67
4.5.1 Numerical Experiments . . . 71
4.6 Parametric inverted Residual Arnoldi method . . . 71
4.6.1 Numerical experiments . . . 73
4.7 Conclusion . . . 76

**5 Solving multiparameter eigenvalue problems via tensor-train representations 78**
5.1 Introduction . . . 78
5.2 Calculating eigenvalues of symmetric matrices as TT-operators 80
5.3 Tensor-algorithm for multiparameter eigenvalue problems . . . 83
5.3.1 $\mathbf{\Delta}$-matrices as TT-operators . . . 84
5.3.2 Size of the projected eigenvalue problem and the rank of $\mathbf{X}$ 85
5.3.3 Stopping criterion . . . 86
5.4 Selecting eigenvalues and convergence criterion . . . 86
5.4.1 Selection of eigenvalues . . . 87
5.4.2 Convergence criterion . . . 91
5.5 Numerical experiments . . . 92
5.5.1 Performance . . . 92

5.6 Dependency on parameters . . . . . 95
5.7 Conclusion . . . . . 98

**6 Conclusion** **102**
6.1 Future research directions . . . . . 103

**A Helmholtz equation with ellipsoidal coordinates** **105**
A.1 Ellipsoidal coordinates . . . . . 105
A.2 Solving Helmholtz equation . . . . . 107

**Bibliography** **111**

**Curriculum vitae** **119**

**Publications and talks** **121**

# List of Figures


1.1 The geometry of the underlying PDE in Example 2. The parameter $\omega_2$ determines the length of the domain. . . . . 9
1.2 Surface of $\lambda_1(\omega)$ in Example 2 . . . . . 11
1.3 Surfaces that represent $\lambda_1(\omega)$ and $\lambda_2(\omega)$ in Example 3 . . . . . 12

3.1 Figure corresponding to Example 5. (a) and (b): $\lambda_1(\omega)$ and $\lambda_2(\omega)$ with $\omega_1 = 0.5$ (left) and $\omega_1 = 0.05$ (right), (c) and (d): norm of $\frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_2}$ with $\omega_1 = 0.5$ (left) and $\omega_1 = 0.05$ (right) . . . . 40
3.2 Figure corresponding to Example 6. If both $\mathbf{w}_1$ and $\mathbf{w}_2$ do not depend much on $\omega_1$, then $\mathcal{V}_1 := \text{span}\{\mathbf{x}_1(\omega^i), i = 1, 2\} \approx \text{span}\{\mathbf{x}_i(\omega^1), i = 1, 2\} =: \mathcal{V}_2$ but $\mathcal{V}_2$ is less costly to calculate than $\mathcal{V}_1$. . . . . 44
3.3 The minimum eigenvalue $\lambda_1(\omega)$ over the parameter for the example from structural mechanics. We see the minimum eigenvalue does not smoothly depend on the variable. . . . . 50

4.1 Results of the PRA algorithm for different tolerances $\eta_X$ and $\eta_R$ with maximal subspace dimension 150 for Example 7. In (a) we plot the Frobenius norm of the residual, scaled with the number of sample points. In (b) the average first error $\mathbf{e}_1$ relative to the average residual, scaled with the norm of $\mathbf{A}_0$ is shown. As long as the value is below or around $10^{-3}$, the algorithm converges as if there were no error, see the criterium in (4.8). If the value is large ($> 1$) convergence is lost. The almost vertical lines in (b) are present just after iterations where we needed to restart the subspace. . . . . 66

4.2 Results of the PRA algorithm for different tolerances $\eta_X$ and $\eta_R$ with maximal subspace dimension 150 for Example 7. We compare the results when taking a constant low tolerance $\eta_X$ with the case that $\eta_X$ is relative to the residual. In (a) we plot the Frobenius norm of the residual, scaled with the number of samples. We see that in all four cases the algorithm converges. In (b), the average error $\mathbf{e}_1$ relative to the average residual, scaled with the norm of $\mathbf{A}_1$ is shown. As expected, we see that this error relative to the residual stays constant. . . . . 68
4.3 Results of the PRA algorithm with maximal subspace dimension 100 for Example 9. In (a), we see the convergence when taking $n_1 = n_2 = n_3 = 10$. We see that we need quite a large number of iterations but tests showed us that using Arnoldi for all parameter points separately needs approximately the same number of iterations. If we take the resulting subspace $\mathcal{V}$ and we use this as the initial subspace for $n_1 = n_2 = n_3 = 30$, we see that we just need a few iterations . . . . . 72
4.4 The spectral abscissa over the parameters for the two different cases in Example 10. We see in (a) that the spectral abscissa varies smoothly over the parameters, in (b) this is not the case anymore. The convergence results for the two different cases in Example 10 are plotted in (c) and (d). We see that in the two cases the algorithm converges within nine iterations. . . . . 75
4.5 In (a)-(b), The spectral abscissa over the parameters for the two different cases in Example 11. The convergence results for the two different cases in Example 11 can be seen in (c)-(d). . . . . 76

5.1 We plot the wanted eigenvalues together with the eigenvalues that we found for Example 12. In (a) the eigenvalues are well separated and we can find the first 10 eigenvalues. In (b) the eigenvalues are interior and we see the algorithm has much more difficulties in finding the wanted eigenvalues. We note that different runs can give diverent converged eigenvalues as the algorithm is subject to random numbers. . . . . 93
5.2 Figures (a)-(b) resp. (c)-(d) the results for the random $m$EP in Example 5.2. We plot the wanted eigenvalue $\mu_m$ in blue and the found eigenvalues in red. We see that if the wanted eigenvalues are exterior we can find them. In case they are interior or/and more close to each other, we see that it is far more difficult for the algorithm to find the wanted eigenvalues. Even after 100 sweeps we can only find a few. . . . . 95

5.3 Figures corresponding to Example 5.4. We plot the time needed to perform the different parts of the algorithm when we increase the size $n$ of the matrices. We see that solving of the projected eigenvalue problem is computationally the most demanding. The computational time needed for the selection and the convergence, depends on how many eigenvalues are converged and at which sweep it converges, explaining the decrease in the computational time in (c). . . . . 97
5.4 Figures corresponding to Example 5.4. We plot the time needed to perform the different parts of the algorithm when we increase the size $n$ of the matrices. We see that the solving of the projected eigenvalue problem is the most computationally demanding. The computational time needed for the selection and the convergence, depend on how many eigenvalues are converged and at which sweep it converges, explaining the decrease in the computational time in (c). We observe in (a) and (b) a kink in the curve at $n = 130$. We believe that this can be explained by the fact that the rank of $\mathbf{X}_{\neq k}$ at $n = 130$ was smaller than for other values of $n$. . . . . 101

# List of Tables

3.1 The results for Example 2 . . . . . 49
3.2 Results . . . . . 50
3.3 Results for example from structural mechanics . . . . . 51

4.1 Statistics associated to the results shown in Figure 4.1. . . . . 67

# Chapter 1

# State of the art and objectives

*To a mathematician, real life is a special case*
*(Unknown)*

## 1.1 Introduction

For people with no interest in mathematics or engineering, it is difficult to imagine what eigenvalues are so we start by explaining why it is of interest to the scientific-and engineering community. An eigenvalue is a mathematical concept that appears in a lot of applications. It is theoretically well understood but for some applications the current algorithms hit their limits. Therefore eigenvalue problems are still a popular research subject from the more theoretical side as well as in applications. Examples of applications can be found in the construction of buildings and bridges to study resonance, Google's PageRank algorithm, in uncertainty quantification (determining the probability of certain outcomes when not everything is precisely known) [70, 48]. Also in data science and machine learning eigenvalue problems play a role, see for example the technique Principal Component Analysis [33].

The aim of this chapter is to give an introduction into the main topics of this thesis along with the state-of-the-art. We start with defining eigenvalues and we elaborate more on some specific applications of eigenvalues. The problems we solve in this thesis are derived from the eigenvalue problem. We define these formally, give some applications and describe the state-of-the-art along with the current challenges. We end this chapter with an overview of the contributions

per chapter.

## 1.2 Eigenvalue problems

All different topics we discuss in this thesis are related to eigenvalue problems. Therefore we start with explaining what is meant with this and we go through its main properties.

An *eigenpair* $(\lambda, \mathbf{x})$ of a matrix pair $(\mathbf{A}, \mathbf{B}), \mathbf{A}, \mathbf{B} \in \mathbb{C}^{n\times n}$ is a pair such that

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}, \tag{1.1}$$

where $\lambda \in \mathbb{C} \cup \{\infty\}$ is called an *eigenvalue* and $\mathbf{x} \neq 0 \in \mathbb{C}^n$ is called an *eigenvector*. If $\mathbf{B}$ is the identity matrix $\mathbf{I}_n$, then (1.1) is called a *standard eigenvalue problem*. In that case, we say that $(\lambda, \mathbf{x})$ is an *eigenpair* of $\mathbf{A}$.

We assume in this thesis that $\mathbf{B}$ is invertible which means that (1.1) is equivalent to the standard eigenvalue problem

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda\mathbf{x}. \tag{1.2}$$

As a consequence, $\lambda$ can not be infinite. The set of all eigenvalues is denoted by $\mathbf{\Lambda}(\mathbf{A}, \mathbf{B})$ and is called the *spectrum*. From (1.1) it follows that

$$(\mathbf{A} - \lambda\mathbf{B})\mathbf{x} = 0,$$

so $\mathbf{x} \in \mathrm{Ker}(\mathbf{A}-\lambda\mathbf{B})$. We call $\mathrm{Ker}(\mathbf{A}-\lambda\mathbf{B})$ the *eigenspace* of $\lambda$ and its dimension is called the *geometric multiplicity* of $\lambda$. If it is unclear from the context from which matrix pair $\lambda$ is the eigenvalue, we denote an eigenvalue of the pencil $(\mathbf{A}, \mathbf{B})$ by $\lambda(\mathbf{A}, \mathbf{B})$.

Eigenvalues can also be defined through the characteristic polynomial

$$p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{B}).$$

The roots of $p(\lambda)$ are the eigenvalues of $(\mathbf{A}, \mathbf{B})$. If we take a closer look at $p(\lambda)$, we see that this is a polynomial of degree $n$, so it has $n$ roots and therefore $(\mathbf{A}, \mathbf{B})$ has $n$ eigenvalues. This does not mean that there are $n$ different eigenvalues, as $p(\lambda)$ may have a root with a multiplicity greater than one. The multiplicity of $\lambda$ as a root of $p(\lambda)$ is called the *algebraic multiplicity* of $\lambda$. The geometric multiplicity is always smaller or equal than its algebraic multiplicity. If it is strictly smaller, then $\lambda$ is called *defective*. If an eigenvalue has algebraic multiplicity equal to one, this eigenvalue is called *simple*.

Properties of a matrix (pencil) induce properties of their eigenvalues. We say that $\mathbf{A}$ is *Hermitian* if

$$\mathbf{A}(i,j) = \overline{\mathbf{A}(j,i)}, \quad i,j = 1,\ldots,n$$

where $\bar{\cdot}$ represents the complex conjugate and we call $\mathbf{A}$ *positive definite* if all eigenvalues of $\mathbf{A}$ are strictly positive.

Before we state some properties of eigenvalues, we define the concept of orthonormality. Let $\mathbf{B}$ be a positive definite matrix, then we define the scalar product with respect to $\mathbf{B}$ as

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle_B = \mathbf{v}_i^T \mathbf{B} \mathbf{v}_j. \tag{1.3}$$

We call the set of vectors $\mathbf{v}_1,\ldots,\mathbf{v}_k$ $\mathbf{B}$-*orthonormal* if

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle_B = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}. \tag{1.4}$$

Note that the identity matrix $\mathbf{I}$ is Hermitian and positive definite, so it defines a scalar product. In this case we omit the subscript and we call it just *orthonormal*. Remark that eigenvectors are defined up to a constant so without loss of generality we assume that for every eigenvector $\mathbf{x}$ of $(\mathbf{A},\mathbf{B})$ it holds that

$$\langle \mathbf{x}, \mathbf{x} \rangle_B = 1.$$

**Property 1.** *1. Let $\mathbf{A}$ and $\mathbf{B}$ be real and symmetric and assume that $\mathbf{B}$ is positive definite. All eigenvalues $\lambda_1,\ldots,\lambda_n$ and eigenvectors $\mathbf{x}_1,\ldots,\mathbf{x}_n$ are real and the eigenvectors can be chosen as a $\mathbf{B}$-orthonormal set. Define $\mathbf{X}_\lambda = [\mathbf{x}_1,\ldots,x_n]$ and $\mathbf{\Lambda}_{diag} = diag(\lambda_1,\ldots,\lambda_n)$, then*

$$\mathbf{X}_\lambda^* \mathbf{A} \mathbf{X}_\lambda = \mathbf{\Lambda}_{diag}. \tag{1.5}$$

*2. Let $\mathbf{A}$ and $\mathbf{B}$ both be real and let $(\lambda,\mathbf{x})$ be an eigenpair of $(\mathbf{A},\mathbf{B})$, then $\left(\bar{\lambda},\bar{\mathbf{x}}\right)$ is also an eigenpair.*

In applications one is usually not interested in all eigenvalues, but in eigenvalues that meet a certain condition. This condition can be that we only want to calculate the $k$ eigenvalues with largest/smallest real part, largest/smallest in modulus, .... Calculating all eigenvalues, while one is only interested in the $k$ smallest or largest, is inefficient certainly if the matrix is large. Even the most efficient methods have a computational cost of $\mathcal{O}(n^3)$ operations for calculating all eigenvalues of non-structured matrices, for structured matrices it can be $\mathcal{O}(n^2)$. In Section 2.1 we discuss how we can approximate some eigenvalues using subspaces.

### 1.2.1 Applications of eigenvalue problems

In the introduction we gave a short overview of the different applications of eigenvalues. Here we discuss a few of them more deeply.

#### PageRank algorithm

By far the most famous application of eigenvalue problems is in the web search engine that everyone knows: Google. With around 90 billion ($90 \times 10^9$ ) searches every month it is the most used web search engine and *google.com* is the most visited website [64]. The algorithm that used to form the basis of this search engine is based on eigenvalues. We explain briefly how it works. The following is based on [11] and [51].

Let $n$ be the number of webpages and we associate with each number from 1 to $n$ a certain website. We make a large sparse matrix $\mathbf{G}$ where we place a one at position $(i, j)$ if there is a hyperlink at page $i$ to page $j$.

We now model an average user of a web search engine. An average user goes from website to website by clicking on the hyperlinks on a website. As at some moment he/she wants to search something completely different, he/she goes to a completely different website with probability $1 - p$ $(0 < p < 1)$. This can be modelled by a discrete time Markov[1]-chain. The matrix $\mathbf{A}$ is now defined as

$$\mathbf{A}(i, j) = p\frac{\mathbf{G}(i, j)}{r_i} + \delta$$

with $\delta = (1 - p)/n$ and $r_i$ the number of websites with a link on page $i$. It can be seen that each row of $\mathbf{A}$ sums up to one, which means this is a row-stochastic matrix. If we are at website $j$, $\mathbf{A}(i, j)$ represents the probability that we were coming from website $i$.

Let us say that we start at website 1. Define $\mathbf{v}_0 = [1, 0, \ldots, 0]^T$ and define $\mathbf{v}_1$ as

$$\mathbf{v}_1 = \mathbf{A}^T\mathbf{v}_0$$

then the $j$th element of $\mathbf{v}_1$ represents the probability of going to the $j$th website starting from website 1. This means that $\mathbf{v}_1$ is the distribution after taking one step of this discrete time Markov Chain.

If we multiply $\mathbf{A}^T$ with $\mathbf{v}_1$, this $\mathbf{v}_2$ represents the probability distribution of going to website $j$ when coming from website 1 after two website views. It can

[1] Andrey Markov (1856-1922) [77] was a Russian mathematician. His main contribution is the concept Markov-chains. Together with his brother he proved the "Markov brothers' inequality". Also his son was a notable mathematician.

be proven via the Perron[2]-Frobenius[3] theorem that by repeating this enough times (let us say $m$ times), we find a vector $\mathbf{v}_m$ such that $\mathbf{v}_m = \mathbf{A}^T\mathbf{v}_m$. This means that $\mathbf{v}_m$ is an eigenvector of $\mathbf{A}^T$ with eigenvalue 1. As we are doing nothing more than the power method for calculating the largest eigenvalue, we can see that the maximal eigenvalue of $\mathbf{A}^T$ (and also of $\mathbf{A}$) is 1.

In the literature this eigenvector $\mathbf{v}_m$ is the stationary distribution of this Markov Chain as this vector does not change after taking extra steps of the Markov Chain. This $\mathbf{v}_m$ contains a measure for the fraction of time someone spends on a website. In this way we obtain a way to rank all websites.

If Google search would be purely based on the PageRank algorithm, a search for "eigenvalues", gives you all websites containing "eigenvalues" ranked according to its value in $\mathbf{v}_m$ in decreasing order.

Other applications of eigenvalues are more related to engineering.

### Stability of steady state solutions

Consider a non-linear system of the form

$$\mathbf{B}\frac{\partial \mathbf{u}}{\partial t} = f(\mathbf{u}), f : \mathbb{R}^n \to \mathbb{R}^n, u \in \mathbb{R}^n, \tag{1.6}$$

with $\mathbf{u}$ a state variable and with $\mathbf{B}$ the mass matrix [43]. A *steady state solution* of (1.6) is a solution that is independent of the time $t$, meaning that the left-hand side of (1.6) is 0.

Eigenvalues come into play to determine if a steady state solution is *asymptotically stable*. Roughly said, a steady state solution is asymptotically stable if starting from another solution sufficiently close to a steady state solution, it moves towards this steady state solution. For a more elaborate study on stability of solution of differential equations, we refer to [68].

Following property links the stability of a steady state solution to eigenvalues.

**Property 2.** *A steady state* $\mathbf{u}^*$ *of the system given in* (1.6) *is* asymptotically stable *if all eigenvalues of* $(\mathbf{A}, \mathbf{B})$ *with* $\mathbf{A}$ *the* Jacobian matrix $\dfrac{\partial f}{\partial u}$ *evaluated in* $\mathbf{u}^*$ *and* $\mathbf{B}$ *the mass matrix, have a negative real part.*

[2]Oscar Perron (1880-1975) [91] was a German mathematics professor. He made contributions in the field of differential equations. He introduced a paradox to make attention that a solution of an optimization problem might not exist: "Let $n$ be the largest positive integer. If $n > 1$, then $n^2 > n$ contradicting the definition of $n$. Hence $n = 1$".

[3]Ferdinand Georg Frobenius (1849–1917) [82] was a German mathematician. His main research topics were elliptic functions, differential equations, number theory and group theory. He was a student of Karl Weierstrass.

Systems of the form (1.6) usually come from a discretisation of a partial differential equation (PDE). PDEs arise in a large variety of interesting applications. Papers about stability analysis based on the rightmost eigenvalue can be found in fluid mechanics [8], structural engineering [23], electrical engineering [58] and ecology [22] (to name a few).

In Example 1 we give an example of a couple of PDEs in the field of fluid dynamics which after discretisation in space can be written in the form (1.6).

**Example 1.** *The following equations model the flow of a layer of viscoelastic fluid heated from below:*

$$\frac{\partial u}{\partial t} = (1-c)\frac{\partial^2 v}{\partial x^2} + c\frac{\partial^2 u}{\partial x^2} + ru - u^3$$

$$b\frac{\partial v}{\partial t} = u - v.$$

*The variable $u$ is the speed of the fluid and $v$ represents a quantity related to the viscoelastic forces. It is known in the literature as the Olmstead model [49].*

#### Resonance

In the introduction, we briefly mentioned resonance of structures as an application where eigenvalues come into play. Resonance happens when the frequency of an applied force on the considered object is equal (or close) to one of the natural frequencies. The object starts to wobble, which can lead to serious damage. Engineers need to take these natural frequencies into account when designing buildings and bridges. The natural frequencies of an object can be approximated by first discretising the involved PDE using (usually) a finite element method and then solving the eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}$$

with $\mathbf{A}$ the stiffness matrix and $\mathbf{B}$ the mass matrix. Typically the matrices $\mathbf{A}$ and $\mathbf{B}$ are both symmetric and $\mathbf{B}$ is positive definite. The approximated natural frequency $\omega$ is then derived via $\lambda = \omega^2$.

## 1.3 Parametrized eigenvalue problems

Two of the three papers that I wrote during my Phd, were about algorithms for determining extreme eigenvalues of parametrized eigenvalues. Suppose we are

given a matrix pair

$$(\mathbf{A}(\omega), \mathbf{B}(\omega))$$

depending on a parameter $\omega \in \Omega$, with $\Omega$ a compact subset of $\mathbb{R}^d$. The goal is to calculate an eigenvalue $\lambda(\omega)$ of $(\mathbf{A}(\omega), \mathbf{B}(\omega))$ that is extreme in some sense for many different parameter values $\omega$. An extreme eigenvalue can be the eigenvalue with largest real part, which is largest in absolute value or closest to a prescribed complex number $\sigma$. We always sort the eigenvalues such that $\lambda_1$ is the eigenvalue we look for. Due to the possible large set of parameter values that we consider, it might be infeasible from a computational point of view to apply a standard solver on each matrix pair $(\mathbf{A}(\omega), \mathbf{B}(\omega))$.

### 1.3.1 Applications of parametrized eigenvalue problems

We discuss two main applications of parametrized eigenvalue problems.

#### Reduced basis method for PDEs

The first application can be found in the reduced basis method for parametrized symmetric elliptic coercive PDEs. We only explain briefly what is meant with this, for a more extensive study we refer to the review paper [59]. A parametrized elliptic PDE is a PDE of the form

$$a_1(\omega)\frac{\partial^2 u}{\partial x^2} + 2a_2(\omega)\frac{\partial^2 u}{\partial x \partial y} + a_3(\omega)\frac{\partial^2 u}{\partial y^2} + a_4(\omega)\frac{\partial u}{\partial x} + a_5(\omega)\frac{\partial u}{\partial y} + a_6(\omega)u + a_7(\omega) = 0 \tag{1.7}$$

such that

$$a_2(\omega)^2 - a_1(\omega)a_3(\omega) < 0.$$

The function $u$ which depends on the parameter $\omega$ is unknown. To guarantee uniqueness of $u$, boundary conditions on the domain of $u$ are imposed. For explaining the terms 'symmetric' and 'coercive' in this context, we define the *weak formulation of a parametrized PDE* as

$$a(u(\omega), v; \omega) = f(v), \quad \forall v \in X \tag{1.8}$$

where $a(\cdot, \cdot; \omega) : X \times X \to \mathbb{R}$ is a bilinear form for all parameter values $\omega \in \Omega$ and $X$ is a Hilbert space. Furthermore, we call (1.7) symmetric if $a(\cdot, \cdot; \omega)$ is a symmetric function, i.e., $a(u, v; \omega) = a(v, u; \omega), \forall u, v \in X$. We assume that $a$ is continuous. Without going into the details, we call the PDE *coercive* if

$$\alpha(\omega) := \inf_{u \in X \setminus \{0\}} \frac{a(u, u; \omega)}{\|u\|_X^2} > 0, \quad \forall \omega \in \Omega, \tag{1.9}$$

where $\|.\|_X$ is the induced energy norm on $X$.

To approximate $a(\omega)$, a matrix $\mathbf{A}(\omega) \in \mathbb{R}^{n\times n}$ that comes from a conforming finite element discretisation of (1.8) and the matrix $\mathbf{B}$ is the discretisation of the induced energy norm on $X$, are computed. A discretized version of (1.9) is then

$$\alpha^n(\omega) = \inf_{\mathbf{u}\in\mathbb{R}^n\setminus\{0\}} \frac{\mathbf{u}^T\mathbf{A}(\omega)\mathbf{u}}{\mathbf{u}^T\mathbf{B}\mathbf{u}}$$

and satisfies

$$0 < \alpha(\omega) \leq \alpha^n(\omega), \quad \forall\omega \in \Omega.$$

In the next paragraph we explain how this is used within the reduced basis method. Because of the positive definiteness of $\mathbf{B}$ and the symmetry of both $\mathbf{A}(\omega)$ and $\mathbf{B}$, calculating $\alpha^n(\omega)$ is equivalent to calculating the minimal eigenvalue of the matrix pencil $(\mathbf{A}(\omega), \mathbf{B})$.

We explained the origin of parametrized eigenvalue problems, we discuss now its use within the reduced basis method. The reduced basis method is an example of a model order reduction technique. This is a class of techniques for reducing the complexity of models in numerical simulations. It is sufficient for now that in the reduced basis method we build up a subspace $\mathcal{V}$ in which we look for the solution of a parametrized linear system

$$\mathbf{A}(\omega)\mathbf{u} = \mathbf{b}, \quad \omega \in \Omega,$$

where $\mathbf{b}$ comes from the finite element discretisation of $f$ in (1.8). The approximated solution is of the form $\hat{\mathbf{u}}(\omega) := \mathbf{V}\mathbf{u}^{\mathcal{V}}(\omega)$ with $\mathbf{V}$ the matrix consisting of an orthonormal basis of $\mathcal{V}$, then the error on the true solution is bounded by

$$\|\mathbf{u}(\omega) - \hat{\mathbf{u}}(\omega)\| \leq \frac{\|\mathbf{r}(\omega)\|_X}{\lambda_1(\omega)} \tag{1.10}$$

with $\mathbf{r}(\omega) := \mathbf{A}(\omega)\hat{\mathbf{u}}(\omega) - \mathbf{b}$ and $\lambda_1(\omega)$ is the minimal eigenvalue of $(\mathbf{A}(\omega), \mathbf{B})$. It is for calculating the bound in (1.10) that we make a global approximation of the minimal eigenvalue of the parametrized eigenvalue problem $(\mathbf{A}(\omega), \mathbf{B})$.

**Example 2.** *This example is taken from [65, Example 4.4]. We consider a stationary heat equation. The parameter space is* $\Omega = [0.02, 0.5] \times [2, 8]$. *The first parameter determines the coefficient in the Robin*[4] *boundary condition and the second parameter determines the length of the domain, see Figure 1.1. There are analytic functions* $\theta_i(\omega), i = 1, 2, 3$ *and matrices* $\mathbf{A}_i, i = 1, 2, 3$ *such that*

$$\mathbf{A}(\omega) = \sum_{i=1}^{3} \theta_i(\omega)\mathbf{A}_i.$$

[4]Victor Gustave Robin (1855-1897) [98] was a French applied mathematician.

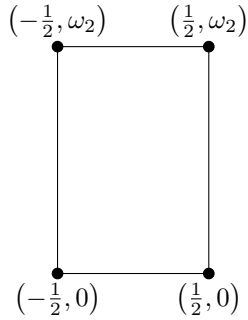



Figure 1.1: The geometry of the underlying PDE in Example 2. The parameter $\omega_2$ determines the length of the domain.

*The matrix* $\mathbf{B}$ *is constant. The dimension of the matrices* $\mathbf{A}(\omega)$ *and* $\mathbf{B}$ *is* 1311.

### Pseudospectrum

We saw in Property 2 that a steady state solution of (1.6) is asymptotically stable if all eigenvalues of the Jacobian matrix $\mathbf{A}$ (we assume that the mass matrix $\mathbf{B} = \mathbf{I}$) have a negative real part. To determine whether all eigenvalues have real part below 0, only the eigenvalue with maximal real part matters. This is related to the *spectral abscissa*, defined as

$$a = \max\{\mathrm{Re}(\lambda)|\lambda \in \Lambda(\mathbf{A})\}.$$

It is known (see [72]) that this is not a robust measure of stability as small perturbations can lead to significant changes in the spectrum. Therefore we define the $\epsilon$-*pseudospectrum* as

$$\Lambda_\epsilon(\mathbf{A}) = \{z \in \mathbb{C}|z \in \Lambda(\mathbf{B}) \text{ for some } \mathbf{B} \in \mathbb{C}^{n\times n} \text{ with } \|\mathbf{A}-\mathbf{B}\|_2 \leq \epsilon\}. \quad (1.11)$$

In words, this is the union of the spectra of all matrices which are on distance $\epsilon$ of $\mathbf{A}$. It was proven in [24], that (1.11) can be rephrased as

$$\Lambda_\epsilon(\mathbf{A}) = \{z \in \mathbb{C}|\sigma_n(\mathbf{A}-z\mathbf{I}_n) \leq \epsilon\}. \quad (1.12)$$

where $\sigma_n(.)$ represents the smallest singular value. Analogously the $\epsilon$-*pseudospectral abscissa* is defined as

$$a_\epsilon = \max\{\mathrm{Re}(\lambda)|\lambda \in \Lambda_\epsilon(\mathbf{A})\}.$$

For a Hurwitz[5] matrix $\mathbf{A}$ i.e. a matrix with only eigenvalues with real part smaller than zero, the perturbation $\epsilon$ for which $a_\epsilon = 0$ is called the *distance to instability*. This is the maximal perturbation that can be made, such that the system stays asymptotically stable. We now explain how this is related to parametrized eigenvalue problems.

Let $\Omega = [a_1, b_1] \times [a_2, b_2] \subset \mathbb{R} \times \mathbb{R}$ be a parameter space, then for all $\epsilon > 0$, $\Lambda_\epsilon(\mathbf{A}) \cap \Omega$ can be calculated via following parametrized eigenvalue problem. Define $z = \omega_1 + \omega_2 i$ with $(\omega_1, \omega_2) \in \Omega$, then

$$\begin{aligned}\sigma_n^2(\mathbf{A} - z\mathbf{I}_n) &= \lambda_1\left((\mathbf{A} - z\mathbf{I}_n)^*(\mathbf{A} - z\mathbf{I}_n)\right) \\ &= \lambda_1\big(\mathbf{A}^*\mathbf{A} - \omega_1(\mathbf{A} + \mathbf{A}^*) + \omega_2 i(\mathbf{A} - \mathbf{A}^*) \\ &\quad + \left(\omega_1^2 + \omega_2^2\right)\mathbf{I}_n\big),\end{aligned}$$

where $\lambda_1$ represents in this case the smallest eigenvalue. Define matrices $\mathbf{A}_1 = \mathbf{A}^*\mathbf{A}, \mathbf{A}_2 = -(\mathbf{A} + \mathbf{A}^*), \mathbf{A}_3 = i(\mathbf{A} - \mathbf{A}^*)$ and $\mathbf{A}_4 = \mathbf{I}$ and the functions $f_1(\omega_1, \omega_2) = 1, f_2(\omega_1, \omega_2) = \omega_1, f_3(\omega_1, \omega_2) = \omega_2$ and $f_4(\omega_1, \omega_2) = \omega_1^2 + \omega_2^2$, then by solving the parametrized eigenvalue problem with matrix

$$\hat{\mathbf{A}}(\omega_1, \omega_2) = \sum_{i=1}^{4} f_i(\omega_1, \omega_2)\mathbf{A}_i,$$

we can calculate an approximation of the $\epsilon$-pseudospectrum within $\Omega$ and its $\epsilon$-pseudospectral abcissa.

A related problem consists of solving stochastic eigenvalue problems. The difference with parametrized eigenvalue problems is that the parameters correspond to realizations of stochastic variables. Instead of making a global approximation of the extreme eigenvalue, we are interested in statistical moments of the extreme eigenvalue. There are several approaches proposed for this problem, for example adaptive sparse grid collocation and stochastic inverse iteration [27, 45]. In Section 3.5 we propose an algorithm for calculating the expected value of the minimal eigenvalue. This algorithm is based on the algorithm we made for parametrized eigenvalue problems in Chapter 3.

### 1.3.2 State-of-the-art

In the literature there are several ideas proposed for solving parametrized eigenvalue problem. Most of them deal with symmetric matrices where at least

[5] Adolf Hurwitz (1859 - 1919) [101] was a German mathematician. His domains of expertise were algebra, analysis, geometry and number theory.

one of $\mathbf{A}(\omega), \mathbf{B}(\omega)$ is positive definite for all $\omega$. Without loss of generality, we assume that $\mathbf{B}(\omega)$ is a positive definite matrix. In particular this means that all eigenvalues are real valued. We calculate the minimal or maximal eigenvalue. We assume we are calculating the minimal eigenvalue.

The first class of algorithms are polynomial based. This means that the extreme eigenvalue over the parameter is approximated by a polynomial. In [1], the authors approximate $\lambda_1(\omega)$ by sparse tensor products of Legendre[6] polynomials and in [21], polynomial chaos is advocated, a technique which is frequently used when the parameters are stochastic. Recently this technique was combined with low-rank approximations, see [17].

We believe that polynomials are not ideal for approximating the wanted eigenvalue over the parameter as it is likely that there are non-smooth parts in the surface of the eigenvalue under consideration even if the matrix pencil smoothly depends on $\omega$. This means we potentially need a polynomial of high degree for approximating the wanted eigenvalue over the parameter. We explain this via two examples. For a more elaborate study on the effect of the non-smoothness on polynomial-based methods, we refer to [19].

There are two causes for non-smooth behaviour of the minimal eigenvalue and they both have to do with parameter values for which the minimal eigenvalue has a multiplicity larger than one. If the minimal eigenvalue can be decomposed in multiple smooth surfaces, then at the intersection lines the minimal eigenvalue is non-smooth, see Figure 1.2. This figure represents the minimal eigenvalue over the parameter space in Example 2.

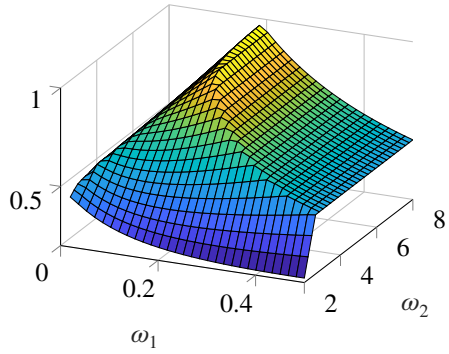



Figure 1.2: Surface of $\lambda_1(\omega)$ in Example 2

[6] Adrien-Marie Legendre (1752-1833) [75] was a French mathematician. His main mathematical contributions are Legendre polynomials and Legendre transformations. When the Eiffel Tower first opened, he was together with other prominent French scientists commemorated on plaques at the first stage of the tower. There is a lunar crater and a main-belt asteroid named after him.

It is not necessarily the case that if there is a parameter value with a multiple minimal eigenvalue, the surface is decomposable in multiple smooth surfaces. It does hold that if a parameter value $\omega$ has a minimal eigenvalue with multiplicity $m > 1$ for a given $\omega$ and is simple in an open set around $\omega$, then it can be decomposed in this open set in each direction into $m$ differentiable curves and something similar can be done for $\mathbf{B}$-normalized eigenvectors, see e.g [39, Ch. 9.]. However, this no longer holds for higher-dimensional surfaces, see Example 3.

**Example 3.** *Let $\Omega = [-0.5, 0.5] \times [-0.5, 0.5]$ be the parameter space and let the matrices be*

$$\mathbf{A}(\omega) = \mathbf{W}(\omega)^T \begin{bmatrix} \omega_1 + 1 & \omega_2 & & & \\ \omega_2 & -\omega_1 + 1 & & & \\ & & 3 & & \\ & & & \ddots & \\ & & & & n \end{bmatrix} \mathbf{W}(\omega), \mathbf{B}(\omega) = \mathbf{I}_n$$

*with $\mathbf{W}(\omega)$ such that $\mathbf{W}(\omega)^T\mathbf{W}(\omega) = \mathbf{I}_n$ for all $\omega \in \Omega$. All elements in the matrix which are left blank are assumed to be zero. The eigenvalues are $\lambda_1(\omega) = -\sqrt{\omega_1^2 + \omega_2^2} + 1$, $\lambda_2(\omega) = \sqrt{\omega_1^2 + \omega_2^2} + 1, \lambda_3(\omega) = 3, \lambda_4(\omega) = 4, \ldots, \lambda_n(\omega) = n$. The minimal eigenvalue $\lambda_1(\omega)$ is continuous but not derivable in $\omega = (0, 0)$. The two minimal eigenvalues are displayed in Figure 1.3. We observe that the surface defined by $\lambda_1(\omega)$ over the parameter cannot be decomposed in 2 smooth surfaces.*

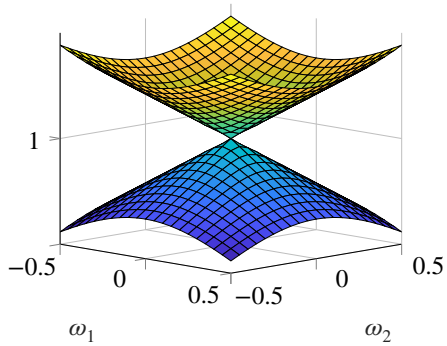



Figure 1.3: Surfaces that represent $\lambda_1(\omega)$ and $\lambda_2(\omega)$ in Example 3

Another approach is the so-called Successive Constraint Method (SCM), see [65] and the references therein. In paper [65], lower-and upper bounds for the wanted eigenvalue are calculated for each chosen parameter value $\omega$. By making these bounds more tight in each iteration, the authors associate an eigenvalue

with each parameter value. The main disadvantage of this method is that it does not provide a way to calculate a wanted eigenvalue of a parameter value other than the parameter values in the initial subset.

When the matrices are not symmetric or when one of the matrices is not positive definite, to the best of my knowledge, the methods used are based on polynomial (chaos) approximations, see e.g. [17, 21]. These methods are proposed in the context of stochastic eigenvalue problem.

## 1.4 Multiparameter eigenvalue problem

The second type of problem we handle in this thesis, is the solution of multiparameter eigenvalue problems. Although the name resembles the topic of the previous section, it is a totally different problem. Here, we compute solutions $((\mu_1, \mu_2, \ldots, \mu_m), (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m))$ of the *m-parameter eigenvalue problem* ($m$EP)

$$\begin{cases} \mathbf{A}_1\mathbf{x}_1 &= \mu_1\mathbf{B}_{11}\mathbf{x}_1 + \ldots + \mu_m\mathbf{B}_{1m}\mathbf{x}_1 \\ \mathbf{A}_2\mathbf{x}_2 &= \mu_1\mathbf{B}_{21}\mathbf{x}_2 + \ldots + \mu_m\mathbf{B}_{2m}\mathbf{x}_2 \\ & \vdots \\ \mathbf{A}_m\mathbf{x}_m &= \mu_1\mathbf{B}_{m1}\mathbf{x}_m + \ldots + \mu_m\mathbf{B}_{mm}\mathbf{x}_m \end{cases}, \tag{1.13}$$

where $\mathbf{A}_i, \mathbf{B}_{ij} \in \mathbb{R}^{n_i \times n_i}, i, j = 1, \ldots, m$. We call the $m$-tuple $(\mu_1, \mu_2, \ldots, \mu_m)$ an *eigenvalue-tuple* and $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ the associated *eigenvector-tuple*.

We are interested in a selection of eigenvalues. In this thesis, we aim for the eigenvalue-tuples whose $i$th component, $\mu_i$ has smallest modulus. Without loss of generality we assume that $i = m$. Note that for $m = 1$, we just have a generalised eigenvalue problem.

We refer to [3] for a detailed description of multiparameter eigenvalue problems.

### 1.4.1 Applications of multiparameter eigenvalue problems

The main application of multiparameter eigenvalue problems consists of solving certain separable boundary value problems. A boundary value problem is a differential equation such that additional constraints on the boundaries are satisfied.

It is difficult to state which boundary value problems are separable and which problems are not. An overview of separable boundary-value problems in physics

can be found in [103]. Whether the boundary value problem is separable depends not only on the differential equation but also on the object on which we solve the equation and the used coordinate system. If we take for example the Helmholtz[7] equation

$$\nabla^2 v + \alpha^2 v = 0 \tag{1.14}$$

on a certain object $\Omega$ with Dirichlet[8] boundary conditions $v_{|\delta\Omega} = 0$, then it is not separable when using Cartesian[9] coordinates on the cube but it is separable using ellipsoidal coordinates over an ellipsoid. Other equations that are separable on certain boundaries in certain coordinate systems are the Laplace[10] and the Schrödinger[11] equations. From (1.14) we can immediately derive that for all $\alpha, v = 0$ is a solution. We want to find the values for $\alpha$ such that there is a nonzero solution $v$.

If we assume a differential equation of second degree in the space variable $x$ and the boundary value problem is separable, we can rephrase the separable boundary value problem as a system of $m$ linear ordinary differential equations (ODEs) of the form

$$p_j(x_j)\, y_j''(x_j) + q_j(x_j)\, y_j'(x_j) + r_j(x_j)\, y_j(x_j) = \sum_{i=1}^{m} \mu_i s_{ji}(x_j)\, y_j(x_j) \tag{1.15}$$

with $j = 1, \ldots, m$ and $x_j \in [a_j, b_j]$. The solution of (1.15) is an $m$-tuple $(\lambda_1, \ldots, \lambda_m)$ and nontrivial functions $y_1, \ldots, y_m$ such that beside the equations (1.15) also additional boundary conditions are satisfied. In the next example we explain how we obtain the system of linear ODEs.

Discretisation of (1.15) leads to a multiparameter eigenvalue problem. There are multiple ways to discretise problems of the form (1.15) towards multiparameter eigenvalue problems. Mostly differential equations are discretised using finite

[7]Hermann von Helmholtz (1821-1894) [85] was a German medical doctor and physicist. He is most famous for his 'law of Conservation of Energy'.

[8]Johann Peter Gustav Lejeune Dirichlet (1805-1859) [93] was a German mathematician. He made numerous contributions to number theory and Fourier series. A lunar crater and an asteroid are named after him.

[9]Renatus Cartesius was the Latinized version of René Descartes (1596-1650) [95]. He was a French philosopher and scientist. He is known for the quote 'Je pense donc je suis' (I think therefore I am).

[10]Pierre-Simon Laplace (1749-1827) [94] was a French scientist. He made important contributions to engineering, mathematics, statistics, physics and astronomy. In mathematics, the Laplacian (a differential operator) and the Laplace transformation are named after him. He frequently used the quote 'it is therefore obvious that...' in his book 'Celestial mechanics' to give a signal for something true, but very difficult to prove. An asteroid is named after him.

[11]Erwin Schrödinger (1887-1961) [81] was an Austrian-Irish physicist. He made a number of significant contributions to quantum theory. He received a Nobel prize for the Schrödinger equation. A lunar crater and an asteroid are named after him.

differences, but this leads to large matrices and $m$EP with large matrices are challenging to solve. For the discretisation we use the spectral collocation method introduced in the paper [55]. The authors of this paper use Chebyshev[12] nodes as collocation points.

In the remainder of this section we explain how the Helmholtz problem can be written in the form (1.15) with ellipsoidal coordinates. In Chapter 5 we apply our method for solving multiparameter eigenvalue problems on this example.

**Example 4.** *[30, Section 2.1] Our aim is to look for couples $\alpha, v \neq 0$ that form the solution of the Helmholtz equation*

$$\nabla^2 v + \alpha^2 v = 0$$

*over the ellipsoid*

$$\Omega := \left\{(x,y,z) \in \mathbb{R}^3 \mid (x/x_0)^2 + (y/y_0)^2 + (z/z_0)^2 \leq 1\right\} \tag{1.16}$$

*subject to the Dirichlet boundary condition $u_{|\delta\Omega} = 0$ . The constants $x_0, y_0$ and $z_0$ are the radii of the semi-axes of the ellipsoid $\Omega$. We can assume that $z_0 > y_0 > x_0$. As the solution $v$ is separable in the ellipsoidal coordinates $(u_1, u_2, u_3)$, there exist functions $v_1(u_1), v_2(u_2)$ and $v_3(u_3)$ such that $v$ can be written as*

$$v(x(u_1,u_2,u_3), y(u_1,u_2,u_3), z(u_1,u_2,u_3)) = v_1(u_1)v_2(u_2)v_3(u_3).$$

*Using this property, solving the Helmholtz equation is equivalent with solving following system of ODEs*

$$\begin{aligned} t_i(t_i-1)(t_i-c)\tfrac{\partial^2 v_i}{\partial t_i^2} + \tfrac{1}{2}\left(3t_i^2 - 2(1+c)t_i + c\right)\tfrac{\partial v_i}{\partial t_i} + \\ (\mu_1 + \mu_2 t_i + \mu_3 t_i^2)v_i = 0, \quad i = 1,2,3 \end{aligned} \tag{1.17}$$

*with $c = a^2/b^2, a = (z_0^2 - x_0^2)^{1/2}, b = (z_0^2 - y_0^2)^{1/2}$ and $t_i = u_i^2/b^2$. The variables $\mu_1, \mu_2$ and $\mu_3$ are introduced to separate the Helmholtz equation into a set of ODEs. Only $\mu_3$ depends on $\alpha$. The proof of the result in* (1.17) *is quite elaborate and is therefore given in Appendix A. The boundaries on the elliptical coordinates $(u_1, u_2, u_3)$ satisfy*

$$z_0 > u_1 > a > u_2 > b > u_3 > 0,$$

*from this it follows that the new variables $(t_1, t_2, t_3)$ satisfy*

$$\frac{z_0^2}{b^2} > t_1 > \frac{a^2}{b^2} > t_2 > 1 > t_3 > 0.$$

[12]Pafnuty Chebyshev (1821-1984) [92] was a Russian mathematician. His research topics were probability, mechanics, statisics and number theory. He is known for the Chebyshev polynomials and the Chebyshev inequality. There is a lunar crater and an asteroid named after him.

*It can be proven that the solution $v$ is of the form*

$$v_i(t_i) = t_i^{\rho/2}(t_i-1)^{\sigma/2}(t_i-c)^{\tau/2} f_i(t_i), \quad i = 1,2,3 \tag{1.18}$$

*with $\rho, \sigma$ and $\tau$ can take values in $\{0,1\}$ and $f_i$ an integral function of $t_i$. We use this property to rewrite the ODEs and we obtain*

$$\begin{array}{l} t_i\,(t_i-1)\,(t_i-c)\,f_i'' + \frac{1}{2}\left(k_2 t_i^2 - 2k_1 t_i + k_0\right) f_i' + (-\lambda_0 + \mu_0 t_i)\, f_i \\ = (-\mu_1 - \mu_2 t_i - \mu_3 t_i^2) f_i, \quad i = 1,2,3 \end{array} \tag{1.19}$$

*with*

$$\begin{array}{c} \lambda_0 = \frac{1}{4}\left((\rho+\tau)^2 + (\rho+\sigma)^2 c\right), \quad \mu_0 = \frac{1}{4}(\rho+\sigma+\tau)(\rho+\sigma+\tau+1) \\ k_0 = (2\rho+1)c, \quad k_1 = (1+\rho)(1+c) + \tau + \sigma c, \quad k_2 = 2(\rho+\sigma+\tau) + 3. \end{array}$$

*The Dirichlet conditions on the boundary and the boundedness conditions at singular points, lead to following boundary conditions:*

$$\left\{\begin{array}{rl} F_1\left(z_0^2/b^2\right) & = 0 \\ \left(k_2c^2 - 2k_1c + k_0\right) F_j'(c) + 2\left(\mu_1 - \lambda_0 + (\mu_2+\mu_0)\,c + \mu_3 c^2\right) F_j(c) & = 0 \\ & \text{for } j = 1,2 \\ \left(k_2 - 2k_1 + k_0\right) F_j'(1) + 2\left(\mu_1 - \lambda_0 + \mu_2 + \mu_0 + \mu_3\right) F_j(1) & = 0 \\ & \text{for } j = 2,3 \\ k_0 F_3'(0) + 2\left(\mu_1 - \lambda_0\right) F_3(0) & = 0. \end{array}\right. \tag{1.20}$$

*We use the same discretisation scheme as in [55]. Discretisation of the left and right side respectively of* (1.19) *for $i = 1,2,3$ leads together with the boundary equations* (1.20) *to the matrices $\mathbf{A}_i$ respectively $\mathbf{B}_{i1}, \mathbf{B}_{i2}$ and $\mathbf{B}_{i3}$ in the multiparameter eigenvalue problem* (1.13)*.*

Two-parameter eigenvalue problems are also considered in the context of identifying Hopf[13] bifurcations, see [44, 16].

### 1.4.2 State-of-the-art

There are already methods available for large $n_i, i = 1,2,\ldots,m$, for the case $m = 2$ [55], [42] and $m = 3$ [30], but for $m > 3$, as far as we know, there are no methods yet that are applicable to multiparameter eigenvalue problems with large $n_i, i = 1,2,\ldots,m$.

[13]Eberhard Hopf (1902-1983) [102] was an Austrian-American mathematician and astronomer. He is known for his work in ergodic theory and bifurcation analysis.

Before we explain the methods in the aforementioned papers, we explain how we can recast the problem in (1.13) as a generalised eigenvalue problem. We define the $m \times m$-operator determinants

$$\boldsymbol{\Delta}_0 := \begin{vmatrix} \mathbf{B}_{11} & \dots & \mathbf{B}_{1m} \\ \vdots & & \vdots \\ \mathbf{B}_{m1} & \dots & \mathbf{B}_{mm} \end{vmatrix}_{\otimes} = \sum_{\sigma \in S_m} \operatorname{sgn}(\sigma) \mathbf{B}_{1\sigma_1} \otimes \mathbf{B}_{2\sigma_2} \otimes \dots \otimes \mathbf{B}_{m\sigma_m} \quad (1.21)$$

with $S_m$ the set of permutations of $\{1, \dots, m\}$ and $\operatorname{sgn}(\sigma)$ defined as the sign of the permutation $\sigma$. Analogously we define $\boldsymbol{\Delta}_i$ as

$$\boldsymbol{\Delta}_i := \begin{vmatrix} \mathbf{B}_{11} & \dots & \mathbf{B}_{1,i-1} & \mathbf{A}_1 & \mathbf{B}_{1,i+1} & \dots & \mathbf{B}_{1m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{m1} & \dots & \mathbf{B}_{m,i-1} & \mathbf{A}_m & \mathbf{B}_{m,i+1} & \dots & \mathbf{B}_{mm} \end{vmatrix}_{\otimes}, i = 1, \dots, m. \quad (1.22)$$

It can be proven that (1.13) is equivalent with the generalised eigenvalue problems

$$\boldsymbol{\Delta}_i \mathbf{x} = \mu_i \boldsymbol{\Delta}_0 \mathbf{x} \quad , i = 1, \dots m. \quad (1.23)$$

The dimension of $\boldsymbol{\Delta}_i$ is $n_1 n_2 \dots n_m$, which soon gets high even for modest $n_j, j = 1, \dots, m$, and $m$. The eigenvalue $\mu_i$ is the $i$th element of an eigenvalue-tuple of (1.13) and eigenvectors $\mathbf{x}$ of (1.23) are of the form $\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \dots \otimes \mathbf{x}_m$ where the $\mathbf{x}_i, i = 1, \dots, m$, together form an eigenvector-tuple of (1.13). In [42, 30] for $m = 2$ resp. 3, the idea is to iteratively project each equation of (1.13) on a subspace and to solve the smaller multiparameter eigenvalue problem. The problem for $m > 3$ is that even when all subspace dimensions are as low as 10, the projected eigenvalue problem (1.23) already has dimension $10^m$, which may become a challenging problem from a computational point of view. We conclude that the limit $m = 3$ is more a soft limit for the current state-of-the art methods.

If we broaden our scope to calculating all eigenvalue-tuples of a multiparameter eigenvalue problem, recently papers on homotopy methods [14, 57] were published for this problem. Due to the high number of eigenvalue-tuples, these methods are restricted to problems of small size.

## 1.5 Contributions and overview

We give an overview of the topics in each chapter together with the contributions we made.

**Chapter 2** This chapter presents preliminary material. We start by explaining the concept of subspace methods and how we can use them for eigenvalue problems. The idea is that we calculate eigenvalue of small matrix pairs that has the same extreme eigenvalues as the large matrix pair. We obtain the smaller matrix pairs by projecting the matrix pairs on a small subspace. We introduce a special kind of subspaces, named Krylov subspaces. In Chapters 4 and 5, we use tensors (or multidimensional arrays). We start with a general description of tensors, then we proceed with the Tensor-Train representation. Under certain conditions, this allows us to represent a large matrix or a tensor in a compact way and to do operations more efficiently.

**Chapter 3** In this chapter we present the first algorithm that we developed. For a matrix pencil $(\mathbf{A}, \mathbf{B})$, where both matrices depend on parameters in the parameter space, we propose a subspace method for making a global approximation of the minimal eigenvalue over the parameter space. Both matrices are assumed to be symmetric and at least one of the two needs to be positive definite. We build the subspace iteratively with eigenvectors and its partial derivatives evaluated in chosen points in the parameter space. Our major contribution is that we proved that adding partial derivatives of the eigenvector to the subspace, leads to eigenvalue problems for which its minimal eigenvalue has the property of Hermite interpolation of degree two. This means that we need fewer iterations before we have a sufficiently large subspace. The advantage of solving a parametrized eigenvalue problem by successively solving projected (and therefore) smaller parametrized eigenvalue problem is that we do not suffer from a possible non-smooth minimal eigenvalue. We motivate our choices theoretically and show its effectiveness on several examples. In the last section of this chapter, we extend our algorithm to calculate the expected value of the minimal eigenvalue when the parameters correspond to realizations of random variables.

**Chapter 4** In the second algorithm we solve the same problem as in Chapter 3, but now for standard eigenvalue problems which are not (necessarily) symmetric. The way the subspace is built up, is completely different. Here we extend a Krylov method for calculating extreme eigenvalues of a single matrix. The method boils down to applying a Krylov method on all parameter values at the same time. As we cannot add the iteration vectors for all parameter values to the subspace, we add a subspace based on low-rank decompositions of the tensor containing all these vectors to the subspace in every iteration. This way, we introduce errors in the algorithm. To be precise, we make a low-rank decomposition of the eigenvector and of the residual over the parameter. The major contribution is that we give theoretical justification and that we illustrate

with examples that, to some extent, it converges faster if we tolerate a larger error. As far as we know, this is the first time a tensor-Krylov method is used for solving parametrized eigenvalue problems.

**Chapter 5** The last project discussed in this thesis, is on solving multiparameter eigenvalue problems (1.13) using tensors. This kind of problems is proven to be equivalent to (potentially) very large generalised eigenvalue problems. To the best of our knowledge, the current subspace methods can only solve small problems or problems with $m \leq 3$, because otherwise the constructed subspace becomes too large. We explore the path of solving the involved generalised eigenvalue problem (1.23) by interpreting the involved matrices as Tensor-Train operators. The idea is based on an algorithm for standard symmetric eigenvalue problems. As our problem can be recast as a nonsymmetric generalised eigenvalue problem with specific properties (e.g the eigenvectors are of rank one seeing as a tensor), we rewrote large parts of the original algorithm such that it works well for our type of problems. Our major contribution is that we showed that tensors can be used to solve multiparameter eigenvalue problems with $m > 3$ and reasonable size of the involved matrices efficiently. Numerical experiments confirm these claims.

**Chapter 6** We conclude this thesis by summarizing the main results of this thesis.

The code that is used to make the figures and tables in this thesis is made publicly available. The link to the code is stated in the introduction of each chapter. All algorithms written in the previous sections are implemented in Matlab version R2020a. All experiments are performed on an Intel i5-6300U with 2.5 GHz and 8 GB RAM.

# Chapter 2

# Preliminaries

*Do not worry about your difficulties in mathematics. I can assure you mine are still greater.*
*(Albert Einstein)*

In this chapter we introduce the concept of (Krylov) subspaces and tensors. All of these terms will be extensively used and form the main basis for the remainder of the thesis. The part about (Krylov) spaces is based on [61, 9] and the part about tensors is based on [37, 50]. We state some basic properties necessary for understanding the thesis. More advanced tools and properties are discussed when they are used.

## 2.1 Subspaces and projection methods

All different problems we tackle in this thesis are solved by making *subspaces* and we approximate the wanted eigenvalues within this subspace. We introduce subspaces in general and explain how they are used.

A subspace $\mathcal{V}$ is a vectorspace that is (in our case) embedded within $\mathbb{C}^n$. A *basis* of $\mathcal{V}$ is a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{n_v}$ such that

$$\mathcal{V} = \text{span}\left(\mathbf{v}_1, \ldots, \mathbf{v}_{n_v}\right).$$

This means that for every vector $\mathbf{v} \in \mathcal{V}$ there exists constants $a_1, \ldots, a_{n_v} \in \mathbb{C}$ such that

$$\mathbf{v} = \sum_{i=1}^{n_v} a_i \mathbf{v}_i. \tag{2.1}$$

In our applications, we keep the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{n_v}$ real valued and we store them in a matrix $\mathbf{V} \in \mathbb{R}^{n \times n_v}$. We assume that the columns of $\mathbf{V}$ are an orthonormal set, meaning that $\mathbf{V}^T\mathbf{V} = \mathbf{I}_{n_v}$. If it is not the case, the columns of $\mathbf{V}$ can be orthonormalised via the *Gram-Schmidt method* which we discuss in the next section.

In this thesis we apply projection methods on different types of eigenvalue problems. In a projection method, an eigenvector is approximated by a vector inside a subspace $\mathcal{V}$ such that the residual is orthogonal with respect to a subspace $\mathcal{W}$. The following property proves that we can approximate an eigenpair in this way.

**Property 3.** *$(\lambda, \mathbf{x})$ is an eigenpair of $(\mathbf{A}, \mathbf{B})$ and $\mathbf{x} \in \mathcal{V}$ then $\left(\lambda, \mathbf{V}^T\mathbf{x}\right)$ is an eigenpair of $\left(\mathbf{W}^T\mathbf{A}\mathbf{V}, \mathbf{W}^T\mathbf{B}\mathbf{V}\right)$ where the columns of $\mathbf{W}$ form an orthonormal basis for $\mathcal{W}$.*

If $\mathcal{V}$ is different from $\mathcal{W}$, then the technique corresponds to an *oblique projection method*. In this thesis only the case $\mathcal{W} = \mathcal{V}$ is considered and these methods are called *orthogonal projection methods*. This means that we approximate eigenvalues of $(\mathbf{A}, \mathbf{B})$ by constructing a small subspace $\mathcal{V}$ and calculating the eigenvalues of the *projected eigenvalue problem*

$$\left(\mathbf{V}^T\mathbf{A}\mathbf{V}, \mathbf{V}^T\mathbf{B}\mathbf{V}\right). \tag{2.2}$$

In the literature this procedure is known as the *Rayleigh-Ritz procedure*[1].

Eigenpairs of (2.2) are denoted by $\left(\lambda^{\mathcal{V}}, \mathbf{x}^{\mathcal{V}}\right)$. In the literature $\mathbf{y} := \mathbf{V}\mathbf{x}^{\mathcal{V}}$ is called a *Ritz vector* and $\left(\lambda^{\mathcal{V}}, \mathbf{y}\right)$ is called a *Ritz pair*. The following result is well-known in the literature [61, Proposition 6.8] for standard eigenvalue problems.

**Property 4.** *Let $\left(\lambda^{\mathcal{V}}, \mathbf{y}\right)$ be a Ritz pair of $\mathbf{A}$ with respect to the subspace $\mathcal{V}$, then the residual satisfies*

$$\mathbf{r} = \left(\mathbf{I} - \mathbf{V}\mathbf{V}^T\right)\mathbf{A}\mathbf{y} \tag{2.3}$$

*so the residual is the result of applying matrix $\mathbf{A}$ on $\mathbf{y}$ orthogonalized with respect to $\mathbf{V}$.*

[1] John William Strutt, 3rd Baron Rayleigh (1842-1919) [87] was an English physicist. Among other awards, he won the Noble Prize in Physics in 1904 for "his investigations of the densities of the most important gases and for his discovery of argon in connection with these studies." A lunar as well as a Martian crater are named after him. Walther Heinrich Wilhelm Ritz (1878–1909) [100] was a Swiss theoretical physicist. He has a lunar crater named after him. He invented the method but due to the similarity to the Rayleigh quotient, Rayleigh is wrongly considered as one of the inventors.

*Proof.* We can express

$$\begin{aligned}\mathbf{r} &= \mathbf{A}\mathbf{y} - \lambda^{\mathcal{V}}\mathbf{y} \\ &= \mathbf{A}\mathbf{V}\mathbf{x}^{\mathcal{V}} - \mathbf{V}\left(\mathbf{V}^T\mathbf{A}\mathbf{V}\right)\mathbf{x}^{\mathcal{V}} \\ &= \left(\mathbf{I} - \mathbf{V}\mathbf{V}^T\right)\mathbf{A}\mathbf{V}\mathbf{x}^{\mathcal{V}}.\end{aligned}$$

□

In each chapter we tackle the problem using subpaces, but the way it is constructed is different and so are the properties. Before using the subspace $\mathcal{V}$, we need to construct an orthonormal basis for it.

### 2.1.1 Orthonormalisation

The basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_v}$ of the $n_v$-dimensional subspace $\mathcal{V}$ is stored in the matrix $\mathbf{V}$ as $\mathbf{V}(:, i) = \mathbf{v}_i, i = 1, \ldots, n_v$. This basis is called orthonormal if the basis vectors are mutually orthonormal. In our algorithms we build up a subspace iteratively, usually after each iteration this orthonormality gets (partially) broken, so we need a method that can make an orthonormal basis starting from a non-orthonormal basis.

There are various methods to obtain an orthonormal basis. The command `orth` in Matlab, computes an SVD-decomposition $\mathbf{V} = \mathbf{USZ}$, then $\mathbf{U}$ is the basis we look for. Eventually, one can choose to truncate $\mathbf{U}$ when the associated singular values are below some threshold. We assume that the singular values are in decreasing order.

If we already have a set of orthonormal vectors and we want to add a vector to this set, it is highly inefficient to compute again a SVD-decomposition, instead one can use the *Gram-Schmidt method*[2] for this. The idea behind the Gram-Schmidt algorithm is that before we add each vector to the subpace, we orthonormalise it against the already built subspace. We state this algorithm in Algorithm 1. It has a complexity of $\mathcal{O}({n_v}^2 n)$ elementary operations, with $n_v$ the number of vectors and $n$ the dimension of each vector.

[2] Jørgen Pedersen Gram (1850–1916) [88] was a Danish mathematician and actuary. As a son of a farmer, he published also about forestry. He died after he was hit by a bicycle. Erhard Schmidt (1876-1959) [80] was a Estonian-German mathematician. He is considered as one of the pioneers in functional analysis. The method was already known by Laplace and Cauchy before Gram and Schmidt [84].

**Algorithm 1** Modified Gram-Schmidt algorithm to make an orthonormal set of vectors.

**Input:** Set of $n_v$ non-zero vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_v}$
**Output:** orthonormalized set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_v}$

1: **for** $i = 1, \ldots, n_v$ **do**
2: **for** $j = 1, \ldots, i-1$ **do**
3: $\mathbf{v}_i = \mathbf{v}_i - (\mathbf{v}_j^* \mathbf{v}_i)\mathbf{v}_j$
4: **end for**
5: $\mathbf{v}_i = \mathbf{v}_i / \|\mathbf{v}_i\|_2$
6: **end for**

This algorithm can be instable if $\|\mathbf{v}_i\|$ is close to 0 (on line 5 in Algorithm 1). It can be made stable with reorthogonalization, i.e., repeating the loop 2-4 of Algorithm 1 once more.

This method is commonly used in combination with Krylov subspaces to make an orthonormal basis for this subspace. In the next section we introduce Krylov subspaces and explain their use when calculating a selected subset of eigenvalues of a standard eigenvalue problem.

### 2.1.2 Krylov subspaces

We restrict ourselves in this subsection to the calculation of the largest eigenvalue (in absolute value) of a nondefective matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Furthermore we assume that $\mathbf{A}$ has a dominant eigenvalue and we sort the eigenvalues as $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. Starting from an initial vector $\mathbf{v} = \sum_{i=1}^{n} a_i \mathbf{x}_i$, it can be verified that

$$\mathbf{A}^k \mathbf{v} = \sum_{i=1}^{n} \lambda_i^k a_i \mathbf{x}_i = \lambda_1^k \sum_{i=1}^{n} \left( \frac{\lambda_i}{\lambda_1} \right)^k a_i \mathbf{x}_i, \quad k = 1, 2, \ldots.$$

This means that the series

$$\mathbf{v}, \mathbf{A}\mathbf{v}, \mathbf{A}^2\mathbf{v}, \ldots \tag{2.4}$$

converges to $\mathbf{x}_1$ if $a_i \neq 0$ and $\lambda_1$ is a simple eigenvalue. If $\lambda_1$ is an eigenvalue with geometric multiplicity $m$, then it converges to a linear combination of $\mathbf{x}_1, \ldots, \mathbf{x}_m$. This procedure is known as the *power method*. Note that when we want to calculate the eigenvalue closest to a shift $\sigma$, one can use the same procedure but with $(\mathbf{A} - \sigma\mathbf{I})^{-1}$ instead of $\mathbf{A}$.

In the power method the vector $\mathbf{A}^{k-1}\mathbf{v}$ is used as an approximation of the eigenvector in iteration $k$, the main idea of Krylov methods is that we search for an approximation in the subspace spanned by the first vectors of (2.4).

**Definition 1.** *(Krylov subspace) The* Krylov[3] subspace *with matrix* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *and nonzero vector* $\mathbf{v} \in \mathbb{R}^n$ *of order* $m$ *is defined as*

$$\mathcal{V} = \mathcal{K}_m(\mathbf{A}, \mathbf{v}) = \text{span}(\mathbf{v}, \mathbf{Av}, \ldots, \mathbf{A}^{m-1}\mathbf{v}).$$

The following properties can easily be verified.

**Lemma 1.** *Define* $\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ *and let* $m$ *be a nonzero integer. It holds that*

1. *Scale and shift invariant:*

$$\mathcal{K}_m(\mathbf{A}, \mathbf{v}) = \mathcal{K}_m(\alpha\mathbf{A}, \beta\mathbf{v}), \alpha, \beta \neq 0.$$

$$\mathcal{K}_m(\mathbf{A}, \mathbf{v}) = \mathcal{K}_m(\mathbf{A} + \rho\mathbf{I}, \mathbf{v})$$

2. *Change of basis: Let* $\mathbf{X} \in \mathbb{R}^{n\times n}$ *be nonsingular,*

$$\mathcal{K}_m(\mathbf{A}, \mathbf{v}) = \mathbf{X}\mathcal{K}_m(\mathbf{X}^{-1}\mathbf{AX}, \mathbf{X}^{-1}\mathbf{v}).$$

3. *Expansion:*

$$\mathbf{A}\mathcal{K}_m(\mathbf{A}, \mathbf{v}) \subset \mathcal{K}_{m+1}(\mathbf{A}, \mathbf{v}).$$

## 2.2 Tensors

The last concept we need to introduce concerns tensors. Also on this topic, there exists a vast amount of books and papers. We only consider the material that is needed for understanding this thesis. The first part is devoted to introducing the basic concepts and introducing various kind of operations that can be performed on a tensor. In the second part of this section, we introduce the Tensor-Train representation of a tensor.

In essence, a tensor is nothing more than a multidimensional array $\mathcal{X} \in \mathbb{C}^{n_1\times n_2\times\ldots\times n_d}$. For slicing a matrix or tensor, we use Matlab notation. Note that if only one index is left free, we interpret the slice as a column vector. If two indices are left free, the slice is a matrix whose row index corresponds to the first free tensor index and the column to the second free tensor index. For example, let $\mathcal{X}$ be a $n_1 \times n_2 \times n_3$-tensor then $\mathcal{X}(i, :, k)$ is considered a column vector and $\mathcal{X}(:, j, :)$ is a $n_1 \times n_3$ matrix.

[3]Aleksey Nikolaevich Krylov (1863-1945) [76] was a naval engineer and applied mathematician. He wrote about 300 papers and scientific books. There is a crater on the moon named after him.

Before we explain the basic concepts, we explain how tensors can be used in the context of calculating extreme eigenvalues of parametrized matrix pairs and when calculating eigenvalue-tuples of multiparameter eigenvalue problem. When solving parametrized eigenvalue problems, we make the assumption that the eigenvectors associated to the wanted eigenvalue can be represented using a small subspace. Let $\Omega = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ be the parameter space, then a set of samples can be

$$\Omega_0 = \Omega_1^0 \times \Omega_2^0 \times \ldots \times \Omega_d^0,$$

where

$$\Omega_i^0 = \{a_i + (j-1)h_i | j = 1, 2, \ldots, n_i\}, \quad h_i = \frac{b_i - a_i}{n_i - 1}. \tag{2.5}$$

This is a set of $n_1 n_2 \ldots n_d$ points. Define a tensor $\mathcal{X} \in \mathbb{C}^{n \times n_1 \times \ldots \times n_d}$ with $n$ the dimension of the matrices in the matrix pencil, then we can store approximate eigenvectors $\hat{\mathbf{x}}(\omega_{i_1}, \ldots, \omega_{i_d})$ in $\mathcal{X}$ via

$$\mathcal{X}(:, i_1, \ldots, i_d) = \hat{\mathbf{x}}(\omega_{i_1}, \ldots, \omega_{i_d}).$$

We discuss in Section 2.2.2 how to decompose a tensor in the Tensor-Train format.

### 2.2.1 Basic concepts and operations

This section is based on [37]. To clarify the different concepts in this section, we apply them on the tensor $\mathcal{X} \in \mathbb{R}^{4 \times 3 \times 2}$, defined by

$$\mathcal{X}(:,:,1) = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}, \quad \mathcal{X}(:,:,2) = \begin{bmatrix} 13 & 17 & 21 \\ 14 & 18 & 22 \\ 15 & 19 & 23 \\ 16 & 20 & 24 \end{bmatrix}. \tag{2.6}$$

The tensor in (2.6) has order three. What a row or column is in the context of matrices, is called a *fiber* in the context of tensor. This means that fibers are constructed by holding all modes except one fixed. For the example in (2.6) there are three types of fiber: The mode-1 fibers are

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

and the mode-2 fibers are

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

The mode-3 fibers are defined analogously.

Before we can define operations on tensors, we need the concept of *matricization*. In the literature, the term *unfolding* or *flattening* is used for this. This means that we associate with the tensor a matrix by taking together nodes. More formally speaking, the matricization $\mathbf{X}_{(j)}$ of a tensor $\mathcal{X}$ is the matrix formed by all mode-$j$ fibers. The matricization takes the element on position $(i_1, i_2, ..., i_d)$ and maps it on the element $(i_j, k)$ with

$$k = 1 + \sum_{\substack{p=1\\p\neq j}}^{d} (i_p - 1)\, J_p \text{ and } J_p = \prod_{\substack{m=1\\m\neq j}}^{p-1} n_m.$$

This ordering becomes clear if we go back to our example in (2.6). The three different matricizations are

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

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

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

We now have all the tools to define the *$j$-mode product* between a tensor and a matrix (or vector). As the name suggests, we multiply a matrix with the $j$th mode of the tensor. The operation is formally defined as

$$\mathcal{Y} = (\mathbf{I}_{n_1}, \dots, \mathbf{I}_{n_{j-1}}, \mathbf{A}, \mathbf{I}_{n_{j+1}}, \dots, \mathbf{I}_{n_d}).\mathcal{X} \Leftrightarrow \mathbf{Y}_{(j)} = \mathbf{A}\mathbf{X}_{(j)}.$$

for a certain $\mathbf{A} \in \mathbb{R}^{m\times n_j}$.

Further we define the *Hadamard*[4]-product between two tensors as the elementwise multiplication. So if we have tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$, then $\mathcal{Z} = \mathcal{X}._H \mathcal{Y}$ is defined by

$$\mathcal{Z}(i_1, i_2, \ldots, i_d) = \mathcal{X}(i_1, i_2, \ldots, i_d)\mathcal{Y}(i_1, i_2, \ldots, i_d).$$

To conclude this section, we define the left *Kronecker*[5]*-product*. It is a product between matrices $\mathbf{A} \in \mathbb{R}^{n_a \times m_a}$ and $\mathbf{B} \in \mathbb{R}^{n_b \times m_b}$, but is closely related to tensors. The left-Kronecker-product is defined as

$$\mathbf{Z} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{B}(1,1)\mathbf{A} & \mathbf{B}(1,2)\mathbf{A} & \ldots & \mathbf{B}(1,m_b)\mathbf{A} \\ \mathbf{B}(2,1)\mathbf{A} & \mathbf{B}(2,2)\mathbf{A} & \ldots & \mathbf{B}(2,m_b)\mathbf{A} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}(n_b,1)\mathbf{A} & \mathbf{B}(n_b,2)\mathbf{A} & \ldots & \mathbf{B}(n_b,m_b)\mathbf{A} \end{bmatrix} \in \mathbb{R}^{n_a n_b \times m_a m_b}.$$

The reason why we use the left-Kronecker product and not the more commonly used right-Kronecker product, is that in the former there is a more clear correspondence with the Tensor-Train decomposition, introduced in what follows.

### 2.2.2 Tensor-Train decomposition

Tensors can be used to represent very large data-sets (the number of elements grows exponentially in the number of modes). In many applications, tensor representations and tensor decompositions can be used to reveal structure and dependencies in the data, which allows to significantly reduce the storage requirements (i.e. if the tensor can be well approximated by a limited number of terms in the decomposition). Furthermore the operations as defined in the previous section can be performed directly within this format.

There are many ways to decompose tensors. The most famous are the *CP decomposition* [10, 28] and the *Tucker*[6] *decomposition* [73]. In this section we discuss the *Tensor-Train decomposition* (TT-decomposition) [50]. We first describe how the Tensor-Train decomposition is defined, then we explain how

[4]Jacques Solomon Hadamard (1865–1963) [86] was a French mathematician. He wrote a book "Psychology of invention in the mathematical field" where he described his mathematical thought processes and where he surveyed other mathematicians and theoretical physicists about their thought processes

[5]Leopold Kronecker (1823–1891) [90] was a German mathematician. His research was situated in the field of number theory, algebra and logic. He said once: "God made the integers, all else is the work of men". There is an asteroid named after him.

[6]Ledyard R. Tucker (1910–2004) [89] was an American mathematician whose main research topics were statistics and psychometrics.

such a decomposition can be calculated and we clarify why this decomposition is chosen above other decompositions.

**Definition 2.** *A tensor* $\mathcal{X} \in \mathbb{C}^{n_1 \times \ldots \times n_d}$ *is in* TT-format *if*

$$\mathcal{X}(i_1, \ldots, i_d) = \mathbf{G}_1^X(i_1) \ldots \mathbf{G}_m^X(i_d), \quad i_k = 1, \ldots, n_k, \quad k = 1, \ldots d \tag{2.7}$$

*with* $\mathbf{G}_k^X(i_k)$ *a* $r_{k-1}^X \times r_k^X$ *matrix. The length* $r_{k-1}^X$ *and width* $r_k^X$ *of these matrices* $r_k^X, k = 0, \ldots, d$, *are called the* TT-ranks. *It can immediately be observed that* $r_0^X = r_d^X = 1$. *We call* $r_k^X, k = 1, \ldots, d-1$, *the* interior *ranks. The matrices* $\mathbf{G}_k^X$ *can be stored in a tensor* $\mathcal{G}_k^X \in \mathbb{C}^{r_{k-1}^X \times n_k \times r_k^X}, k = 1, \ldots, d$, *as* $\mathcal{G}_k^X(:, i_k, :) = \mathbf{G}_k^X(i_k), i_k = 1, \ldots, n_k$. *The tensor* $\mathcal{G}_k^X, k = 1, \ldots, d$, *is called a* TT-core. *If it is clear from the context from which tensor the decomposition is made, we omit the superscript.*

We associate with a vector $\mathbf{x} \in \mathbb{C}^{n_1 n_2 \ldots n_d}$ a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times n_2 \times \ldots \times n_d}$ such that

$$\mathrm{vec}(\mathcal{X}) = \mathbf{x}.$$

This means that for every $i = 1, \ldots, (n_1 n_2 \ldots n_d)$, there is a unique $(i_1, i_2, \ldots, i_d)$ such that

$$\mathbf{x}(i) = \mathcal{X}(i_1, i_2, \ldots, i_d). \tag{2.8}$$

We say that a vector $\mathbf{x} \in \mathbb{C}^{n_1 n_2 \ldots n_d}$ is in TT-format if its associated tensor $\mathcal{X}$ is in TT-format. We index the vector $\mathbf{x}$ with indices $(i_1, \ldots, i_d)$ of its associated tensor. This means that

$$\mathbf{x}(i) = \mathcal{X}(i_1, \ldots, i_d) = \mathbf{G}_1^X(i_1) \ldots \mathbf{G}_d^X(i_d).$$

**Definition 3.** *The matrix* $\mathbf{X} \in \mathbb{C}^{n_1 n_2 \ldots n_d \times m}$ *is in* block-TT format *with index* $k$ *if*

$$\mathbf{X}(i, i_m) = \mathbf{G}_1(i_1) \ldots \mathbf{G}_{k-1}(i_{k-1}) \hat{\mathbf{G}}_k(i_k, i_m) \mathbf{G}_{k+1}(i_{k+1}) \ldots \mathbf{G}_d(i_d) \tag{2.9}$$

*for a certain* $k = 1, \ldots, d$. *The matrix* $\hat{\mathbf{G}}_k(i_k, i_m)$ *is stored in a tensor* $\mathcal{G}_k$ *of size* $r_{k-1}^X \times n_k \times m \times r_k^X$.

Definition 3 means that we make a TT-decomposition of every column in $\mathbf{X}$ (seen as a tensor) such that all cores except one are in common. Definitions 2 and 3 may look difficult to interpret. To make it more clear, we have the following property which follows from the definition of Tensor-Trains.

**Property 5.** *1. Let* $\mathcal{X} \in \mathbb{C}^{n_1 \times n_2 \times \ldots \times n_d}$ *be a tensor in TT-format* (2.7) *and* $\mathbf{x} = vec(\mathcal{X}) \in \mathbb{C}^{n_1 n_2 \ldots n_d}$, *then it holds that*

$$\mathbf{x} = \sum_{j_1, \ldots, j_{d-1}=1}^{r_1, \ldots, r_{d-1}} \mathcal{G}_1(1, :, j_1) \otimes \mathcal{G}_2(j_1, :, j_2) \otimes \ldots \otimes \mathcal{G}_d(j_{d-1}, :, 1).$$

*2. Let matrix* $\mathbf{X} \in \mathbb{C}^{n_1 n_2 \dots n_d \times m}$ *be in* block-TT format *with index* $k$*, then*

$$\mathbf{X}(:, i_m) = \sum_{j_1,\dots,j_{d-1}=1}^{r_1,\dots,r_{d-1}} \mathcal{G}_1(1, :, j_1) \otimes \dots \otimes \mathcal{G}_{k-1}(j_{k-2}, :, j_{k-1}) \otimes \mathcal{G}_k(j_{k-1}, :, i_m, j_k) \otimes \mathcal{G}_{k+1}(j_k, :, j_{k+1}) \otimes \dots \otimes \mathcal{G}_d(j_{d-1}, :, 1).$$

Like we could make a TT-decomposition of a vector by seeing it as a tensor of appropriate dimension (see (2.8)), we can do the same for a matrix. In this way, it is possible to do a matrix-vector product where both are presented in their TT-decomposition. Let $\mathbf{M}$ be a matrix of size $n_1 \dots n_d \times n_1 \dots n_d$. Elements of $\mathbf{M}$ can be indexed by $2d$-tuples $(i_1, \dots, i_d, j_1, \dots, j_d)$ where $(i_1, \dots, i_d)$ resp. $(j_1, \dots, j_d)$ is the index for the rows resp. the columns. Strictly speaking, the dimensions for the rows and columns do not need to be equal, but in order not to overload the notation and as we only need this case in this thesis, we choose them here to be equal. We say that $\mathbf{M}$ is in TT-format if

$$\mathbf{M}(i_1, \dots, i_d, j_1, \dots, j_d) = \mathbf{G}_1^M(i_1, j_1) \dots \mathbf{G}_d^M(i_d, j_d) \tag{2.10}$$

with $\mathbf{G}_k^M(i_k, j_k)$ a $r_{k-1}^M \times r_k^M$ matrix. The matrices $\mathbf{G}_k^M$ can be stored in a tensor $\mathcal{G}_k^M \in \mathbb{C}^{r_{k-1}^M \times n_k \times n_k \times r_k^M}$. Let $\mathbf{x}$ be a vector in TT-format of size $n_1 n_2 \dots n_d$ and consider the product $\mathbf{z} = \mathbf{M}\mathbf{x}$, then

$$\begin{aligned}
\mathbf{z}(i_1, \dots, i_d) &= \sum_{j_1,\dots,j_m} \mathbf{M}(i_1, \dots, i_m, j_1, \dots, j_d)\mathbf{x}(j_1, \dots, j_d) \\
&= \sum_{j_1,\dots,j_d} \mathbf{G}_1^M(i_1, j_1) \dots \mathbf{G}_m^M(i_d, j_d)\mathbf{G}_1^X(j_1) \dots \mathbf{G}_m^X(j_d) \\
&= \sum_{j_1,\dots,j_d} (\mathbf{G}_1^M(i_1, j_1) \otimes \mathbf{G}_1^X(j_1)) \dots (\mathbf{G}_1^M(i_d, j_d) \otimes \mathbf{G}_d^X(j_d)) \\
&= \left( \sum_{j_1} \mathbf{G}_1^M(i_1, j_1) \otimes \mathbf{G}_1^X(j_1) \right) \left( \sum_{j_2} \mathbf{G}_2^M(i_2, j_2) \otimes \mathbf{G}_2^X(j_2) \right) \dots \\
&\quad \left( \sum_{j_d} \mathbf{G}_d^M(i_d, j_d) \otimes \mathbf{G}_d^X(j_d) \right).
\end{aligned}$$

By defining $\mathbf{G}_k^Z(i_k) = \sum_{j_k} \mathbf{G}_k^M(i_k, j_k) \otimes \mathbf{G}_k^X(j_k)$, we observe that the TT-format of the matrix-vector product can directly be derived from the TT-format of its components.

The question now arises how we can make a TT-decomposition given a full tensor. In [50, Theorem 2.1.] a constructive algorithm to make a TT-decomposition is stated. If $d = 2$, it is proven in [15] that the optimal rank-$r$ approximation is obtained by retaining the first $r$ terms from its singular value decomposition (SVD). For $d > 2$ such a theorem does not exist, but this idea is nevertheless generalised for decomposing a full tensor. The different TT-cores are constructed by performing $d - 1$ consecutive SVDs. We omit here the proof and state only the constructive algorithm in pseudo-code in Algorithm 2.

**Algorithm 2** [50, Algorithm 1] TT-SVD

**Input:** Tensor $\mathcal{X}$, accuracy $\epsilon$
**Output:** Tensor $\mathcal{Y}$ in TT-format such that

$$\|\mathcal{X} - \mathcal{Y}\|_F \leq \epsilon \|\mathcal{X}\|_F$$

1: $\mathcal{Z} = \mathcal{X}$; $r_0 = 1$
2: $\delta = \epsilon/\sqrt{d-1}\,\|\mathcal{X}\|_F$
3: **for** $k = 1, \ldots, d-1$ **do**
4: $\quad \mathbf{Z} =$ reshape $\mathcal{Z}$ to a matrix of size $r_{k-1}n_k \times \texttt{numel}(\mathcal{Z})/(r_{k-1}n_k)$
5: $\quad$ Compute truncated SVD: $\mathbf{Z} = \mathbf{USV}^T + \mathbf{E}, \|\mathbf{E}\|_F = \delta, r_k$ is the number of columns in $\mathbf{U}$
6: $\quad \mathcal{G}_k =$ reshape $\mathbf{U}$ to a tensor of size $r_{k-1} \times n_k \times r_k$
7: $\quad \mathbf{Z} = \mathbf{SV}^T$
8: **end for**
9: $\mathcal{G}_d = \mathbf{Z}$
10: $\mathcal{Y}$ has TT-decomposition $\mathcal{G}_1, \ldots, \mathcal{G}_d$.

In the next section we discuss how the various operations on tensors are performed
directly adapting the TT-cores. As a consequence, it is possible that the ranks are overestimated. Therefore it is advised to round its TT-representation before performing other operations. Algorithm 3 is used for rounding a tensor. The complexity of this algorithm is $\mathcal{O}(dnr^3)$.

We come back to the storage issue, if we have $n_i \sim n, i = 1, \ldots, d$ and we let $r = \max_{i=1,\ldots,d} r_i$, then the total needed storage is of order $\mathcal{O}(r^2nd)$. So it is clear that the lower the ranks are, the less storage is needed.

### 2.2.3 Operations

In this section we discuss how certain frequently-used operations are implemented when working with TT-tensors. Let $\mathcal{X}$ and $\mathcal{Y}$ be two tensors

**Algorithm 3** [50, Algorithm 2] TT-rounding

**Input:** Tensor $\mathcal{X}$ with overestimated ranks, accuracy $\epsilon$
**Output:** Tensor $\mathcal{Y}$ in TT-format such that

$$\|\mathcal{X} - \mathcal{Y}\|_F \leq \epsilon \|\mathcal{X}\|_F$$

with $\mathcal{Y}$ having the the real ranks.

$\delta = \epsilon/\sqrt{d-1}\,\|A\|_F$
**for** $k = d, d-1, \ldots, 2$ **do**
  $\mathbf{G} =$ reshape $\mathcal{G}_k$ to a matrix of size $r_{k-1} \times n_k r_k$
  $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{G}^T)$
  $\mathcal{G}_k =$ reshape $\mathbf{Q}^T$ to a tensor of size $r_{k-1} \times n_k \times r_k$
  $\mathbf{G}_{k-1} =$ reshape $\mathcal{G}_{k-1}$ to a matrix of size $r_{k-2} n_{k-1} \times r_{k-1}$
  $\mathcal{G}_{k-1} =$ reshape $\mathcal{G}_{k-1}\mathbf{R}$ to a tensor of size $r_{k-2} \times n_{k-1} \times r_{k-1}$
**end for**
**for** $k = 1, 2, \ldots, d-1$ **do**
  $\mathbf{G} =$ reshape $\mathcal{G}_k$ to $r_{k-1} n_k \times r_k$
  Compute the truncated SVD $\mathbf{G} = \mathbf{U}\mathbf{S}\mathbf{V}^T + \mathbf{E}, \|\mathbf{E}\|_F \leq \delta, r_k =$ number of columns in $\mathbf{U}$
  $\mathbf{G}_k = \mathbf{U}, \mathcal{G}_{k+1} = \left(\mathbf{S}\mathbf{V}^T, \mathbf{I}, \mathbf{I}\right) \mathcal{G}_{k+1}$
**end for**
$\mathcal{Y}$ has TT-decomposition $\mathcal{G}_1, \ldots, \mathcal{G}_d$

of the same size in TT-format and define $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$. We examine how the cores of $\mathcal{Z}$ depend on the cores of $\mathcal{X}$ and $\mathcal{Y}$. The cores $\mathcal{G}^Z$ need to fulfil the following condition

$$\begin{aligned} \mathcal{Z}(i) &= \mathbf{G}_1^Z(i_1)\mathbf{G}_2^Z(i_2)\ldots\mathbf{G}_d^Z(i_d) \\ &= \mathbf{G}_1^X(i_1)\mathbf{G}_2^X(i_2)\ldots\mathbf{G}_d^X(i_d) + \mathbf{G}_1^Y(i_1)\mathbf{G}_2^Y(i_2)\ldots\mathbf{G}_d^Y(i_d). \end{aligned} \tag{2.11}$$

It can be verified that if we define

$$\mathbf{G}_1^Z(i_1) = [\mathbf{G}_1^X(i_1), \mathbf{G}_1^Y(i_1)], \qquad \mathbf{G}_d^Z = [\mathbf{G}_d^X(i_d), \mathbf{G}_d^Y(i_d)]^T,$$

$$\mathbf{G}_k^Z(i_k) = \begin{bmatrix} \mathbf{G}_k^X(i_k) & 0 \\ 0 & \mathbf{G}_k^Y(i_k) \end{bmatrix}, \qquad k = 2, \ldots, d-1.$$

then these cores meet condition (2.11). The multiplication with a constant $a$ is not more than the multiplication of one of the coress with $a$.

The next operation that is used in what follows, is the *Hadamard product* or elementwise multiplication. If we define $\mathcal{Z} = \mathcal{X}.{}_H\mathcal{Y}$, then it follows that

$$\begin{aligned}
\mathcal{Z}(i_1, i_2, \ldots, i_d) &= \mathcal{X}(i_1, i_2, \ldots, i_d)\mathcal{Y}(i_1, i_2, \ldots, i_d) \\
&= \mathbf{G}_1^X(i_1)\mathbf{G}_2^X(i_2)\ldots\mathbf{G}_d^X(i_d)\mathbf{G}_1^Y(i_1)\mathbf{G}_2^Y(i_2)\ldots\mathbf{G}_d^Y(i_d) \\
&= \left(\mathbf{G}_1^X(i_1)\mathbf{G}_2^X(i_2)\ldots\mathbf{G}_d^X(i_d)\right) \otimes \left(\mathbf{G}_1^Y(i_1)\mathbf{G}_2^Y(i_2)\ldots\mathbf{G}_d^Y(i_d)\right) \\
&= \left(\mathbf{G}_1^X(i_1) \otimes \mathbf{G}_1^Y(i_1)\right)\left(\mathbf{G}_2^X(i_2) \otimes \mathbf{G}_2^Y(i_2)\right)\ldots \qquad (2.12) \\
&\quad \left(\mathbf{G}_d^X(i_d) \otimes \mathbf{G}_d^Y(i_d)\right).
\end{aligned}$$

It follows that the cores of $\mathcal{Z}$ are

$$\mathbf{G}_k^Z(i_k) = \mathbf{G}_k^X(i_k) \otimes \mathbf{G}_k^Y(i_k), \qquad k = 1, \ldots, d.$$

The last operation we discuss is the scalar product, used in for example the calculation of the Frobenius-norm of a tensor. This scalar product for tensors, is in comparison with (1.3), an extension (tensors instead of vectors) and on the other hand a restriction (the case $\mathbf{A} = \mathbf{I}$ in (1.3)). It is defined as

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1,\ldots,i_d=1}^{n_1,\ldots,n_d} \mathcal{X}(i_1, \ldots, i_d)\mathcal{Y}(i_1, \ldots, i_d). \qquad (2.13)$$

If follows that

$$\begin{aligned}
\langle \mathcal{X}, \mathcal{Y} \rangle &= \sum_{i_1,\ldots,i_d=1}^{n_1,\ldots,n_d} \left(\mathbf{G}_1^X(i_1) \otimes \mathbf{G}_1^Y(i_1)\right)\left(\mathbf{G}_2^X(i_2) \otimes \mathbf{G}_2^Y(i_2)\right)\ldots \qquad (2.14) \\
&\quad \left(\mathbf{G}_d^X(i_d) \otimes \mathbf{G}_d^Y(i_d)\right) \\
&= \sum_{i_1=1}^{n_1} \left(\mathbf{G}_1^X(i_1) \otimes \mathbf{G}_1^Y(i_1)\right) \sum_{i_2=1}^{n_2} \left(\mathbf{G}_2^X(i_2) \otimes \mathbf{G}_2^Y(i_2)\right)\ldots \\
&\quad \sum_{i_d=1}^{n_d} \left(\mathbf{G}_d^X(i_d) \otimes \mathbf{G}_d^Y(i_d)\right) \\
&=: \mathbf{\Gamma}_1\mathbf{\Gamma}_2\ldots\mathbf{\Gamma}_d,
\end{aligned}$$

where (2.14) is obtained from (2.12).

In practice, we calculate $\langle \mathcal{X}, \mathcal{Y} \rangle$ sequentially via the matrix-vector product

$$\begin{aligned} \mathbf{v}_k &= \mathbf{v}_{k-1} \mathbf{\Gamma}_k \\ &= \mathbf{v}_{k-1} \sum_{i_k} \left( \mathbf{G}_k^X(i_k) \otimes \mathbf{G}_k^Y(i_k) \right) \end{aligned}$$

for $k = 2, \ldots, d$ where $\mathbf{v}_1 = \mathbf{\Gamma}_1$ is a row-vector by construction. Then it follows that $\mathbf{v}_d = \langle \mathcal{X}, \mathcal{Y} \rangle$. If the ranks of both $\mathcal{X}$ and $\mathcal{Y}$ are of order $r$, then it is proven in [50] that the total number of operations is of order $\mathcal{O}(dnr^3)$.

### 2.2.4 Comparison with other decompositions

In this subsection we motivate why the TT-decomposition is a good option for the problems addressed in this thesis and why we prefer this decomposition above other decompositions. We compare the TT-decomposition with the CP-decomposition and the Tucker decomposition.

The main advantage of the TT-format with respect to the Tucker decomposition is in terms of storage. Simply put, in the Tucker decomposition one holds a basis for each mode (so storage cost of $\mathcal{O}(rnd)$) and a core of size $r^d$, so the storage of the core is exponential with the order (the number of modes). This is in contrast with the TT-format which has a total storage of $\mathcal{O}(nr^2d)$ which scales better for high $d$.

In the CP-decomposition a tensor is approximated by a sum of $r$ rank-1 tensors. This $r$ is called the *tensor rank* in this context. The problem here is that some of the computations are more difficult. For example, calculating a best CP-approximation of tensor rank $r$ is still a problem for which no algorithm exists that always provides adequate results [25].

# Chapter 3

# Calculating the minimal eigenvalue of symmetric parametrized matrix pairs

*Doubt is the beginning,*
*not the end, of wisdom.*
*(George Iles)*

Parts of this chapter are copied from [60]:

RUYMBEEK K., MEERBERGEN K., AND MICHIELS W.
"Calculating the minimal/maximal eigenvalue of symmetric parameterized matrices using projection." Numerical Linear Algebra with Applications 26.5 (2019): e2263.

## 3.1 Introduction

Let $\mathbf{A}(\omega), \mathbf{B}(\omega)$ be $n \times n$ real symmetric matrices which smoothly depend on a parameter $\omega$ in a compact subset $\Omega \subset \mathbb{R}^d$. We impose that $\mathbf{B}(\omega)$ is positive definite, for $\omega \in \Omega$, so all eigenvalues $\lambda(\omega)$ of the generalized eigenvalue problem $(\mathbf{A}(\omega), \mathbf{B}(\omega))$,

$$\mathbf{A}(\omega)\mathbf{x}(\omega) = \lambda(\omega)\mathbf{B}(\omega)\mathbf{x}(\omega),$$

are real, see, e.g, [61, Ch. 9].

The objective is to calculate in an efficient way, an accurate global approximation of the minimum eigenvalue $\lambda_1(\omega)$ over the whole parameter space. As we already stated in the previous chapter, we apply a projection method on this problem. This means we build up a subspace $\mathcal{V}$, we project the matrix pencil on this subspace and we estimate the minimum eigenvalue by the minimum eigenvalue of the problem projected on this subspace.

The question arises how we best build up this subspace. Eigenvectors for specific sample points in the parameter space form the basic ingredient. The sample points are calculated based on a greedy algorithm [56] by selecting the sample point with the maximum residual. As calculating the minimum eigenpair can be expensive for large matrices, it is beneficial to use as few sample points as needed. Therefore we research if there is other information freely available to further enrich the subspace and to reduce the number of sample points. We study higher order Hermite interpolation sampling by inserting partial derivatives of the eigenvectors towards the parameters in the subspace and the use of more than one eigenvector approximation in each sample point. We note that calculating an eigenvalue that meets another condition (maximum, second maximum, second minimum,...) can be done similarly. Without loss of generality, we assume that the eigenvalues and their associated eigenvectors are sorted in ascending order for each parameter value.

This chapter is organised as follows. We give an overview of the main properties about derivatives of eigenvectors in Section 3.2. In Section 3.3, we explain our algorithm. We start with the main ideas, then we go over the used theoretical results and the algorithm itself. We illustrate the theory and the algorithm numerically in Section 3.4. We extend our algorithm such that it can calculate the mean of the minimum eigenvalue in Section 3.5. In Section 3.6 we conclude this chapter. The Matlab code can be downloaded from

`http://twr.cs.kuleuven.be/research/software/delay-control/parmat/`.

## 3.2 Derivatives of eigenvalues and eigenvectors

As we want to approximate the minimum eigenvalue, it is necessary to understand how eigenvalues and eigenvectors depend on the parameters. This is also used to proof the Hermite interpolation property in the next section. Before we can start, we need a characterisation of the first and second derivative of an eigenvalue and of the first derivative of an eigenvector. Since eigenvectors

depend on the normalisation, we first outline the adopted setting in which they are uniquely defined. Let $\lambda_i^*$ be a simple eigenvalue with associated eigenvector $\mathbf{x}_i^*$ of $\omega^* \in \Omega$ then there is a neighbourhood $\Omega^*$ around $\omega^*$ and smooth functions

$$\begin{array}{ccc} \lambda_i : \Omega^* \to \mathbb{R} & & \mathbf{x}_i : \Omega^* \to \mathbb{R}^n \\ \omega \mapsto \lambda_i(\omega) & \text{and} & \omega \mapsto \mathbf{x}_i(\omega) \end{array}. \tag{3.1}$$

such that

$$\begin{cases} \mathbf{A}(\omega)\mathbf{x}_i(\omega) - \lambda_i(\omega)\mathbf{B}(\omega)\mathbf{x}_i(\omega) = 0, & \forall \omega \in \Omega^*, \\ & ((\lambda_i(\omega), \mathbf{x}_i(\omega)) \text{ is eigenpair}) \\ \mathbf{x}_i(\omega)^T \mathbf{B}(\omega)\mathbf{x}_i(\omega) = 1, & \forall \omega \in \Omega^*, \\ & (\text{ normalisation condition}) \\ \lambda_i(\omega^*) = \lambda_i^*, & \\ \mathbf{x}_i(\omega^*) = \mathbf{x}_i^*. & \end{cases} \tag{3.2}$$

In the next theorem we prove analytical formulas to calculate partial derivatives of an eigenvalue and give an implicit formulation of the partial derivative of an eigenvector. In the rest of this section we write $\mathbf{A}, \mathbf{B}, \lambda$ resp. $\mathbf{x}$ instead of $\mathbf{A}(\omega), \mathbf{B}(\omega), \lambda(\omega)$ resp. $\mathbf{x}(\omega)$ to increase readability.

**Theorem 1.** *Let $(\lambda_i, \mathbf{x}_i)$ be a simple eigenpair of $(\mathbf{A}, \mathbf{B})$, then it holds that*

*1.*

$$\frac{\partial \lambda_i}{\partial \omega_j} = \mathbf{x}_i^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{x}_i, \quad j = 1, \ldots, d. \tag{3.3}$$

*2.*

$$\frac{\partial^2 \lambda_i}{\partial \omega_j \, \partial \omega_k} = 2\mathbf{x}_i^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \frac{\partial \mathbf{x}_i}{\partial \omega_k} + \ldots \tag{3.4}$$

$$\mathbf{x}_i^T \left( \frac{\partial^2 \mathbf{A}}{\partial \omega_j \, \partial \omega_k} - \frac{\partial \lambda_i}{\partial \omega_k} \frac{\partial \mathbf{B}}{\partial \omega_j} - \lambda_i \frac{\partial^2 \mathbf{B}}{\partial \omega_j \, \partial \omega_k} \right) \mathbf{x}_i, \quad j, k = 1, \ldots, d.$$

*3. The derivative of the eigenvector $\mathbf{x}_i$ associated with the simple eigenvalue $\lambda_i$ is characterized as the solution of the following system:*

$$\begin{bmatrix} \lambda_i \mathbf{B} - \mathbf{A} & \mathbf{B}\mathbf{x}_i \\ \mathbf{x}_i^T \mathbf{B} & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial \mathbf{x}_i}{\partial \omega_j} \\ \frac{\partial \lambda_i}{\partial \omega_j} \end{bmatrix} = \begin{bmatrix} \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{x}_i \\ -\frac{\mathbf{x}_i^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{x}_i}{2} \end{bmatrix}, \quad j = 1, \ldots, d. \tag{3.5}$$

*Proof.* 1. This is a generalisation of the more known result for standard eigenvalue problems, see, e.g., [39, Ch. 9].

2. It is sufficient to differentiate (3.3).

3. This is a generalisation of the result in [20]. We start by differentiating $(\lambda_i \mathbf{B} - \mathbf{A})\,\mathbf{x}_i = 0$ to get

$$\left(\frac{\partial \lambda_i}{\partial \omega_j}\mathbf{B} + \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} - \frac{\partial \mathbf{A}}{\partial \omega_j}\right)\mathbf{x}_i + (\lambda_i \mathbf{B} - \mathbf{A})\,\frac{\partial \mathbf{x}_i}{\partial \omega_i} = 0, \quad j = 1, \ldots, d.$$

As $(\lambda_i \mathbf{B} - \mathbf{A})$ is a singular matrix,

$$(\lambda_i \mathbf{B} - \mathbf{A})\,\frac{\partial \mathbf{x}_i}{\partial \omega_j} = -\left(\frac{\partial \lambda_i}{\partial \omega_j}\mathbf{B} + \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} - \frac{\partial \mathbf{A}}{\partial \omega_j}\right)\mathbf{x}_i, \quad j = 1, \ldots, d, \quad (3.6)$$

is not sufficient to characterise the partial derivative of an eigenvector. The missing information is the information in the direction $\mathbf{x}_i$. We obtain information about this direction by taking the derivative of the normalisation condition in (3.2). This results in

$$\mathbf{x}_i^T \mathbf{B} \frac{\partial \mathbf{x}_i}{\partial \omega_j} = -\frac{1}{2}\mathbf{x}_i^T \frac{\partial \mathbf{B}}{\partial \omega_j}\mathbf{x}_i, \quad j = 1, \ldots, d. \quad (3.7)$$

□

The derivative of an eigenvector is implicitly given as the solution of the system in Theorem 1. Following result starts from this theorem and derives an analytical solution for the derivative of an eigenvector. This result can also be found in [63], but we give our proof to show the connection with the previous theorem.

**Theorem 2.** *If $\mathbf{x}_i$ is an eigenvector associated with a simple eigenvalue $\lambda_i$, then we have*

$$\frac{\partial \mathbf{x}_i}{\partial \omega_j} = -\frac{\mathbf{x}_i^T \frac{\partial \mathbf{B}}{\partial \omega_j}\mathbf{x}_i}{2}\mathbf{x}_i + \sum_{k=1, k\neq i}^{n}\left(\mathbf{x}_k^T \frac{\left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j}\right)\mathbf{x}_i}{\lambda_i - \lambda_k}\right)\mathbf{x}_k, \quad j = 1, \ldots, d. \quad (3.8)$$

*Proof.* Without loss of generality, we prove the statement for $i = 1$. From the eigendecomposition in (1.5) it follows immediately that system (3.5) can be written as

$$\left[\begin{array}{c|c} \lambda_1 \left(\mathbf{X}_\lambda^{-1}\right)^T \mathbf{X}_\lambda^{-1} - \left(\mathbf{X}_\lambda^{-1}\right)^T \mathbf{\Lambda}_{\mathrm{diag}} \mathbf{X}_\lambda^{-1} & \left(\mathbf{X}_\lambda^{-1}\right)^T \mathbf{e}_1 \\ \mathbf{e}_1^T \mathbf{X}_\lambda^{-1} & 0 \end{array}\right] \left[\begin{array}{c} \frac{\partial \mathbf{x}_1}{\partial \omega_j} \\ \frac{\partial \lambda_1}{\partial \omega_j} \end{array}\right] = \left[\begin{array}{c} \left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right)\mathbf{x}_1 \\ -\frac{\mathbf{x}_1^T \frac{\partial \mathbf{B}}{\partial \omega_j}\mathbf{x}_1}{2} \end{array}\right].$$

Using $\mathbf{X}_\lambda \mathbf{X}_\lambda^T \mathbf{B} = \mathbf{I}$, we obtain

$$\left[\begin{array}{c|c} \lambda_1 \left(\mathbf{X}_\lambda^{-1}\right)^T - \left(\mathbf{X}_\lambda^{-1}\right)^T \mathbf{\Lambda}_{\text{diag}} & \left(\mathbf{X}_\lambda^{-1}\right)^T \mathbf{e}_1 \\ \hline \mathbf{e}_1^T & 0 \end{array}\right] \left[\begin{array}{c} \mathbf{X}_\lambda^T \mathbf{B} \frac{\partial \mathbf{x}_1}{\partial \omega_j} \\ \frac{\partial \lambda_1}{\partial \omega_j} \end{array}\right] = \left[\begin{array}{c} \left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right) \mathbf{x}_1 \\ -\frac{\mathbf{x}_1^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{x}_1}{2} \end{array}\right].$$

By multiplying from the left with the non-singular matrix $\begin{bmatrix} \mathbf{X}_\lambda^T & \mathbf{0} \\ 0 & 1 \end{bmatrix}$ we get

$$\left[\begin{array}{c|c} \lambda_1 \mathbf{I} - \mathbf{\Lambda}_{\text{diag}} & \mathbf{e}_1 \\ \hline \mathbf{e}_1^T & 0 \end{array}\right] \left[\begin{array}{c} \mathbf{X}_\lambda^T \mathbf{B} \frac{\partial \mathbf{x}_1}{\partial \omega_i} \\ \frac{\partial \lambda_1}{\partial \omega_j} \end{array}\right] = \left[\begin{array}{c} \mathbf{X}_\lambda^T \left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right) \mathbf{x}_1 \\ \hline -\frac{\mathbf{x}_1^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{x}_1}{2} \end{array}\right].$$

From this expression it follows that

$$\frac{\partial \lambda_1}{\partial \omega_j} = \mathbf{x}_1^T \left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right) \mathbf{x}_1$$

$$\mathbf{x}_k^T \mathbf{B} \frac{\partial \mathbf{x}_1}{\partial \omega_j} = \begin{cases} -\dfrac{\mathbf{x}_k^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{x}_1}{2} & , k = 1 \\ \dfrac{\mathbf{x}_k^T \left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right) \mathbf{x}_1}{\lambda_1 - \lambda_k} & , k \neq 1. \end{cases}$$

Finally, multiplying with $\mathbf{X}_\lambda$ from the left proves the statement. □

We observe that the weight of $\mathbf{x}_k, k = 2, \ldots, n$, in the expression for $\frac{\partial \mathbf{x}_1}{\partial \omega_j}$ depends on how close $\lambda_k$ is to $\lambda_1$ and depends on the projection of $\left(\frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_1 \frac{\partial \mathbf{B}}{\partial \omega_j}\right) \mathbf{x}_1, j = 1, \ldots, d$, on $\mathbf{x}_k$. Commonly, the closer $\lambda_k$ is to $\lambda_1$ the higher the impact is of $\mathbf{x}_k$. Note that the impact depends also on the nominator (in the second part of (3.8)), so it is possible that the impact is small despite the small gap between $\lambda_k$ and $\lambda_1$. We also deduce that if $\mathbf{B}$ does not depend on the parameters then the partial derivative is $\mathbf{B}$-orthonormal to $\mathbf{x}_1$. Example 5 shows that the norm of the partial derivative of the eigenvector cannot be bounded uniformly on the set $\Omega$, despite $\mathbf{A}$ and $\mathbf{B}$ being analytic and $\mathbf{B}$ positive definite for $\omega \in \Omega$.

**Example 5.** *We reconsider Example 3 in Section 1.3.2 with $n = 2$ and $\mathbf{W} = \mathbf{I}_2$. The eigenvalues and eigenvectors are easy to calculate in this case, but this example is chosen to illustrate that derivatives of eigenvectors are not necessarily*

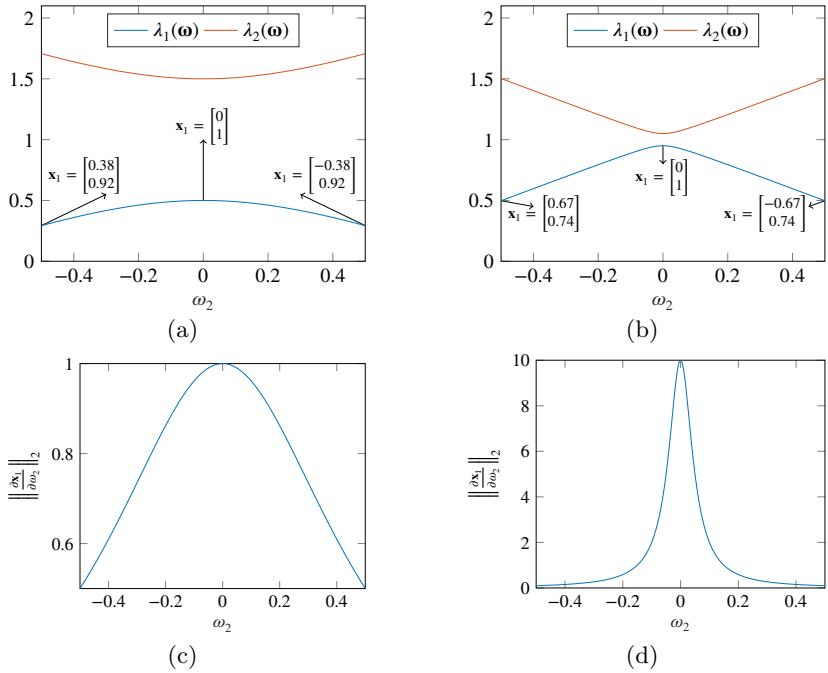



Figure 3.1: Figure corresponding to Example 5. (a) and (b): $\lambda_1(\omega)$ and $\lambda_2(\omega)$ with $\omega_1 = 0.5$ (left) and $\omega_1 = 0.05$ (right), (c) and (d): norm of $\frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_2}$ with $\omega_1 = 0.5$ (left) and $\omega_1 = 0.05$ (right)

*bounded. In Figure 3.1, we plot the eigenvalues* $\lambda_1(\omega)$ *and* $\lambda_2(\omega)$ *in (a) and (b) and the norm of* $\frac{\partial \mathbf{x}_1}{\partial \omega_2}$ *in (c) and (d) as a function of* $\omega_2$*, for fixed values of* $\omega_1$*.*

*One can observe that for* $\omega_2 = 0$*, the norm of the partial derivative* $\mathbf{x}_1$ *to* $\omega_2$ *becomes larger if* $\omega_1$ *goes to 0. It can be shown by Theorem 2 that this norm is unbounded. For* $\omega_2 = 0$*, the eigenvectors are equal to*

$$\mathbf{x}_1(\omega_1, 0) = \begin{bmatrix}0\\1\end{bmatrix} \text{ and } \mathbf{x}_2(\omega_1, 0) = \begin{bmatrix}1\\0\end{bmatrix}$$

*from which we can deduce that*

$$\begin{aligned}\frac{\partial \mathbf{x}_1}{\partial \omega_2}(\omega) &= \mathbf{x}_2(\omega_1,0)\frac{\left\langle \frac{\partial \mathbf{A}(\omega_1,0)}{\partial \omega_2}\mathbf{x}_1(\omega_1,0), \mathbf{x}_2(\omega_1,0)\right\rangle}{\lambda_1(\omega_1,0)-\lambda_2(\omega_1,0)} \\ &= \mathbf{x}_2(\omega_1,0)\frac{\left\langle \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right\rangle}{\lambda_1(\omega_1,0)-\lambda_2(\omega_1,0)} \\ &= \mathbf{x}_2(\omega_1,0)\frac{1}{-2\omega_1}.\end{aligned}$$

*We conclude that if $\omega_1$ goes to 0, the norm of the partial derivative goes to infinity.*

## 3.3 Algorithm

### 3.3.1 Idea

In the previous section we gave some insight how eigenvalues and eigenvectors depend on their parameters. In this section we deduce from these results, ideas how we can efficiently build a subspace. This subspace needs to be rich enough to estimate the minimum eigenvalue but needs to contain as less redundant information as possible. Our subspace contains on the one hand eigenvector samples of the first eigenvector(s) and on the other hand partial derivatives of the eigenvectors. At first sight, it may look strange why adding derivatives is profitable. The next result on Hermite interpolation gives the necessary theoretical justification. We also prove a result, stating that if we make the subspace larger then we only getting better results. This does not mean that it is beneficial to add vectors that does not improve our approximation. The larger the number of vectors that is added, the more costly to solve the projected eigenvalue problems.

**Property 6.** *1. (Hermite interpolation) If $(\lambda_i, \mathbf{x}_i)$ is a simple eigenpair and if $\mathbf{x}_i \in \mathcal{V}$, then*

- $\left(\lambda_i^{\mathcal{V}}, \mathbf{x}_i^{\mathcal{V}}\right) = \left(\lambda_i, \mathbf{V}^T\mathbf{x}_i\right)$ *is an eigenpair of* $\left(\mathbf{V}^T\mathbf{A}\mathbf{V}, \mathbf{V}^T\mathbf{B}\mathbf{V}\right)$.
- $\frac{\partial \lambda_i^{\mathcal{V}}}{\partial \omega_j} = \frac{\partial \lambda_i}{\partial \omega_j}, \quad j = 1, \dots d.$

*If also* $\frac{\partial \mathbf{x}_i}{\partial \omega_j} \in \mathcal{V}$, *then*

$$\frac{\partial^2 \lambda_i^{\mathcal{V}}}{\partial \omega_j \, \partial \omega_k} = \frac{\partial^2 \lambda_i}{\partial \omega_j \, \partial \omega_k}, \quad k = 1, \ldots, d.$$

2. *If* $\mathcal{V}_1 \subset \mathcal{V}_2$ *then*

$$\lambda_1 \leq \lambda_1^{\mathcal{V}_2} \leq \lambda_1^{\mathcal{V}_1}, \quad \forall \omega \in \Omega, \quad j = 1, \ldots, d.$$

*so by adding vectors we can only improve our estimation.*

*Proof.* Assertion 1:
The first two statements are well-known results in the case of a standard eigenvalue problem, see e.g. [35] and the references therein. The proofs for the generalized eigenvalue problem are stated for the sake of completeness. For the first statement, it is sufficient to see that $\mathbf{x}_i = \mathbf{V}\mathbf{V}^T\mathbf{x}_i$ to obtain $\mathbf{V}^T\mathbf{A}\mathbf{V}\mathbf{V}^T\mathbf{x}_i = \lambda_i \mathbf{V}^T\mathbf{B}\mathbf{V}\mathbf{V}^T\mathbf{x}_i$.
For the second, the result follows from

$$\begin{aligned}
\frac{\partial \lambda_i}{\partial \omega_j} &= \mathbf{x}_i^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{x}_i \\
&= \left( \mathbf{V}\mathbf{x}_i^{\mathcal{V}} \right)^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{V}\mathbf{x}_i^{\mathcal{V}} \\
&= \left( \mathbf{x}_i^{\mathcal{V}} \right)^T \left( \mathbf{V}^T \frac{\partial \mathbf{A}}{\partial \omega_j} \mathbf{V} - \lambda_i^{\mathcal{V}} \mathbf{V}^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{V} \right) \mathbf{x}_i^{\mathcal{V}} \\
&= \frac{\partial \lambda_i^{\mathcal{V}}}{\partial \omega_j}.
\end{aligned}$$

If also $\frac{\partial \mathbf{x}_i}{\partial \omega_j} \in \mathcal{V}$, then there exists a vector $\mathbf{z}$ such that $\frac{\partial \mathbf{x}_i}{\partial \omega_j} = \mathbf{V}\mathbf{z}$. We prove that $\mathbf{z} = \frac{\partial \mathbf{x}_i^{\mathcal{V}}}{\partial \omega_j}$, i.e. the projection of $\frac{\partial \mathbf{x}_i}{\partial \omega_j}$ on $\mathcal{V}$ equals the derivative of the first eigenvector of the reduced eigenvalue problem. This derivative is uniquely determined by the system in (3.5), from which it follows that

$$\begin{cases}
(\lambda_i \mathbf{B} - \mathbf{A})\, \mathbf{V}\mathbf{z} + \frac{\partial \lambda_i^{\mathcal{V}}}{\partial \omega_j} \mathbf{B}\mathbf{V}\mathbf{x}_i^{\mathcal{V}} &= \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i^{\mathcal{V}} \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{V}\mathbf{x}_i^{\mathcal{V}} \\
\left( \mathbf{x}_i^{\mathcal{V}} \right)^T \mathbf{V}^T \mathbf{B}\mathbf{V}\mathbf{z} &= -\frac{\left( \mathbf{x}_i^{\mathcal{V}}, \mathbf{V}^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{V}\mathbf{x}_i^{\mathcal{V}} \right)}{2}
\end{cases}$$

which means that $[\mathbf{z}, \frac{\partial \lambda_i^{\nu'}}{\partial \omega_j}]^T$ is the vector such that

$$\begin{cases} \mathbf{V}^T \left(\lambda_i \mathbf{B} - \mathbf{A}\right) \mathbf{V}\mathbf{z} + \frac{\partial \lambda_i^{\nu'}}{\partial \omega_j} \mathbf{V}^T \mathbf{B} \mathbf{V} \mathbf{x}_i^{\nu'} &= \mathbf{V}^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i^{\nu'} \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{V} \mathbf{x}_i^{\nu'} \\ \left(\mathbf{x}_i^{\nu'}\right)^T \mathbf{V}^T \mathbf{B} \mathbf{V} \mathbf{z} &= -\frac{\left( \mathbf{x}_i^{\nu'}, \mathbf{V}^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{V} \mathbf{x}_i^{\nu'} \right)}{2} \end{cases} .$$

This is the system that uniquely determines the derivative of the first eigenvector of the reduced eigenvalue problem, so $\mathbf{z} = \frac{\partial \mathbf{x}_i^{\nu'}}{\partial \omega_j}$.

From (3.4), we obtain

$$\begin{aligned} \frac{\partial^2 \lambda_i}{\partial \omega_j \, \partial \omega_k} &= 2 \left(\mathbf{x}_i^{\nu'}\right)^T \mathbf{V}^T \left( \frac{\partial \mathbf{A}}{\partial \omega_j} - \lambda_i \frac{\partial \mathbf{B}}{\partial \omega_j} \right) \mathbf{V} \frac{\partial \mathbf{x}_i}{\partial \omega_k}^{\nu'} + \dots \\ &\quad \left(\mathbf{x}_i^{\nu'}\right)^T \mathbf{V}^T \left( \frac{\partial^2 \mathbf{A}}{\partial \omega_j \, \partial \omega_k} - \frac{\partial \lambda_i}{\partial \omega_k} \frac{\partial \mathbf{B}}{\partial \omega_j} - \lambda_i \frac{\partial^2 \mathbf{B}}{\partial \omega_j \, \partial \omega_k} \right) \mathbf{V} \mathbf{x}_i^{\nu'}. \\ &= 2 \left(\mathbf{x}_i^{\nu'}\right)^T \left( \mathbf{V}^T \frac{\partial \mathbf{A}}{\partial \omega_j} \mathbf{V} - \lambda_i^{\nu'} \mathbf{V}^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{V} \right) \frac{\partial \mathbf{x}_i^{\nu'}}{\partial \omega_k} + \dots \\ &\quad \left(\mathbf{x}_i^{\nu'}\right)^T \left( \mathbf{V}^T \frac{\partial^2 \mathbf{A}}{\partial \omega_j \, \partial \omega_k} \mathbf{V} - \frac{\partial \lambda_i^{\nu'}}{\partial \omega_k} \mathbf{V}^T \frac{\partial \mathbf{B}}{\partial \omega_j} \mathbf{V} - \lambda_i^{\nu'} \mathbf{V}^T \frac{\partial^2 \mathbf{B}}{\partial \omega_j \, \partial \omega_k} \mathbf{V} \right) \mathbf{x}_i^{\nu'} \\ &= \frac{\partial^2 \lambda_i^{\nu'}}{\partial \omega_j \, \partial \omega_k}. \end{aligned}$$

Assertion 2: This results is well known, we refer to [52] for a proof. □

Property 6 presents the key theory for the construction of the reduced problem: adding eigenvectors and partial derivatives leads to Hermite interpolation of degree two for the eigenvalue. The eigenvalue and associated eigenvector are computed by a subspace projection method. The computation of the derivatives of the eigenvector from (3.5) is usually computationally expensive. Therefore we look for cheaper alternatives. Equation (3.8) suggests that the derivatives mainly depend on the eigenvectors associated with eigenvalues closest to $\lambda_1$. In particular, this is the case when the spectrum is well-separated with only a few eigenvalues near $\lambda_1$.

This implies that adding a few eigenvectors to the subspace in an interpolation point may serve as a computationally cheap surrogate for adding one eigenvector and its derivatives. Approximations to the associated eigenvectors of the eigenvalues nearest $\lambda_1$ are usually available from projection methods. Indeed,

when we use the Lanczos' method, the subspace produces reasonably good approximations to these eigenvectors, at no additional cost. Moreover, it should be noted that there is another reason why adding the second and third eigenvector may be beneficial: the eigenvector of the second minimum eigenvalue may be a good approximation to the eigenvector associated with the minimum eigenvalue at another parameter value $\omega_1 \in \Omega$. This means we probably do not need to solve a costly eigenvalue problem in $\omega_1$. Example 6 gives a good illustration of such a situation. In addition, we see in the next section that adding the second eigenvector helps making a sharp upper bound for the error.

**Example 6.** *We reconsider Example 3 in Section 1.3.2 where we take $n$ arbitrarily large. If $\omega_2 = 0$, then one can verify that*

$$\mathbf{x}_1(\omega_1, 0) = \begin{cases} \mathbf{w}_1(\omega_1, 0) & , \omega_1 \leq 0 \\ \mathbf{w}_2(\omega_1, 0) & , \omega_1 > 0 \end{cases}, \quad \mathbf{x}_2(\omega_1, 0) = \begin{cases} \mathbf{w}_2(\omega_1, 0) & , \omega_1 \leq 0 \\ \mathbf{w}_1(\omega_1, 0) & , \omega_1 > 0 \end{cases}.$$

*Take $\omega^1 < 0$ and $\omega^2 > 0$ two points in the first dimension of $\Omega$. Assume that the vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ do not heavily depend on $\omega$, then we obtain that $\mathbf{x}_1(\omega^1, 0) \approx \mathbf{x}_2(\omega^2, 0)$. Therefore we conclude that $\mathcal{V}_1 := \text{span}\{\mathbf{x}_1(\omega^i, 0) | i = 1, 2\} \approx \text{span}\{\mathbf{x}_i(\omega^1, 0) | i = 1, 2\} =: \mathcal{V}_2$, but $\mathcal{V}_2$ is less costly to calculate than $\mathcal{V}_1$. This is illustrated in Figure 3.2.*

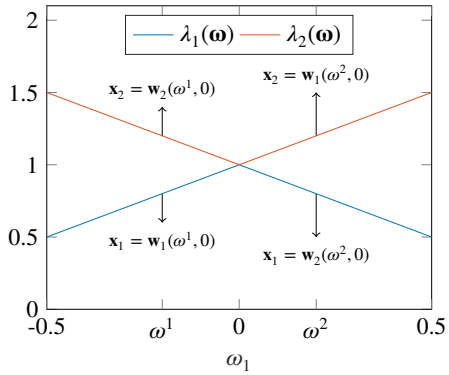



Figure 3.2: Figure corresponding to Example 6. If both $\mathbf{w}_1$ and $\mathbf{w}_2$ do not depend much on $\omega_1$, then $\mathcal{V}_1 := \text{span}\{\mathbf{x}_1(\omega^i), i = 1, 2\} \approx \text{span}\{\mathbf{x}_i(\omega^1), i = 1, 2\} =: \mathcal{V}_2$ but $\mathcal{V}_2$ is less costly to calculate than $\mathcal{V}_1$.

### 3.3.2 Upper bound of the error

Before we can state the algorithm, we need a measure to qualify if the subspace is rich enough or not. The subspace is rich enough if for every point in the

parameter space the difference between the minimum eigenvalue of the reduced problem and of the original large problem is smaller than a given tolerance. We assume that if this holds for every point in a set of parametervalues, it also holds for the whole parameter space. This means we need a way to measure the error for any given $\omega \in \Omega$. An upper bound can be obtained via an extension of the Bauer-Fike[1] theorem (see [4] and more recently [61]) to generalized eigenvalue problems. The proof of this extension is omitted as it is completely analogous to the original proof for the standard case. Note that in contrast to the previous section, in this section all matrices and all eigenpairs do not depend on parameters.

**Theorem 3.** *(Bauer-Fike for generalized eigenvalue problems) Let* $\left(\tilde{\lambda}, \tilde{\mathbf{x}}\right)$ *be an approximate eigenpair of* $(\mathbf{A}, \mathbf{B})$ *where* $\tilde{\mathbf{x}}$ *is of* $\mathbf{B}$*-norm unity and* $\tilde{\lambda} = (\mathbf{A}\tilde{\mathbf{x}}, \tilde{\mathbf{x}})$*. There is an eigenvalue* $\lambda$ *of* $(\mathbf{A}, \mathbf{B})$ *such that*

$$| \tilde{\lambda} - \lambda | \leq \frac{\|\mathbf{r}\|_2}{\sqrt{\lambda_1(\mathbf{B})}}. \tag{3.9}$$

*with* $\lambda_1(\mathbf{B})$ *the minimal eigenvalue of* $\mathbf{B}$*.*

The residual norm is very cheap to calculate for a set of parameter samples if the dependency of $\mathbf{A}$ and $\mathbf{B}$ on the parameters is affine. More precisely, if $\mathbf{A}(\omega)$ and $\mathbf{B}(\omega)$ can be written as $\mathbf{A}(\omega) = \sum_{i=1}^{m_A} f_{A,i}(\omega)\mathbf{A}_i$ resp. $\mathbf{B}(\omega) = \sum_{i=1}^{m_B} f_{B,i}(\omega)\mathbf{B}_i$ with $f_{A,i}, f_{B,j}, i = 1, \ldots m_A, j = 1, \ldots, m_B$, scalar functions then the residual is in this case

$$\mathbf{r}(\omega) = \mathbf{A}(\omega)\tilde{\mathbf{x}}(\omega) - \tilde{\lambda}(\omega)\mathbf{B}(\omega)\tilde{\mathbf{x}}(\omega) \tag{3.10}$$

$$= \sum_{i=1}^{m_A} f_{A,i}(\omega) \left(\mathbf{A}_i\mathbf{V}\right) \mathbf{x}^{\nu}(\omega) - \tilde{\lambda}(\omega) \sum_{i=1}^{m_B} f_{B,i}(\omega) \left(\mathbf{B}_i\mathbf{V}\right) \mathbf{x}^{\nu}(\omega). \tag{3.11}$$

This means that the precomputation of $\mathbf{A}_i\mathbf{V}, i = 1, \ldots, m_0$, and $\mathbf{B}_i\mathbf{V}, i = 1, \ldots, m_1$ reduces the cost of calculating the residual norm. The drawback of the result in Theorem 3 is that this is usually a too crude bound. If we have also information about the second minimum eigenvalue, a sharper bound can be obtained via an extension of the Kato-Temple[2] theorem [36], [69] for generalised eigenvalue problems.

**Theorem 4.** *(Kato-Temple for generalized eigenvalue problems) Let* $\left(\tilde{\lambda}, \tilde{\mathbf{x}}\right)$ *be an approximate eigenpair of* $(\mathbf{A}, \mathbf{B})$ *where* $\tilde{\mathbf{x}}$ *is of* $\mathbf{B}$*-norm unity and* $\tilde{\lambda} = (\mathbf{A}\tilde{\mathbf{x}}, \tilde{\mathbf{x}})$*.*

[1] Friedrich Ludwig "Fritz" Bauer (1924–2015) [83] was a German mathematics and computer scientist professor.

[2] Tosio Kato (1917–1999) [97] was a Japanese mathematics professor. His research interests were partial differential equations, mathematical physics and functional analysis.

*Assume that we know an interval* $]\alpha, \beta[$ *that contains* $\tilde{\lambda}$ *and one eigenvalue* $\lambda$ *of* $(\mathbf{A}, \mathbf{B})$. *Then it holds*

$$-\frac{\|\mathbf{r}\|_2^2}{\lambda_1(\mathbf{B})\left(\tilde{\lambda}-\alpha\right)} \leq \tilde{\lambda}-\lambda \leq \frac{\|\mathbf{r}\|_2^2}{\lambda_1(\mathbf{B})\left(\beta-\tilde{\lambda}\right)}$$

*with* $\lambda_1(\mathbf{B})$ *the minimal eigenvalue of* $\mathbf{B}$.

If we now define $\delta$ as the distance to the second minimum eigenvalue, then we get

$$|\tilde{\lambda}-\lambda_1| \leq \frac{\|\mathbf{r}\|_2^2}{\lambda_1(\mathbf{B})\delta}, \tag{3.12}$$

which is in general a sharper upper bound then the one from Theorem 3.

In both theorems the bound depends on the first eigenvalue of $\mathbf{B}$. As it is expensive to calculate this eigenvalue for every $\omega$, we only calculate it for one parameter value and use it for the whole domain. As already mentioned, we approximate the second, respectively third eigenvalue by the second, respectively third Ritz value from the Krylov subspace. Whether we use the Bauer-Fike theorem or the Kato-Temple theorem depends on how accurately we can determine the gap $\delta$ between the first two eigenvalues. If we add the first $m$ eigenvectors ($m > 1$) corresponding to every sample point to the subspace, we interpolate the first $m$ eigenvalues in the sample points, and obtain in this way a global approximation of the first $m$ eigenvalues over the parameter space. Hence we expect that the gap $\delta$ can be estimated well from the projected eigenvalue problem, and we make in this case use of the Kato-Temple Theorem, where we replace $\delta$ by its estimate. If only one eigenvector is added in a sample point, an accurate estimate of the gap from the projected eigenvalue problem cannot be guaranteed. In this case the Bauer-Fike theorem is invoked instead.

### 3.3.3 Algorithm

We now have all ingredients for an algorithm to compute the reduced problem. A remaining crucial element is the selection of sample points, which we now discuss. The proposed method is inspired by the reduced basis method for solving parametric PDEs, see [56].

The goal is to make a subspace $\mathcal{V}$ such that for a set of parameter values called the training set $\Omega_{\text{train}}$, the first eigenvalue of the projected eigenvalue is within some tolerance of the first eigenvalue of the large eigenvalue problem. For this, we first make an initial subspace $\mathcal{V}$ constructed with the first eigenvectors ($\mathbf{x}_1$ and eventually $\mathbf{x}_2$) and partial derivatives of $\mathbf{x}_1$ for all parameter values in an

initial set of points $\Omega_{\text{init}}$ (see line 1-4 in Algorithm 4). More info about the construction of these sets can be found further in this subsection. The set $\Omega_{\text{init}}$ is called the initial set.

The idea is that as long as the upper bound of the error for one of the parameter values in $\Omega_{\text{train}}$ is too high, we add the first eigenvectors and partial derivatives of the parameter value in the training set $\Omega_{\text{train}}$ where this upper bound is maximal (see line 16-26 in Algorithm 4). Whether we use the Kato-Temple Theorem (Theorem 4) or the Bauer-Fike Theorem (Theorem 3) for constructing an upper bound, depends on the number of eigenvectors we add per interpolation point, see previous section. If for some parameter value, our calculated upper bound is below the tolerance, this parameter value is removed from the training set $\Omega_{\text{train}}$ (see line 7-12 in Algorithm 4). We are allowed to do this because this implies that the error on the eigenvalue is below the tolerance and furthermore the error cannot increase by the second assertion in Theorem 6.

We explain in this paragraph how we constructed the sets $\Omega_{\text{init}}$ and $\Omega_{\text{train}}$. We consider here the case when $\Omega$ is a product of intervals:

$$\Omega = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d].$$

An initial set $\Omega_{\text{init}}$ and $\Omega_{\text{train}}$ can then be computed as described in (2.5) in Section 2.2. For the training set a much finer grid is used than for the initial set. The optimal number of initial points is difficult to say since it depends mainly on how many times the first eigenvalue switches in the parameter space. The initial subspace must contain for every parameter value at least a part of its first eigenvector, therefore it is better to make this set too large than too small.

The algorithm is stated in pseudocode in Algorithm 4. As we do experiments with adding partial derivatives to the subspace and with not adding them to the subspace, we state 'optionally' after the lines where we calculate the partial derivatives.

### 3.3.4 Saturation assumption

Although the training set becomes smaller in every iteration, calculating the upper bound for all sample points in the training set can become expensive if many iterations are needed. Instead of updating the upper bounds for all parameter values, we update only those for which its upper bound is potentially the largest. For this we make use of the so-called saturation assumption, see [29]. Let $u_k(\omega)$ be the upper bound at iteration $k$ for parameter value $\omega$. The saturation assumption says that there exists a $C > 0$ such that

$$u_l(\omega) < C u_k(\omega), \forall l > k, \forall \omega \in \Omega. \tag{3.13}$$

The saturation assumption tells us that the upper bound in iteration $k$ is smaller that the upper bound in the next iterations upto a factor $C$. In our method the saturation assumption is fulfilled for $C = 1$ via the second assertion in Property 6.

We first sort the parameter values in $\Omega_{\text{train}}$ by the upper bound of the previous iteration in descending order. We recompute the upper bound and update after each computation the maximum upper bound $u_{\max}$. If for a certain $\omega$, it holds that $u_k(\omega) < u_{\max}$ we are allowed to skip all the next sample points in the sequence by (3.13). The next $\omega$ for which we add the eigenvectors and all partial derivatives, is the $\omega$ which has the upper bound $u_{\max}$.

## 3.4 Numerical results

In all examples, the linear systems of the form (3.5) to compute the derivative of an eigenvector could still be solved efficiently using a direct method. We note that there are other methods for calculating derivatives of eigenvectors, see [47, 74]. We use the `eigs`-command in Matlab to calculate the minimum eigenvalue which uses the shift-and-invert Arnoldi's method with shift 0. This method makes a Krylov subspace and we adapt the code such that it also returns this space. The linear systems solved in this Krylov algorithm are also computed using a direct method. We use the other vectors in the Krylov subspace to approximate the second eigenvector. In all examples a tolerance for the error of $10^{-5}$ is used.

We used two examples from the paper [65]. This is the state-of-the-art for solving parametrized eigenvalue problems (see Section 1.3.2) and therefore this paper contains representative examples. In both examples the minimum eigenvalue corresponds to the coercivity constant. As these two examples have a constant $\mathbf{B}$, we apply the method also on an example where the matrix $\mathbf{B}$ is dependent on the parameters. We elaborate more on this example later in this section. We compare the results from four cases: adding only the first eigenvector, adding the first two eigenvectors, adding the first eigenvector and all partial derivatives and finally the first two eigenvectors and all partial derivatives. The reason we do not include experiments with adding more than two eigenvectors is that it did not add any valuable information to the subspace. As said before, the main reason we add the second eigenvector is that we can prove a sharper bound for the error. As far as we know, there is no result we can use the third or fourth eigenvector for obtaining a sharper upper bound. In the next tables, we put the dimension of the found subspace, the number of sample points, the

total computing time and the average time we needed to compute the partial derivatives and eigenvectors.

For Example 2 in Section 1.3.2, the initial set is built by discretising both dimensions of the parameter space into four to get an initial subset of 16 points. For the training set, we discretise the first respectively the second dimension of the parameter space into 25 respectively 40 points, to have a total of 1000 points in our training set. The results are stated in Table 3.1. We see that adding other information than only the first eigenvector is beneficial. The number of points where we need to calculate the second eigenvector and the partial derivative is much lower which results in a smaller computational time. The main benefit from adding the second eigenvector is that we can use the more strict upper bound of Kato-Temple. We also noticed that the time needed to compute the derivatives is much lower than calculating the eigenvalues. We see we get the best results when we add both the eigenvectors and the partial derivatives into the subspace.

Table 3.1: The results for Example 2

| | 1 eigv | 2 eigv | 1 eigv + partial deriv. | 2 eigv + partial deriv. |
|---|---|---|---|---|
| dimension $\mathcal{V}$ | 96 | 108 | 138 | 132 |
| nbr points | 96 | 54 | 46 | 33 |
| total time | 59.04s | 38.92s | 50s | 29.293s |
| time derivative (per vector) | / | / | 0.002s | 0.002s |
| time eigenv (per vector) | 0.08s | 0.04s | 0.079s | 0.044s |

The next example is taken from [65, Example 4.3]. The parameter space is $\Omega = [-0.1, 0.1] \times [0.2, 0.3]$ and the matrix $\mathbf{A}$ is of the form $\mathbf{A}(\omega) = \sum_{i=1}^{16} \theta_i(\omega)\mathbf{A}_i$ with $\theta_i(\omega)$ analytic functions and matrix $\mathbf{B}$ is constant. The dimension of the problem is 2183. As training set we choose a grid where we discretise the interval for $\omega_1$ into 40 and $\omega_2$ into 25 points. As initial set we discretise both intervals into three. The results are stated in Table 3.2. We see that in this case the needed subspace is low-dimensional which results in a low computational time. Similar conclusions can be drawn for this example.

In the previous examples matrix $\mathbf{B}$ was constant, since the article from which we took the examples considered only eigenvalue problems with constant $\mathbf{B}$.

The next example originates from an application in structural mechanics and here matrix $\mathbf{B}$ depends on $\omega$. The matrices in this example are the stiffness and mass matrices arising from a finite element approximation of a vibrating beam that can deflect in both directions perpendicular to its own axis. The beam

Table 3.2: Results

| | 1 eigv | 2 eigv | 1 eigv + partial deriv. | 2 eigv + partial deriv. |
|---|---|---|---|---|
| dimension $\mathcal{V}$ | 37 | 30 | 48 | 40 |
| nbr points | 37 | 15 | 16 | 10 |
| total time | 32.04s | 14.62s | 24.372s | 8.499s |
| time derivative (per vector) | / | / | 0.005s | 0.004s |
| time eigenv (per vector) | 0.03s | 0.015s | 0.026s | 0.012s |

length is 1 and its size along the z-axis is 0.5. The beam size along the y-axis varies between 0.1 and 1 and is in Figure 3.3 denoted with $\omega_1$. We take meters as unit of length. The second parameter $\omega_2$ is the elastic modulus and lays in the interval $[100, 1000]$. The elastic modulus is expressed in $N/m^2$. We divided the beam in eight equal parts along the x-axis where part $i$ has elastic modulus $\omega_2 + 10(i-1), i = 1, \ldots, 8$ and density $70 + 10(i-7)$. For the initial set, we discretise both dimensions into three. The matrices have size 1404 and they both depend on at least one of the parameters. In this problem, the stiffness and mass matrix both depend on the two parameters. One is usually interested in the minimum eigenfrequencies which boils down to calculating the minimum eigenvalue. In Figure 3.3 we depict the minimum eigenvalue over the parameter space and Table 3.3 summarizes the results. We observe that only a small subspace was needed and that we benefit from including partial derivatives in the subspace since the computation of the eigenvectors takes much more time than solving the system to compute the partial derivatives.

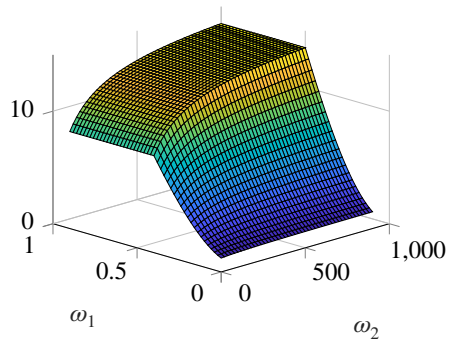



Figure 3.3: The minimum eigenvalue $\lambda_1(\omega)$ over the parameter for the example from structural mechanics. We see the minimum eigenvalue does not smoothly depend on the variable.

Table 3.3: Results for example from structural mechanics

| | 1 eigv | 2 eigv | 1 eigv + partial deriv. | 2 eigv + partial deriv. |
|---|---|---|---|---|
| dimension $\mathcal{V}$ | 17 | 22 | 32 | 31 |
| nbr points | 17 | 11 | 12 | 9 |
| total time | 193.73s | 105.81s | 123.13s | 83.53s |
| time derivative per vector | / | / | 0.005s | 0.005s |
| time eigenv per vector | 1.743s | 0.871s | 1.787s | 0.890s |

## 3.5 Expected value of the minimum eigenvalue

So far in this chapter we wanted to approximate the minimum eigenvalue of a parametrized matrix-pair. By modifying a bit our algorithm we can apply it to calculate the mean of the minimum eigenvalue of a matrix-pair where the parameter is a stochastic variable. The key idea is that we tolerate an error dependent on the distribution function. We restrict ourselves here to the case that the distribution function is known and can be evaluated.

We prove how we can set bounds on the tolerance dependent on the solution of the random variable. We approximate the expected value by a Quasi Monte-Carlo (QMC) simulation where we only calculate approximations $\lambda_1^{\mathcal{V}}$. To bound the error, we split it into 2 parts by

$$\begin{aligned}
&\left|\mathbb{E}[\lambda_1] - \frac{1}{n}\sum_{i=1}^{n}\lambda_1^{\mathcal{V}}(\omega_i)\right| \\
&= \left|\mathbb{E}[\lambda_1] - \mathbb{E}[\lambda_1^{\mathcal{V}}] + \mathbb{E}[\lambda_1^{\mathcal{V}}] - \frac{1}{n}\sum_{i=1}^{n}\lambda_1^{\mathcal{V}}(\omega_i)\right| \\
&\leq \left|\mathbb{E}[\lambda_1] - \mathbb{E}[\lambda_1^{\mathcal{V}}]\right| + \left|\mathbb{E}[\lambda_1^{\mathcal{V}}] - \frac{1}{n}\sum_{i=1}^{n}\lambda_1^{\mathcal{V}}(\omega_i)\right|.
\end{aligned}$$

The first part is the expected error that we make on the eigenvalue and the second term tells something about (Q)MC-method we are applying. This last part can be made arbitrary small by increasing the number of samples. We want to bound the first part by a certain tolerance $\epsilon > 0$. For this, we put a restriction on the accuracy of our approximation. We assume that our parameter space is a Cartesian product of intervals in $\mathbb{R}$, but distributions (like the normal distribution) are usually defined on unbounded intervals. In that case we limit

the size of the interval such that the probability of a sample to be outside this interval, is negligible. For example, in the case of the normal distribution with mean $\mu$ and standard deviation $\sigma$, we cut of the interval in $[\mu - 4\sigma, \mu + 4\sigma]$. We denote the surface area of $\Omega$ as $\text{surf}(\Omega)$.

**Property 7.** *If* $\left|\lambda_1(\omega) - \lambda_1^{\mathcal{V}}(\omega)\right| \leq \dfrac{\epsilon}{\text{surf}(\Omega)p(\omega)}, \forall \omega \in \Omega$, *then*

$$\mathbb{E}[\lambda_1^{\mathcal{V}} - \lambda_1] \leq \epsilon.$$

*Proof.*

$$\begin{aligned}
&\left|\mathbb{E}[\lambda_1^{\mathcal{V}} - \lambda_1]\right| \\
=&\int_\Omega \left(\lambda_1^{\mathcal{V}}(\omega) - \lambda_1(\omega)\right) p(\omega) d\omega \\
\leq&\int_\Omega \frac{\epsilon}{\text{surf}(\Omega)p(\omega)} p(\omega) d\omega \\
=&\frac{\epsilon}{\text{surf}(\Omega)} \int_\Omega d\omega \\
=&\epsilon.
\end{aligned}$$

□

This property has the meaning that the higher the probability in a sample point, the more strict our upper bound is. In comparison with Algorithm 4 we need to change two things: on line 11 the tolerance needs to be

$$\epsilon(\omega) = \frac{\epsilon}{\text{surf}(\Omega)p(\omega)}$$

and on line 17 we take the parameter value for which holds that the relative value $u(\omega)/\epsilon(\omega)$ is maximal.

### 3.5.1 Numerical experiment

We consider Example 2 again where our parameters are now stochastic variables. They both follow a multivariate normal distribution $(\mu, \mathbf{\Sigma})$ with

$$\mu = \begin{bmatrix} 0.26 \\ 5 \end{bmatrix}, \quad \mathbf{\Sigma} = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

The training set $\Omega_{\text{train}}$ consists of 1000 values, our initial set $\Omega_{\text{init}}$ consists of 16 elements and the tolerance $\epsilon$ is set on $10^{-4}$. From this, we make a subspace of dimension 84, which is approximately 6.4 percent of the total dimension. Evaluating $\lambda_1^{\nu}$ for a random parameter value takes 10 times less time than evaluating $\lambda_1$. This means that we need less time for evaluating the expected value using the projected matrix pair than with the full matrix pair and it compensates by far the fact that we do not evaluate the exact eigenvalue.

## 3.6 Conclusion

The main contribution of this chapter is that we showed that for making a global approximation of an extreme eigenvalue of a parametrized matrix pencil where both matrices are symmetric and at least one of the two matrices is positive definite, it is beneficial to add, besides the first eigenvector, also its partial derivatives or alternatively the Ritz vector associated with the second Ritz value. We have proved that in the case of the addition of partial derivatives, the eigenvalues satisfy a Hermite interpolation property of order two. Numerical examples confirm this statement and showed that this property can also be used in practice.

We extend our method such that we can also calculate the mean of the minimum eigenvalue if the parameters are stochastic.

**Algorithm 4** Algorithm for calculating a subspace $\mathcal{V}$ such that $|\lambda_1^{\mathcal{V}}(\omega) - \lambda_1(\omega)| < \epsilon, \forall \omega \in \Omega_{\text{train}}$.

**Input:**

1. Matrix $\mathbf{A}(\omega)$ and $\mathbf{B}(\omega)$
2. All partial derivatives $\frac{\partial \mathbf{A}(\omega)}{\partial \omega_i}$ and $\frac{\partial \mathbf{B}(\omega)}{\partial \omega_i}, i = 1, 2, \ldots, d$
3. Choose an initial set $\Omega_{\text{init}}$ and a training set $\Omega_{\text{train}}$ of sample points.
4. $n_{\max}$ maximum number of iterations

**Output:** Subspace $\mathcal{V}$ such that $|\lambda_1^{\mathcal{V}}(\omega) - \lambda_1(\omega)| < \epsilon, \forall \omega \in \Omega_{\text{train}}$.

Calculate $(\lambda_1(\omega), \mathbf{x}_1(\omega)), \omega \in \Omega_{\text{init}}$
Estimate $\left(\tilde{\lambda}_j(\omega), \tilde{\mathbf{x}}_j(\omega)\right), j = 2, 3, \ldots, m, \omega \in \Omega_{\text{init}}$
Calculate partial derivatives $\frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_j}, j = 1, 2, \ldots, d, \omega \in \Omega_{\text{init}}$ from system (3.5) if $|\lambda_1 - \tilde{\lambda}_2| > 10^{-8}$ (If $\lambda_1$ is simple) (Optionally)
$\mathcal{V} := \text{span}\{\mathbf{x}_1(\omega), \tilde{\mathbf{x}}_2(\omega) \ldots, \tilde{\mathbf{x}}_m(\omega), \frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_1}, \frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_2}, \ldots, \frac{\partial \mathbf{x}_1(\omega)}{\partial \omega_d} | \omega \in \Omega_{\text{init}}\}$
$u_{\text{old}}(\omega) = 1, \omega \in \Omega_{\text{train}}$
**for** $i = 1, 2, \ldots, n_{\max}$ **do**
    Update the training set
    Calculate $\left(\lambda_1^{\mathcal{V}}(\omega), \mathbf{x}_1^{\mathcal{V}}(\omega)\right)$ for all $\omega \in \Omega_{\text{train}}$ (except those excluded by the saturation assumption)
    Calculate new upper bound $u_{\text{new}}(\omega)$ for all $\omega \in \Omega_{\text{train}}$ using Theorem 3 if $m = 1$ or Theorem 4 if $m > 1$. (except those excluded by the saturation assumption)
    Upper bound $u(\omega) = \min(u_{\text{old}}(\omega), u_{\text{new}}(\omega))$
    $\Omega_{\text{valid}} = \{\omega | \omega \in \Omega_{\text{train}} \text{ and } u(\omega) < \epsilon\}$
    $\Omega_{\text{train}} = \Omega_{\text{train}} \setminus \Omega_{\text{valid}}$
    **if** $\Omega_{\text{train}}$ empty **then**
        break
    **else**
        Adding vectors to the subspace
        $\omega^i = \arg\max_{\omega \in \Omega_{\text{train}}} u(\omega)$
        Calculate $\left(\lambda_1(\omega^i), \mathbf{x}_1(\omega^i)\right)$
        Estimate $\left(\tilde{\lambda}_j(\omega^i), \tilde{\mathbf{x}}_j(\omega^i)\right), j = 2, 3, \ldots, m$
        **if** $|\lambda_1 - \tilde{\lambda}_2| > 10^{-8}$ **then** (Check if $\lambda_1$ is simple)
            Calculate partial derivatives $\frac{\partial \mathbf{x}_1(\omega^i)}{\partial \omega_j}, j = 1, 2, \ldots, d$ from system (3.5) (Optionally)
            $\mathcal{V} = \mathcal{V} \bigcup \{\mathbf{x}_1(\omega^i), \tilde{\mathbf{x}}_2(\omega^i) \ldots, \tilde{\mathbf{x}}_m(\omega^i)\}$
            $\left(\bigcup \{\frac{\partial \mathbf{x}_1(\omega^i)}{\partial \omega_1}, \frac{\partial \mathbf{x}_1(\omega^i)}{\partial \omega_2}, \ldots, \frac{\partial \mathbf{x}_1(\omega^i)}{\partial \omega_d}\} (\text{optionally})\right)$
        **else**
            $\mathcal{V} = \mathcal{V} \bigcup \{\mathbf{x}_1(\omega^i), \tilde{\mathbf{x}}_2(\omega^i) \ldots, \tilde{\mathbf{x}}_m(\omega^i)\}$
        **end if**
        Make basis for $\mathcal{V}$
        $\Omega_{\text{train}} = \Omega_{\text{train}} \setminus \omega^i$
    **end if**
**end for**

# Chapter 4

# Tensor-Krylov method for computing eigenvalues of parametrized matrices

*Insanity is doing the same thing*
*over and over again*
*and expecting different results*
*(Unknown)*

Parts of this chapter are copied from:

RUYMBEEK K., MEERBERGEN K., AND MICHIELS W.
"Tensor-Krylov method for computing eigenvalues of parameter-dependent matrices", submitted for publication to Journal of Computational and Applied mathematics.

## 4.1 Introduction

In the previous chapter, we presented a projection method to calculate a global approximation of the minimal eigenvalue of a parametrized pair $(\mathbf{A}, \mathbf{B})$ where both matrices are symmetric and $\mathbf{B}$ is positive definite. The goal in this chapter is similar, but is focused on a different class of matrix pencils. We want to make

a global approximation of an extreme eigenvalue following some ordering (e.g. sorted by modulus if $\lambda_1$ is a dominant eigenvalue) of a matrix $\mathbf{A}$ dependent on parameters. We consider both symmetric and non-symmetric matrices. In the last section, we adapt our method such that we can calculate eigenvalues closest to shift $\sigma$. For each parameter value, the eigenvalues are sorted such that the wanted eigenvalue is $\lambda_1$. We assume that $\mathbf{A}$ is of the form

$$\mathbf{A} : \Omega \to \mathbb{R}^{n\times n} : \omega \mapsto \sum_{i=1}^{m} f_i(\omega)\mathbf{A}_i \tag{4.1}$$

on a parameter set $\Omega \subset \mathbb{R}^d$ and functions $f_i, i = 1, 2, \ldots, m$, where $m$ is small relative to $n$. Furthermore, we assume that the wanted eigenvalue converges for any $\omega$ after a few iterations of a Krylov-type method meaning that it is a well-separated eigenvalue of $\mathbf{A}$.

Like in the previous chapter, we tackle this problem using a projection method but its construction is totally different. As in the previous chapter, we have a set of parameter values for which we want to calculate the wanted eigenvalue. We do not add vectors related to the eigenvalue for a specific parameter value to the subspace but instead we add information of all considered parameter values to the subspace.

This is done via a Tensor-Krylov method. In this type of methods, Krylov methods are combined with low-rank tensors. Tensor-Krylov techniques are already discussed for various problems, for example to solve a large system of non-linear equations [5, 6] and parametrized linear systems [38]. Here, we apply a Krylov method on $\mathbf{A}(\omega)$ for all $\omega$ in our set of parameter values at the same time. As we cannot add the iteration vector for each parameter value to the subspace in each iteration, we make a low-rank decomposition of the tensor containing the eigenvectors and we only add the column basis to the subspace. This introduces error in the algorithm. A practical issue is the growth of the dimension of this subspace as we add multiple vectors in each iteration. Therefore we often need to restart to keep the algorithm feasible. We gain in efficiency if we need to add fewer vectors to the subspace (and so introducing a larger error) and have a good way to restart the subspace. We observed that making a large, to some extent, error does not affect the convergence and can be even profitable. We explain this phenomenon and we give practical bounds for the introduced errors.

The plan of this chapter is as follows. In Section 4.2 we discuss the Arnoldi method and analyse how introduced errors are propagated in the algorithm and give convergence results. We explain the proposed algorithm in the one parameter case and discuss why it is an extension of the Residual Arnoldi algorithm in Section 4.3. Subsequently, we illustrate the convergence and error

propagation with some numerical experiments. In Section 4.5, we revisit the method for multiple parameters and report a numerical experiment. We extend our method to shift-and-invert Residual Arnoldi when all matrices in (4.1) except one is of rank one and give some numerical examples. We finish this chapter with some concluding remarks. The Matlab code can be downloaded from

`http://twr.cs.kuleuven.be/research/software/delay-control/TensorKrylov`.

## 4.2 Arnoldi method

In Section 2.1.2 we introduced Krylov spaces and motivated why they are used for the calculation of some extreme eigenvalues. In this section we discuss the *Arnoldi*[1] *method* and a variant on this, the Residual Arnoldi algorithm which both build up an orthonormal basis for a Krylov space. We explain how convergence is measured and give some results regarding its performance when the iterations are contaminated by error.

**Algorithm 5** Arnoldi algorithm

**Input:** Matrix $\mathbf{A}$, normalized vector $\mathbf{u} \in \mathbb{R}^n$
**Output:** Approximate eigenpair $(\hat{\lambda}_1, \hat{\mathbf{x}}_1)$ of $\mathbf{A}$

$\mathbf{w}^\perp = \mathbf{u}$
$\mathbf{t} = 1$
**for** $k = 1$ to $n - 1$ **do**
    Add $\mathbf{w}^\perp$ to the subspace $\mathcal{V}$
    $\mathbf{w} = \mathbf{AVt}$
    Orthogonalise $\mathbf{w}$ with respect to $\mathbf{V}$ and call this $\mathbf{w}^\perp$
    Calculate wanted eigenpair $\left(\lambda_1^{\mathcal{V}}, \mathbf{x}_1^{\mathcal{V}}\right)$ of $\mathbf{V}^T\mathbf{AV}$
    Break down if $\left(\lambda_1^{\mathcal{V}}, \mathbf{x}_1^{\mathcal{V}}\right)$ meets a certain convergence criterium
    Construct the continuation vector $\mathbf{t}$ (see further in text)
**end for**
$\hat{\lambda}_1 = \lambda_1^{\mathcal{V}}$ and Ritz vector $\hat{\mathbf{x}}_1 = \mathbf{V}\mathbf{x}_1^{\mathcal{V}}$

For our application, it is important that the algorithm is robust against introduced error. The reason is that we apply a version of the Arnoldi algorithm on all parameter values in our set at the same time. To prevent that our subspace becomes too large, we make a low-rank approximation of the tensor consisting of

[1] Walter Edwin Arnoldi (1917–1995) [99] was an American engineer.

all iteration vectors and so error is introduced in the algorithm. In the original Arnoldi algorithm [2], the used continuation vector is $\mathbf{t} = [0, \ldots, 0, 1]^T$, so that $\mathbf{Vt}$ is equal to the last column of $\mathbf{V}$ (so the last added vector to the subspace). It is showed in [40] that in case there is a significant error (numerical or/and intented) present in one of the vectors, it does not converge anymore. So we look for a continuation vector such that the Arnoldi algorithm converges even when the iteration vector is (to a limited extend) contaminated with error. In the papers [105, 106] the authors answered the question what the best choice is for the continuation vector. For this, they searched for the continuation vector that maximizes

$$\begin{aligned} \cos\left(\theta(\mathbf{x}_1, \text{Range}\{\mathbf{V}, \mathbf{AVt}\})\right) &:= \max_{\mathbf{z} \in \text{Range}\{\mathbf{V}, \mathbf{AVt}\}} \cos(\theta(\mathbf{x}_1, \mathbf{z})) \\ &= \max_{\mathbf{z} \in \text{Range}\{\mathbf{V}, \mathbf{AVt}\}} \frac{\mathbf{x}_1^* \mathbf{z}}{\|\mathbf{x}_1\| \, \|\mathbf{z}\|}, \end{aligned} \tag{4.2}$$

so they look which continuation vector is such that the angle between the sought eigenvector and the extended subspace is minimized.

In [106], they proved that the vector that maximizes (4.2) is of the form

$$\mathbf{t} = \mathbf{R}^+ \mathbf{x}_1 + \mathbf{b} \tag{4.3}$$

with $\mathbf{b}$ a vector in the nullspace of $\mathbf{R} = \mathbf{AV} - \mathbf{V}\left(\mathbf{V}^T\mathbf{AV}\right)$, $\mathbf{R}^+$ the Moore-Penrose[2] pseudo-inverse of $\mathbf{R}$.

Although this result gives us insight in the optimal continuation vector, it is impossible for practical use as it depends on the searched eigenvector. The author of [106] argued that in practice, the eigenvector

$$\mathbf{t} = \mathbf{x}_1^{\mathcal{V}}. \tag{4.4}$$

associated with the desired eigenvalue of the reduced problem $\mathbf{V}^T\mathbf{AV}$ is a good practical choice for the continuation vector. In the remainder of this chapter, this is used as continuation vector, despite the fact that in the case of large non-Hermitian matrices, it may be better to use the *refined Ritz vector* [105].

In the literature, the Arnoldi algorithm with (4.4) as continuation vector is known as the *Residual Arnoldi method* [40, 41] . It was already observed that this choice leads to a more robust version of the Arnoldi algorithm. This is

[2]Sir Roger Penrose (1931) [96] is a well-known English mathematical physicist, mathematician and philosopher of science. He has received several prizes and awards. He inspired the artist M.C. Esher for his world famous lithograph "Waterfall". Eliakim Hastings Moore (1862-1932) [79] was an American mathematician. The American Mathematical Society established a prize in his honor

further discussed in Section 4.2.1. The name originates from the property that the residual is added in each iteration, see Property 4. If the norm of the residual is used as convergence criterion, we can simplify the Residual Arnoldi Algorithm. The resulting algorithm is shown in Algorithm 6.

**Algorithm 6** Residual Arnoldi (RA) Algorithm

**Input:** Matrix $\mathbf{A}$, initial vector $\mathbf{u} \in \mathbb{R}^n$
**Output:** Approximate eigenpair $(\hat{\lambda}_1, \mathbf{x}_1)$ of $\mathbf{A}$

1: $\mathbf{r} = \mathbf{Au} - \lambda^{\mathbf{u}}\mathbf{u}$, with $\lambda^{\mathbf{u}}$ the Ritz value associated to the subspace .
2: **for** $k = 1$ to $n$ **do**
3: add $\mathbf{r}$ to the subspace $\mathcal{V}$
4: Calculate wanted eigenpair $\left(\lambda_1^{\mathcal{V}}, \mathbf{x}_1^{\mathcal{V}}\right)$ of $\mathbf{V}^T\mathbf{AV}$
5: $\hat{\mathbf{x}}_1 = \mathbf{V}\mathbf{x}^{\mathcal{V}}$
6: Calculate the residual $\mathbf{r}$ of $\left(\lambda_1^{\mathcal{V}}, \hat{\mathbf{x}}_1\right)$ via Property 4 in Section 2.1
7: Stop if $\|\mathbf{r}\|$ is small enough
8: **end for**
9: $\hat{\lambda}_1 = \lambda_1^{\mathcal{V}}$ and Ritz vector $\hat{\mathbf{x}}_1 = \mathbf{V}\mathbf{x}_1^{\mathcal{V}}$

### 4.2.1 Convergence of the Residual Arnoldi algorithm

We noted at the end of last section that the use of (4.4) is more robust against introduced error. We discuss the convergence and give tolerances for the different errors such that convergence stays assured. The possible error $\mathbf{e}$ is modelled as an error on the residual $\mathbf{r}$, i.e on the vector that we want to add to the subspace. The error $\mathbf{e}$ consists of an error $\mathbf{e}_1$ on $\mathbf{x}_1^{\mathcal{V}}$ (before the orthogonalization on line 5 in the RA algorithm) and an error $\mathbf{e}_2$ on $\mathbf{r}$ (after the orthogonalization on line 6). We assume that $\mathbf{e}_2$ is already orthogonal to $\mathcal{V}$, because otherwise the added vector is no longer orthogonal to $\mathcal{V}$. To resume, the vector that we add to the subspace can be written as

$$\begin{aligned}\hat{\mathbf{r}} &= \left(\mathbf{I} - \mathbf{V}\mathbf{V}^T\right)\mathbf{AV}\left(\mathbf{x}_1^{\mathcal{V}} + \mathbf{e}_1\right) + \mathbf{e}_2 \\ &= \mathbf{r} + \underbrace{\underbrace{\left(\mathbf{I} - \mathbf{V}\mathbf{V}^T\right)\mathbf{AV}\mathbf{e}_1}_{\mathbf{e}_1^+} + \mathbf{e}_2}_{\mathbf{e}} .\end{aligned} \tag{4.5}$$

The convergence of this method was analysed in [40, 41]. The authors assume that in each iteration $k$ the error $\mathbf{e}^{(k)}$ relative to the residual, is bounded by

some $\eta$, in the sense

$$\frac{\left\|\mathbf{e}^{(k)}\right\|}{\left\|\mathbf{r}^{(k)}\right\|} \leq \eta. \tag{4.6}$$

The main result on which the analysis relies, is that in iteration $k$ there always exists $\mathbf{B}_k$ such that $k$ iterations of the original Arnoldi method on the matrix

$$\hat{\mathbf{A}}_k := \mathbf{A} + \mathbf{B}_k,$$

in the absence of errors results in the same subspace as performing $k$ iterations of RA on matrix $\mathbf{A}$ where in each iteration an error $\mathbf{e}^{(k)}$ is made. In these papers, the authors make the assumption that the size of $\|\mathbf{B}_k\|$ is proportional to $\eta$, i.e. there exists a $C_\eta$ such that

$$\|\mathbf{B}_k\| \leq \eta C_\eta, \quad \forall k.$$

The authors indicate that they cannot prove this assumption, although they have some empirical evidence that it is satisfied. Other assumptions were made on the separation of the spectrum of both $\mathbf{A}$ and $\hat{\mathbf{A}}$.

**Theorem 5.** *Assume that the above conditions are satisfied, let $C_X$ be a constant depending on $C_\eta$ and the spectrum of $\mathbf{A}$. If $2\eta C_X < 1$, then*

$$\left\|\hat{\mathbf{x}}_1^{(k)} - \mathbf{x}_1\right\| \leq \frac{\kappa_k}{1 - 2\eta C_X},$$

*where $\kappa_k$ is a bound for the difference between the approximation and the searched eigenvector when no error is made. This value goes to zero as $k$ increases.*

We refer to [41] for a proof and more details. In practice this bound is not (or hardly) computable, so a workable value of $\eta$ needs to be used in practice.

If the method converges, then the question arises how it depends on $\eta$. Empirically, the authors of [40, 41] noticed that the speed of convergence for $\eta$ lower than or equal to $10^{-3}$ stays unchanged but that for $1 \gg \eta > 10^{-3}$ the convergence can slow down a bit. For $\eta$ around 1, we do not have convergence as the norm of the error is as large as the residual norm.

We now bound the norm of the errors $\mathbf{e}_1$ and $\mathbf{e}_2$ such that convergence is still preserved, i.e. such that condition (4.6) is fulfilled with $\eta < 10^{-3}$. Let

$$\frac{\|\mathbf{e}_1\|}{\|\mathbf{x}^{\nu}\|} = \|\mathbf{e}_1\| \leq \eta_X \text{ and } \frac{\|\mathbf{e}_2\|}{\left\|\mathbf{r} + \mathbf{e}_1^{\perp}\right\|} \leq \eta_R \tag{4.7}$$

be tolerances for the relative error, then we obtain starting from (4.5)

$$\begin{aligned}\frac{\|\mathbf{e}\|}{\|\mathbf{r}\|} &\leq \frac{\left\|\mathbf{e}_1^{\perp}\right\|}{\|\mathbf{r}\|} + \frac{\|\mathbf{e}_2\|}{\|\mathbf{r}\|} \\ &= \frac{\left\|(\mathbf{I}-\mathbf{V}\mathbf{V}^T)\mathbf{A}\mathbf{V}\mathbf{e}_1\right\|}{\|\mathbf{r}\|} + \frac{\|\mathbf{e}_2\|}{\|\mathbf{r}\|} \\ &\leq \underbrace{\frac{\|\mathbf{A}\|\,\|\mathbf{e}_1\|}{\|\mathbf{r}\|}}_{\leq \eta/2} + \underbrace{\frac{\|\mathbf{e}_2\|}{\|\mathbf{r}\|}}_{\leq \eta/2}.\end{aligned}$$

The aim is now to bound each term by $\eta/2$. Remark that the first term means that the error propagation due to the matrix-vector product needs to be limited. From (4.7) it follows that

$$\eta_X \leq \frac{\eta\,\|\mathbf{r}\|}{2\,\|\mathbf{A}\|} \tag{4.8}$$

which means that $\eta_X$ scales with the residual norm and the norm of the matrix $\mathbf{A}$. The lower the residual is, the lower $\eta_X$ needs to be. In the example in Section 4.4, we demonstrate that the convergence stagnates if the residual norm is small and $\eta_X$ is kept constant.

Using $\left\|\mathbf{r}+\mathbf{e}_1^{\perp}\right\| \leq \|\mathbf{r}\| + \left\|\mathbf{e}_1^{\perp}\right\|$, we obtain

$$\begin{aligned}\frac{\|\mathbf{e}_2\|}{\|\mathbf{r}\|} &= \frac{\|\mathbf{e}_2\|}{\left\|\mathbf{r}+\mathbf{e}_1^{\perp}\right\|}\frac{\left\|\mathbf{r}+\mathbf{e}_1^{\perp}\right\|}{\|\mathbf{r}\|} \\ &\leq \frac{\|\mathbf{e}_2\|}{\left\|\mathbf{r}+\mathbf{e}_1^{\perp}\right\|}\frac{\|\mathbf{r}\| + \left\|\mathbf{e}_1^{\perp}\right\|}{\|\mathbf{r}\|} \\ &\leq \frac{\|\mathbf{e}_2\|}{\left\|\mathbf{r}+\mathbf{e}_1^{\perp}\right\|}(1+\eta/2).\end{aligned}$$

As this needs to be smaller than $\eta/2$, we must impose that

$$\eta_R \leq \frac{\eta}{2+\eta}. \tag{4.9}$$

## 4.3 Tensor-Arnoldi for one parameter

In the previous sections we already give a glimpse how our algorithm looks like. In this section we discuss in more detail how it works and how the results from

the previous section are applied in the algorithm. In this section we restrict ourself to the case the parameter space is one dimensional. The extension to more parameters is the subject of Section 4.5. We describe first the algorithm in a general form, then we go into more detail on some specific steps of the algorithm.

We first discretise $\Omega$ into points $\omega_1, \ldots, \omega_{n_1}$. We want to compute the desired eigenvalue $\lambda_1$ at these points. For this, we consider the linear operator $\mathcal{A}$ such that

$$\mathcal{A}\mathbf{x} = \begin{bmatrix} \mathbf{A}(\omega_1) & & \\ & \ddots & \\ & & \mathbf{A}(\omega_{n_1}) \end{bmatrix} \begin{bmatrix} \mathbf{x}(\omega_1) \\ \mathbf{x}(\omega_2) \\ \vdots \\ \mathbf{x}(\omega_{n_1}) \end{bmatrix} = \begin{bmatrix} \mathbf{A}(\omega_1)\mathbf{x}(\omega_1) \\ \vdots \\ \mathbf{A}(\omega_{n_1})\mathbf{x}(\omega_{n_1}) \end{bmatrix} \in \mathbb{R}^{nn_1}. \quad (4.10)$$

If $\mathbf{x}$ is reordered as matrix $\mathbf{X} = [\mathbf{x}(\omega_1), \mathbf{x}(\omega_2), \ldots, \mathbf{x}(\omega_{n_1})] \in \mathbb{C}^{n \times n_1}$, then (4.10) is equivalent to an operator $\mathcal{A}(\mathbf{X})$ such that $\mathrm{vec}(\mathcal{A}(\mathbf{X})) = \mathcal{A}\mathbf{x}$. Conceptually, our method boils down to applying RA to (4.10) which is basically applying RA to each $\omega_i$ individually.

In order to efficiently compute $\mathcal{A}(\mathbf{X})$ we need the expansion in (4.1). If a decomposition $\mathbf{X} = \mathbf{U}\mathbf{Z}$ with $\mathbf{U} \in \mathbb{R}^{n \times n_U}$ and $\mathbf{Z} \in \mathbb{C}^{n_U \times n_1}$ is given, then following [38] we obtain

$$\mathcal{A}(\mathbf{X}) = \underbrace{[\mathbf{A}_1\mathbf{U}, \mathbf{A}_2\mathbf{U}, \ldots, \mathbf{A}_m\mathbf{U}]}_{\mathbf{W}_U \in \mathbb{R}^{n \times n_U m}} \underbrace{\begin{bmatrix} \mathbf{Z} \cdot \mathrm{diag}([f_1(\omega_1), f_1(\omega_2), \ldots, f_1(\omega_{n_1})]) \\ \mathbf{Z} \cdot \mathrm{diag}([f_2(\omega_1), f_2(\omega_2), \ldots, f_2(\omega_{n_1})]) \\ \vdots \\ \mathbf{Z} \cdot \mathrm{diag}([f_m(\omega_1), f_m(\omega_2), \ldots, f_m(\omega_{n_1})]) \end{bmatrix}}_{\mathbf{W}_Z \in \mathbb{C}^{n_U m \times n_1}}. \quad (4.11)$$

Remark that

$$\mathbf{Z} \cdot \mathrm{diag}([f_i(\omega_1), \ldots, f_i(\omega_{n_1})]) = [f_i(\omega_1)\mathbf{Z}(:,1), \ldots, f_i(\omega_{n_1})\mathbf{Z}(:,n_1)],$$

which means that the column $j$ of $\mathbf{Z}$ is multiplied with the function $f_i$ evaluated in $\omega_j$.

Let $\mathbf{X}^{\mathcal{V}}$ be the matrix where we place in column $i$ the eigenvector $\mathbf{x}_1^{\mathcal{V}}(\omega_i)$ of the projected eigenvalue problem $\mathbf{V}^T\mathbf{A}(\omega_i)\mathbf{V}$. We first make a low-rank decomposition $\mathbf{U}\mathbf{Z}$ of $\mathbf{X}^{\mathcal{V}}$, then we compute

$$\mathbf{W}_U\mathbf{W}_Z = \mathcal{A}(\mathbf{V}\mathbf{X}^{\mathcal{V}}) = \mathcal{A}((\mathbf{V}\mathbf{U})\mathbf{Z}) \quad (4.12)$$

via (4.11) and we orthogonalise the result with respect to $\mathcal{V}$ to obtain an approximate residual for each parameter value, see Property 4. This is analogous

to lines 5 and 6 in the RA algorithm (see Algorithm 6). We cannot add vectors for all parameter values, so we first orthogonalise (4.12) with respect to $\mathcal{V}$, then we make a low-rank decomposition of the resulting matrix and we add its column space to $\mathcal{V}$. For the low-rank algorithm the method described in [38] is used. We finish an iteration by calculating new eigenvalues and eigenvectors.

This procedure is repeated until all residuals are below a certain tolerance $\eta$. As the dimension of the subspace is prescribed and we add multiple vectors in each iteration, we need to restart the subspace often. The techniques for restarting known in the literature try to delete the part of the subspace that corresponds to an unwanted part of the spectrum but still keep a significant part of the subspace, see [67]. The reason is that not only the Ritz vector associated with the wanted eigenvalue, but also some other Ritz vectors whose eigenvalues are close to $\lambda_1^{\mathcal{V}}$ help improving the convergence to the wanted eigenvalue. This is called *thick restarting*.

In our case, eigenvectors depend on parameters which means that for just representing the wanted eigenvalue for each parameter value, we already need quite a large subspace. If beside the first eigenvector, also eigenvectors associated to other eigenvalues need to be present in the restarted subspace, we can do fewer iterations before we need to restart again. This is a good strategy since it is known from Theorem 2 (see previous chapter) and [63], that the partial derivative of an eigenvector, which is a measure for the change of the eigenvector, depends the most on the eigenvectors whose eigenvalues are the closest to $\lambda_1(\omega)$. As we estimate the eigenvector over the parameter space, we therefore also estimate these eigenvectors. The algorithm can be found in Algorithm 7.

Similar to what was explained in (3.11) in Section 3.3.2, for the calculation of $\mathbf{V}^T\mathbf{A}(\omega_i)\mathbf{V}, i = 1, \ldots, n_1$, we first calculate $\mathbf{V}^T\mathbf{A}_i\mathbf{V}, i = 1, \ldots, m$, and sum these smaller matrices appropriately. As $\mathbf{V}$ is built iteratively, results from the previous iteration can be used if the subspace was not restarted. The lower $\eta_X$ and $\eta_R$, the higher the number of these inner products. In the example in the Section 4.4, we count the number of these inner products and compare them for different values of $\eta_X$ and $\eta_R$.

## 4.4 Error analysis

In Section 4.2.1 we conclude that in the Residual Arnoldi algorithm, $\eta_X$ need to be chosen relative to the current residual and the tolerance $\eta_R$ can be as high as $\dfrac{10^{-3}}{2+10^{-3}}$ without breaking down the convergence. The PRA algorithm is equivalent to applying the RA algorithm to a set of parameter values at the

**Algorithm 7** Parametric Residual Arnoldi (PRA) Algorithm

**Input:** $\mathbf{A}(\omega)$, orthonormal matrix $\mathbf{U} \in \mathbb{R}^{n\times p}$, tolerance $\epsilon$, tolerances $\eta_X$ and $\eta_R$ for the two decompositions, maximal dimension of subspace $n^{\mathcal{V}}$

**Output:** Approximate eigenpair $(\hat{\lambda}_1(\omega_i), \mathbf{y}_1(\omega_i)), i = 1, 2, \ldots, n_1$

$\hat{\mathbf{R}} = [1, 1, \ldots, 1]$
**while** $\left\|\hat{\mathbf{R}}\right\|_F / \sqrt{n_1} > \epsilon$ **do**
  Add $\mathbf{U}$ to the subspace $\mathcal{V}$
  Calculate eigenpair $\left(\lambda_1^{\mathcal{V}}(\omega_i), \mathbf{x}_1^{\mathcal{V}}(\omega_i)\right)$ of $\mathbf{V}^T\mathbf{A}(\omega_i)\mathbf{V}, i = 1, 2, \ldots, n_1$
  $\mathbf{X}^{\mathcal{V}} = [\mathbf{x}_1^{\mathcal{V}}(\omega_1), \mathbf{x}_1^{\mathcal{V}}(\omega_2), \ldots, \mathbf{x}_1^{\mathcal{V}}(\omega_{n_1})]$
  **if** dimension $\mathcal{V} > n^{\mathcal{V}}$ **then**
    $[\mathbf{U}, \mathbf{Z}] = \text{lowrank}(\mathbf{X}^{\mathcal{V}}, 10^{-12})$
    $\mathbf{U} = \mathbf{V}\mathbf{U}$
    Make $\mathcal{V}$ empty
  **else**
    $[\mathbf{U}, \mathbf{Z}] = \text{lowrank}(\mathbf{X}^{\mathcal{V}}, \eta_X)$
    $[\mathbf{W}_U, \mathbf{W}_Z] = \mathcal{A}((\mathbf{V}\mathbf{U})\,\mathbf{Z})$ using (4.11)
    $\mathbf{W}_U^{\perp} = (\mathbf{I} - \mathbf{V}\mathbf{V}^T)\mathbf{W}_U$ ( Orthogonalize $\mathbf{W}_U$ w.r.t $\mathbf{V}$)
    Residual $\hat{\mathbf{R}} := [\hat{\mathbf{r}}(\omega_1), \hat{\mathbf{r}}(\omega_2), \ldots, \hat{\mathbf{r}}(\omega_{n_1})] = \mathbf{W}_U^{\perp}\mathbf{W}_Z$
    $[\mathbf{U}, \sim] = \text{lowrank}(\mathbf{W}_U^{\perp}\mathbf{W}_Z, \eta_R)$;
  **end if**
**end while**
$\hat{\lambda}_1(\omega_i) = \lambda_1^{\mathcal{V}}(\omega_i)$ and Ritz vector $\mathbf{y}(\omega_i) = \mathbf{V}\mathbf{X}^{\mathcal{V}}(\omega_i), i = 1, 2, \ldots, n_1$

same time. In this section we show with an example why it is even favourable to choose $\eta_R$ relatively high, as we need fewer iterations and fewer expensive matrix-vector products. The example is described in Example 7.

**Example 7.** *Consider the convection-diffusion differential equation*

$$f = c_1 u_x + c_2 u_y + d_{11} u_{xx} + 2d_{12} u_{xy} + d_{22} u_{yy} \tag{4.13}$$

*for* $(x, y) \in \Xi = (0, 1) \times (0, 1)$. *The real function* $u$ *has the argument* $(x, y)$. *The constants* $c_1$ *and* $c_2$ *are associated with the convection part and the constants* $d_{11}$, $d_{12}$ *and* $d_{22}$ *are associated with the diffusion part of the equation. It is always assumed that matrix*

$$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} \\ d_{12} & d_{22} \end{bmatrix}$$

*is positive semi-definite. For the discretisation of* $\Xi$ *we use second-order central finite difference formulas and we assume Dirichlet boundary conditions. In this way, we obtain a matrix* $\mathbf{A}$ *depending on the values* $c_1$,$c_2$,$d_{11}$,$d_{12}$, $d_{22}$ *such that*

$$\mathbf{A}(c_1, c_2, d_{11}, d_{22}, d_{12}) = c_1\mathbf{A}_1 + c_2\mathbf{A}_2 + d_{11}\mathbf{A}_3 + d_{22}\mathbf{A}_4 + d_{12}\mathbf{A}_5. \tag{4.14}$$

*For this first example we take* $c_2 = 1, d_{11} = 1.1, d_{22} = 1, d_{12} = 1$ *and* $c_1$ *takes values in the interval* $\Omega = [-2.5, 2.5]$. *We discretize this interval in* 100 *points. This means we can rewrite* (4.13) *as*

$$\mathbf{A}(c_1) = \mathbf{A}_0 + c_1\mathbf{A}_1,$$

*with* $\mathbf{A}_0 = c_2\mathbf{A}_2 + d_{11}\mathbf{A}_3 + d_{22}\mathbf{A}_4 + d_{12}\mathbf{A}_5$. *The desired eigenvalue is the eigenvalue with largest real part, in this case the same as the eigenvalue closest to zero. The region* $\Xi$ *is discretised into* 100 *points in each dimension, so the resulting matrices* $\mathbf{A}_i$ *are of size* $10^4 \times 10^4$. *The results can be found in Figure 4.1 and Table 4.1. Remark that we need many iterations but tests with individual parameters values reveal that also the Arnoldi algorithm needs many iterations. We choose an example where the convergence is slow to be able to see clearly what effect different values for* $\eta_X$ *and* $\eta_R$ *have on the convergence.*

At line 11 on the one hand, and at line 15 of the PRA algorithm on the other, we make low-rank approximations of the matrix consisting of the eigenvector resp. the approximate residual and so introducing errors for each parameter value $\mathbf{E}_1 := [\mathbf{e}_1(\omega_1), \mathbf{e}_1(\omega_2), \ldots, \mathbf{e}_1(\omega_{n_1})]$ and $\mathbf{E}_2 := [\mathbf{e}_2(\omega_1), \mathbf{e}_2(\omega_2), \ldots, \mathbf{e}_2(\omega_{n_1})]$. We first discuss the error $\mathbf{E}_1$. The low-rank approximation is such that the relative error is bounded by the tolerance $\eta_X$, i.e

$$\frac{\|\mathbf{E}_1\|_F}{\|\mathbf{X}^{\nu}\|_F} \leq \eta_X. \tag{4.15}$$

Since $\left\|\mathbf{X}^{\nu}\right\|_F = \sqrt{n_1}$, it follows that $\|\mathbf{E}_1\|_F \leq \eta_X\sqrt{n_1}$. From this, it can be seen that

$$\|\mathbf{e}_1(\omega_i)\|^2 \leq \eta_X^2 n_1, \quad i = 1, 2, \ldots, n_1,$$

but assuming the error is uniformly distributed over the sample points, we can state that

$$\|\mathbf{e}_1(\omega_i)\| \lessapprox \eta_X. \tag{4.16}$$

Figure 4.1 (a) shows the average residual over the parameter and we observe that for $\eta_X = 10^{-3}$ the convergence slows down after 200 iterations. This is related to the average relative error $\mathbf{e}_1$ plotted in Figure 4.1 (b), as this becomes too high such that condition (4.8) is not fulfilled. Analoguously to (4.16) we can prove that

$$\frac{\|\mathbf{e}_2(\omega_i)\|}{\left\|\mathbf{r}(\omega_i) + \mathbf{e}_1^{\perp}\right\|} \lessapprox \eta_R, \quad i = 1, 2, \ldots, n_1.$$

As seen in (4.9), we only need to make sure that $\eta_R \leq \dfrac{\eta}{2+\eta}$, where $\eta$ can be chosen as high as $10^{-3}$. This explains why in Figure 4.1 (a) choosing

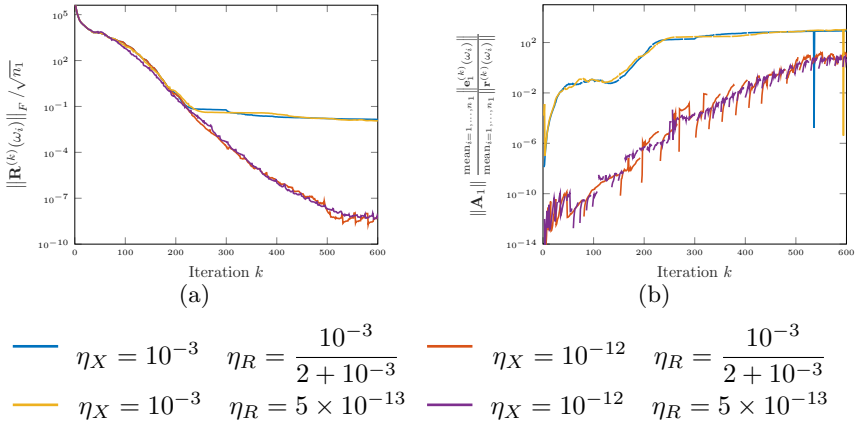



Figure 4.1: Results of the PRA algorithm for different tolerances $\eta_X$ and $\eta_R$ with maximal subspace dimension 150 for Example 7. In (a) we plot the Frobenius norm of the residual, scaled with the number of sample points. In (b) the average first error $\mathbf{e}_1$ relative to the average residual, scaled with the norm of $\mathbf{A}_0$ is shown. As long as the value is below or around $10^{-3}$, the algorithm converges as if there were no error, see the criterium in (4.8). If the value is large ($> 1$) convergence is lost. The almost vertical lines in (b) are present just after iterations where we needed to restart the subspace.

$\eta_R = \dfrac{10^{-3}}{2+10^{-3}}$ does not harm convergence. Furthermore, it seems even that convergence is faster for higher $\eta_R$. This phenomenon is intriguing and cannot be explained only by a relative error plot. Commonly, convergence of Arnoldi-type algorithms slows down with each restart. It is generally known in Arnoldi-type algorithms that, with each restart, convergence can slow down. The higher the tolerance $\eta_R$ the fewer vectors are added to the subspace, so we do not need to restart so often. This means that we can do more iterations within the same subspace so the convergence is faster.

In Table 4.1, we see statistics about the computational cost. As expected, we observe that the number of restarts increases as the tolerances decrease and that the average size of the subspace is independent of the chosen tolerances. As indication of the increased computational cost when taking the tolerances smaller, we count the number of inner products when creating the reduced eigenvalue problem (line 4 in the PRA algorithm). A more detailed explanation on this was given at the end of Section 4.5.1.

In the experiments in Figure 4.1 both $\eta_X$ and $\eta_R$ are kept constant. We observe

that if $\eta_X$ is chosen high, that the convergence is disrupted when the norm of the error $\mathbf{e}_1$ becomes high relative to the residual norm. The solution for this is to choose the tolerance $\eta_X$ in iteration $k$ relative to the residual in the previous iteration. In Figure 4.2, we compare results when holding $\eta_X$ constant and when we adapt its value relatively with respect to the residual. We see that we obtain similar results. This is explained by Figure 4.2 (b) as we see that the error $\mathbf{e}_1$ relative to the residual remains constant.

| | Number of restarts | Avg. size of subspace | Avg. number of inner products for reduced eigenvalue problem |
|---|---|---|---|
| $\eta_X = 10^{-3}$ $\eta_R = 5 \times 10^{-4}$ | 7 | 79.5233 | 691.74 |
| $\eta_X = 10^{-12}$ $\eta_R = 5 \times 10^{-4}$ | 16 | 78.7383 | 1429.9167 |
| $\eta_X = 10^{-3}$ $\eta_R = 5 \times 10^{-13}$ | 15 | 79.545 | 1272.9133 |
| $\eta_X = 10^{-12}$ $\eta_R = 5 \times 10^{-13}$ | 38 | 81.0283 | 3434.36 |

Table 4.1: Statistics associated to the results shown in Figure 4.1.

The conclusion is that we obtain the best results when taking $\eta_R$ high and taking $\eta_X$ relative to the residual. In this way the norm of both errors $\mathbf{e}_1$ and $\mathbf{e}_2$ stays constant with respect to the residual, which preserves convergence if this constant is low enough ($< 10^{-3}$), see Section 4.4.

## 4.5 More than one parameter

If there is more than one parameter, the PRA algorithm has a natural extension using tensors. The TT-tensor format introduced in Section 2.2.2 seemed to us the best way to represent these tensors. This format does not only allow us to represent all tensors efficiently, also all necessary computations have elegant counterparts if the tensors are in this format. Before we dive into this, we first introduce the used notation.

We assume that $\Omega$ is a Cartesian product of intervals in $\mathbb{R}$, so

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_d = [a^1, b^1] \times [a^2, b^2] \times \ldots \times [a^d, b^d].$$

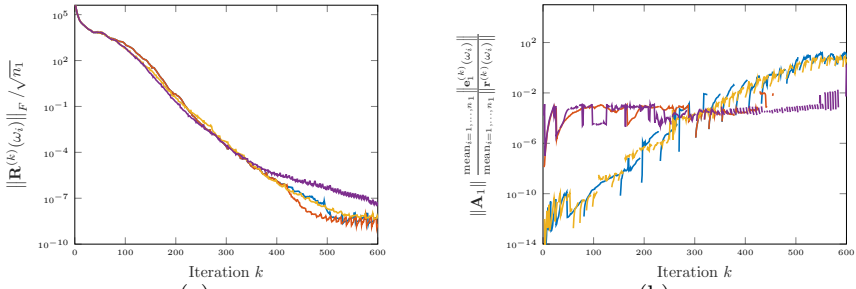



Figure 4.2: Results of the PRA algorithm for different tolerances $\eta_X$ and $\eta_R$ with maximal subspace dimension 150 for Example 7. We compare the results when taking a constant low tolerance $\eta_X$ with the case that $\eta_X$ is relative to the residual. In (a) we plot the Frobenius norm of the residual, scaled with the number of samples. We see that in all four cases the algorithm converges. In (b), the average error $\mathbf{e}_1$ relative to the average residual, scaled with the norm of $\mathbf{A}_1$ is shown. As expected, we see that this error relative to the residual stays constant.

The discretisation is done in the same way as in (2.5) in Section 2.2

All tensor equivalents are the calligraphic version of the letters used in the one parameter case. For example, the approximate eigenvectors $\hat{\mathbf{x}}_1(\omega_{i_1},\ldots,\omega_{i_d})$ are stored in tensor $\mathcal{X}\in\mathbb{C}^{n\times n_1\times\ldots\times n_d}$ where

$$\mathcal{X}(:,i_1,i_2,\ldots,i_d)=\hat{\mathbf{x}}_1(\omega_{i_1},\ldots,\omega_{i_d}).$$

We define tensor $\mathcal{F}\in\mathbb{R}^{m\times n_1\times n_2\times\ldots\times n_d}$ as the tensor with all function values, so

$$\mathcal{F}(i_0,i_1,\ldots,i_d)=f_{i_0}\left(\omega_{i_1},\ldots,\omega_{i_d}\right)$$

for a parameter value $\omega=(\omega_{i_1},\omega_{i_2},\ldots,\omega_{i_d})$. If $d=1$, this is the matrix

$$\mathbf{F}=\begin{bmatrix} f_1(\omega_1) & f_1(\omega_2) & \ldots & f_1(\omega_{n_1}) \\ f_2(\omega_1) & f_2(\omega_2) & \ldots & f_2(\omega_{n_1}) \\ \vdots & \vdots & \ddots & \vdots \\ f_m(\omega_1) & f_m(\omega_2) & \ldots & f_m(\omega_{n_1}) \end{bmatrix}.$$

In comparison with Algorithm 6, following parts need to be adapted.

1. Computing a decomposition of $\mathcal{W}_Z$ given the decomposition of $\mathcal{Z}$ (line 12 in the PRA Algorithm (Algorithm 7) ).
2. Calculating a basis for the column space.
3. Rounding a tensor up to a given tolerance where one is only interested in the column space (see line 15)

We start by explaining how we calculate a TT-decomposition of $\mathcal{W}_Z$ (the equivalent of $\mathbf{W}_Z$ in (4.11)), directly from the decomposition of $\mathcal{Z}$. To see this, we recall the case of $d = 1$. Let $\mathbf{1}_q$ be a vector with all ones of length $q$. We first note that

$$\mathbf{Z} \cdot \text{diag}([f_i(\omega_1), f_i(\omega_2), \ldots, f_i(\omega_{n_1})]) = \mathbf{Z}.{}_H(\mathbf{1}_{n_U}[f_i(\omega_1), f_i(\omega_2), \ldots, f_i(\omega_{n_1})])$$

from which it follows that

$$\mathbf{W}_Z = ((\mathbf{1}_{n_U} \otimes \mathbf{I}_m)\mathbf{F}) \cdot_H ((\mathbf{I}_{n_U} \otimes \mathbf{1}_m)\mathbf{Z}).$$

This means that in case of multiple parameters, $\mathcal{W}_Z$ can be generalised to

$$\mathcal{W}_Z = (\mathbf{1}_{n_U} \otimes \mathbf{I}_m, \mathbf{I}, \ldots, \mathbf{I}) .\mathcal{F}.{}_H (\mathbf{1}_{n_U} \otimes \mathbf{1}_m, \mathbf{I}, \ldots, \mathbf{I}) .\mathcal{Z}.$$

From [50], we obtain that

$$\begin{cases} \mathbf{G}_{j+1}^{W_Z}(i_j) &= \mathbf{G}_{j+1}^{F}(i_j) \otimes \mathbf{G}_{j+1}^{Z}(i_j) \qquad j = 1, 2, \ldots, d, i_j = 1, 2, \ldots, n_j, \\ \mathbf{G}_1^{W_Z} &= \mathbf{G}_1^F \otimes \mathbf{G}_1^Z. \end{cases} \tag{4.17}$$

Note that the TT-decomposition is not unique. Moreover, it is possible that the ranks of the decomposition in (4.17) of the tensor $\mathcal{W}_Z$ are overestimated. Calculating a TT-decomposition up to a given tolerance was discussed in Algorithm 2 in Section 2.2.2. At line 7 and 11 of our algorithm, we want to approximate a possibly complex tensor $\mathcal{X}^{\mathcal{V}}$ up to a given tolerance and keeping the basis $\mathbf{U}$ for the first mode real-valued.

For the rounding at line 15, the first for-loop of Algorithm 3 in Section 2.2.2 is fully applied but as we are only interested in a basis for the first mode we only do the first iteration from the second for-loop. Moreover, we make sure that the basis is real.

If all functions $f_i, i = 1, 2, \ldots, m$, are separable, then there is no need to first calculate a TT-decomposition of $\mathcal{F}$. We can get a decomposition directly from the separability of the functions $f_i$. As the functions $f_i$ are separable there are functions $g_{i_j}^i(\omega_{i_j})$ such that $f_i(\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_d}) = g_{i_1}^i(\omega_{i_1}) g_{i_2}^i(\omega_{i_2}) \ldots g_{i_d}^i(\omega_{i_d})$.

It can be seen that a possible TT-decomposition is of the form

$$\begin{cases} \mathbf{G}_1^F & = & \mathbf{I}_m & \\ \mathbf{G}_{j+1}^F(i_j) & = & \operatorname{diag}([g_1^j(\omega_{i_j}), g_2^j(\omega_{i_j}), \ldots, g_m^j(\omega_{i_j})]) & j = 1, 2, \ldots, d-1, \\ & & & i_j = 1, 2, \ldots, n_j, \\ \mathbf{G}_{d+1}^F(i_d) & = & [g_1^d(\omega_{i_d}), g_2^d(\omega_{i_d}), \ldots, g_m^d(\omega_{i_d})]^T & j = d, i_d = 1, 2, \ldots, n_d. \end{cases} \tag{4.18}$$

In this case, all interior ranks are $m$. This means that the interior ranks of the tensor $\mathcal{W}_Z$ are maximal $mr_i^Z, i = 0, 1, \ldots, d$. Note that also here the ranks of the decomposition are possibly overestimated. The lower are the ranks of the tensor $\mathcal{Z}$, the more efficient the other operations are. To have this, it is important how the eigenvectors are stored in the tensor $\mathcal{X}^{\mathcal{V}}$. Eigenvectors are only unique up to a complex constant, but for the ranks of the tensor $\mathcal{X}^{\mathcal{V}}$ it makes a difference which constant is used. We illustrate this in Example 8. The best we can do is to compare the eigenvector $\mathbf{x}(\omega)$ for a point $\omega$ with a certain vector $\mathbf{a}$, and store instead of $\mathbf{x}(\omega)$ the vector $\dfrac{(\mathbf{a}^H\mathbf{x}(\omega))}{\|\mathbf{a}^H\mathbf{x}(\omega)\|}\mathbf{x}(\omega)$. Remark that if $\mathbf{a}$ and the eigenvector have no complex elements then this boils down to multiplying with 1 or $-1$. A possible choice for $\mathbf{a}$ is to take the eigenvector corresponding to a sample point.

**Example 8.** *Let $\mathbf{A}(\omega_1, \omega_2)$ be a $1 \times 1$-matrix, depending on two parameters $\omega_1$ and $\omega_2$ in a parameter space $[a_1, b_1] \times [a_2, b_2]$. This means that the eigenvector is equal to $-1$ or $1$. We discretise both intervals in three. The tensor $\mathcal{X} \in \mathbb{R}^{1\times 3\times 3}$ representing the eigenvector over the parameters is*

$$\mathcal{X}(1, :, :) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix},$$

*so the rank is one. If for one of the parameter values we use as eigenvector $-1$ instead of $1$, then*

$$\mathcal{X}(1, :, :) = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},$$

*so the rank is two. This means that the ranks of the decomposition depend on the constant that is used for the eigenvector.*

If there are not many points in each dimension, it might be better not to use tensors but to reshape the tensor $\mathcal{X}^{\mathcal{V}}$ as a matrix and to apply the PRA algorithm as described in Algorithm 7. The advantage of using a decomposition of the tensor is to reduce the storage and to reduce the number of operations,

but if the number of parameter values is low in each dimension, then we cannot gain from the TT-representation as the cost of decomposing is too high in comparison with the possible gain in the remaining steps of the algorithm.

### 4.5.1 Numerical Experiments

In this section we extend Example 7 to multiple parameters and give the convergence results.

**Example 9.** *Consider the convection-diffusion differential equation* (4.13) *in Example 7. We consider the case where* $d_{12} = 0$*, we vary the variables* $c_1$ *and* $c_2$ *both in the interval* $[4, 6]$ *and we let the variable* $d := d_{11} = d_{22}$ *vary in the interval* $[-1.1, -0.9]$*. In this way we can rewrite* (4.14) *into the form*

$$\mathbf{A}(c_1, c_2, d) = c_1\mathbf{A}_1 + c_2\mathbf{A}_2 + d\mathbf{A}_3.$$

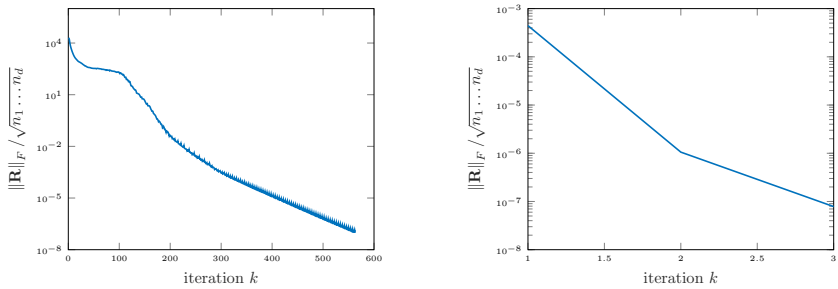
*The desired eigenvalue is the eigenvalue with smallest real part, in this case the same as the eigenvalue closest to zero. The matrices are of size* $10^3 \times 10^3$ *and we choose* $n_1 = n_2 = n_3 = 30$ *resulting in a total of* $n_0 = 27 \times 10^3$ *parameter values. As tolerances we choose* $\eta_R = \dfrac{10^{-3}}{2 + 10^{-3}}$ *and* $\eta_X^{(k)} = 10^{-3} \left\|\hat{\mathbf{R}}^{(k)}\right\|_F / \sqrt{n_0}$

We first solve the problem discretised in each dimension by 10 interior discretisation points. We see the result in Figure 4.3 (a). We observe that we need many iterations for convergence to the solution which is not surprising as using Arnoldi's method for all parameter points separately needs approximately the same number of iterations. The obtained subspace $\mathcal{V}$ is then used as initial subspace when discretising each dimension in 30. We see in Figure 4.3 (b) that we only need a few iterations.

## 4.6 Parametric inverted Residual Arnoldi method

In each iteration of the residual Arnoldi method, we applied the operator $\mathcal{A}$ (see (4.10)) or its tensor counterpart to Ritz vectors contaminated with error. In this way we are able to compute eigenvalues which are extreme in some sense. Users are often interested in the eigenvalue close to a certain shift $\sigma$. For this, the shift-inverted Arnoldi method can be used. We assume for now that $\sigma = 0$.

In this case we need to be able to efficiently compute linear systems with $\mathbf{A}(\omega)$ for all parameter samples. The inverse of $\mathbf{A}(\omega)$ can in general not be written in




(a) (b)

Figure 4.3: Results of the PRA algorithm with maximal subspace dimension 100 for Example 9. In (a), we see the convergence when taking $n_1 = n_2 = n_3 = 10$. We see that we need quite a large number of iterations but tests showed us that using Arnoldi for all parameter points separately needs approximately the same number of iterations. If we take the resulting subspace $\mathcal{V}$ and we use this as the initial subspace for $n_1 = n_2 = n_3 = 30$, we see that we just need a few iterations

the form (4.1). However if all matrices except one is of rank one, i.e

$$\mathbf{A}(\omega) = f_0(\omega)\mathbf{A}_0 + \sum_{i=1}^{m} f_i(\omega)\mathbf{A}_i \tag{4.19}$$

with $\mathbf{A}_i = \mathbf{u}_i\mathbf{q}_i^T, i = 1, \dots, m$ with $m$ a small number, it can be proven via the Sherman-Morrison-Woodbury formula [26, 104] that for a vector $\mathbf{q}$, it holds that

$$\mathbf{A}(\omega)^{-1}\mathbf{q} \in \text{span}(\mathbf{A}_0^{-1}\mathbf{q}, \mathbf{A}_0^{-1}\mathbf{u}_1, \mathbf{A}_0^{-1}\mathbf{u}_2, \dots, \mathbf{A}_0^{-1}\mathbf{u}_m). \tag{4.20}$$

We see that there is only one basis vector of the subspace that depends on $\mathbf{q}$, while other vectors are the same for any $\mathbf{q}$ and for all parameter samples. Hence, one could keep all other vectors during all iterations in subspace $\mathcal{V}$.

It is possible to directly apply the PRA algorithm on $\mathbf{A}(\omega)^{-1}$, however, in [40, 41], the authors proposed an adaptation of RA. In exact arithmetic, if we apply the RA algorithm on $\mathbf{A}^{-1}$, we add the residual vector $\mathbf{r}_{A^{-1}} = \mathbf{A}^{-1}\mathbf{y} - \frac{1}{\lambda}\mathbf{y}$ with $(\lambda, \mathbf{y})$ a Ritz-pair of $\mathbf{A}$. The authors proposed to first calculate the residual $\mathbf{r} = \mathbf{A}\mathbf{y} - \lambda\mathbf{y}$, to solve the system

$$\mathbf{A}\mathbf{u} = \mathbf{r}. \tag{4.21}$$

and to add the resulting $\mathbf{u}$ to the subspace. The authors in [40, 41] proved that $\mathbf{u}$ is parallel to $\mathbf{r}_{A^{-1}}$.

As the authors of [40, 41] claim that also here the error proportional to the residual can be as high as $10^{-3}$ without harming the convergence, it is more appropriate to calculate the vector via (4.21). Their method was called the Shift-and-invert Residual Arnoldi algorithm and similarly to the PRA algorithm, we extend this method to the parametric case. Instead of (4.21) we only need to calculate

$$\mathbf{A}_0\mathbf{u}(\omega) = \mathbf{r}(\omega)$$

upto a relative tolerance $\epsilon_R$, as we keep the other basis vectors in (4.20) in the subspace $\mathcal{V}$ in all iterations. The tolerance for the relative error when solving (4.21) is denoted with $\eta_R$ and the tolerance for the low-rank approximation of the eigenvector over the parameter, is denoted with $\eta_X$.

If $\sigma \neq 0$, then we need to assume $f_0(\omega) = 1$, and we redefine $\mathbf{A}_0$ as

$$\mathbf{A}_0 - \sigma\mathbf{I}$$

in (4.19).

### 4.6.1 Numerical experiments

The following examples arise from spectrum-based stabilization methods for time-delay systems [18, 46]. Time-delay stability can be inferred from the spectral abscissa (real part of the rightmost eigenvalue) of an eigenvalue problem, discretizing the infinitesimal generator operator by the collocation approach [7]. Spectrum-based stabilization is based on minimizing the spectral abscissa as a function of controller parameter and favors situations where the spectral abscissa is not smooth, since minima of the spectral abscissa are often characterized by defective rightmost eigenvalues [19]. In this context, our novel approach may lead to a significant contribution in probabilistic spectrum-based stabilization methods [18] since it permits to tackle larger scale problems with higher stochastic dimension.

**Example 10.** *The spectral abscissa function, associated to the oscillator with feedback delay system [19, Example 2], is approximated by the Infinitesimal Generator approach [7] as the spectral abscissa of the following parametric matrix:*

$$\mathbf{A}(\omega_1, \omega_2; K_1, K_2) = \mathbf{A}_0(K_1, K_2) - \omega_1^2\mathbf{A}_1 + -2\omega_1\omega_2\mathbf{A}_2, \qquad (4.22)$$

$$\textit{with } \omega_1 \in [0.9, 1.1], \omega_2 \in [0.1, 0.2].$$

*The non-singular matrix* $\mathbf{A}_0$ *and the rank-*1 *matrices* $\mathbf{A}_1$, $\mathbf{A}_2$ *have dimension* $1002 \times 1002$ *and are given by:*

$$\mathbf{A}_0(K_1, K_2) = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & K_1 & K_2 \\ & & & \mathbf{D}_M \otimes \mathbf{I}_2 & & & \end{pmatrix}$$

$$\mathbf{A}_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T, \ \mathbf{A}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T,$$

*where* $\mathbf{D}_M \in \mathbb{R}^{M \times M}$ *is the differentiation matrix [71, Chapter 6] obtained with* $M = 500$ *Chebyshev nodes as collocation points. The controller parameters* $K_1$ *and* $K_2$ *are selected so that the spectral abscissa function presents different behaviors on the parametric domain* $(\omega_1, \omega_2) \in [0.9, 1.1] \times [0.1, 0.2]$ *as illustrated in Figures 4.4 (a) and (b): for* $(K_1, K_2) = [0.2, 0.2]$ *the spectral abscissa smoothly behaves, while for* $(K_1, K_2) = [0.6179, -0.0072]$ *the spectral abscissa presents non-Lipschitz continuous points, due to multiple non-semi-simple rightmost eigenvalues. We refer to [19] for further information on the example.*

*We use* $\eta_R = 10^{-3}$ *fixed and* $\eta_X^{(k)} = 10^{-3} \left\| \hat{\mathbf{R}}^{(k)} \right\|_F / \sqrt{n_0}$ *relative to the residual as explained in Section 4.4. The parameter intervals are both discretised into* 30 *points, so we have a total of* $n_0 = 900$ *points for which the spectral abscissa needs to be found. The spectral abscissa is plotted in Figure 4.4 (a)-(b). The convergence results are shown in Figure 4.4 (c)-(d).*

**Example 11.** *We analyse the spectral abscissa function associated to the heat transfer set up with two static feedback controllers* $\mathbf{K}$*, designed by probabilistic stabilization method considering the temperature on the left cooler and both coolers are affected by uncertainty [18, Section 5.3]. The spectral abscissa function is determined by the eigenvalues of the following* $1030 \times 1030$ *parametric matrix*

$$\mathbf{A}(\omega_1, \omega_2; \mathbf{K}) = \mathbf{A}_0(\mathbf{K}) + \frac{1 - \omega_1}{\omega_1} \mathbf{A}_1 + \frac{1 - \omega_2}{\omega_2} \mathbf{A}_2,$$

$$\text{with } \omega_1 \in [13.5, 16.5], \omega_2 \in [15.3, 18.7].$$

*The two static controllers* $\mathbf{K}$*, designed in order to minimize the mean of the spectral abscissa, modelling the uncertainty on the temperatures by independent uniform random variables [18, Section 5.3], lead to different behaviors of the*

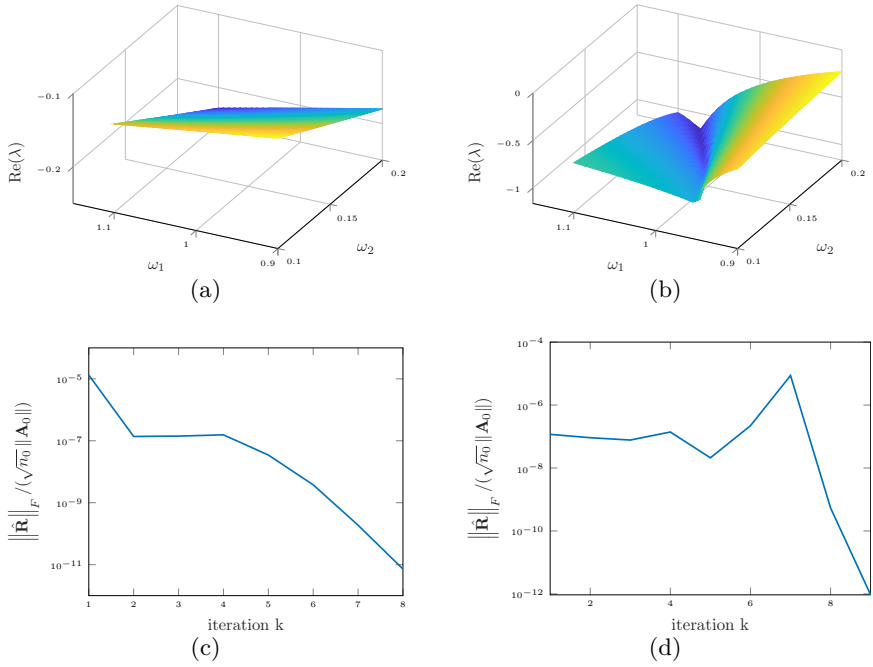



Figure 4.4: The spectral abscissa over the parameters for the two different cases in Example 10. We see in (a) that the spectral abscissa varies smoothly over the parameters, in (b) this is not the case anymore. The convergence results for the two different cases in Example 10 are plotted in (c) and (d). We see that in the two cases the algorithm converges within nine iterations.

*spectral abscissa. In Figure 4.5 (a) resp. (b), the spectral abscissa for* $\mathbf{K} = [0.5105, -9.1810 \times 10^{-2}]$ *resp.* $\mathbf{K} = [0.45043, -0.1631]$ *is used. The parameter intervals are both discretised into* 30 *points, so we have a total of* $n_0 = 900$ *parameter values for which the spectral abscissa needs to be found. We use* $\eta_R = 10^{-3}$ *fixed and* $\eta_X^{(k)} = 10^{-3} \left\|\hat{\mathbf{R}}^{(k)}\right\|_F / \sqrt{n_0}$ *relative to the residual as explained in Section 4.4. The convergence results are shown in Figure 4.5 (c)-(d).*

We see in Figures 4.4 (c)-(d) and 4.5 (c)-(d) that in both examples only a few iterations were needed. In Figure 4.4 (b) and 4.5 (a)-(b) it can be observed that the spectral abscissa over the parameter is not differentiable as in some points (we only count eigenvalues with imaginary part equal or greater than zero) there are multiple eigenvalues for which the real part collides. We need to be careful when some eigenvalues of the large eigenvalues are close to each

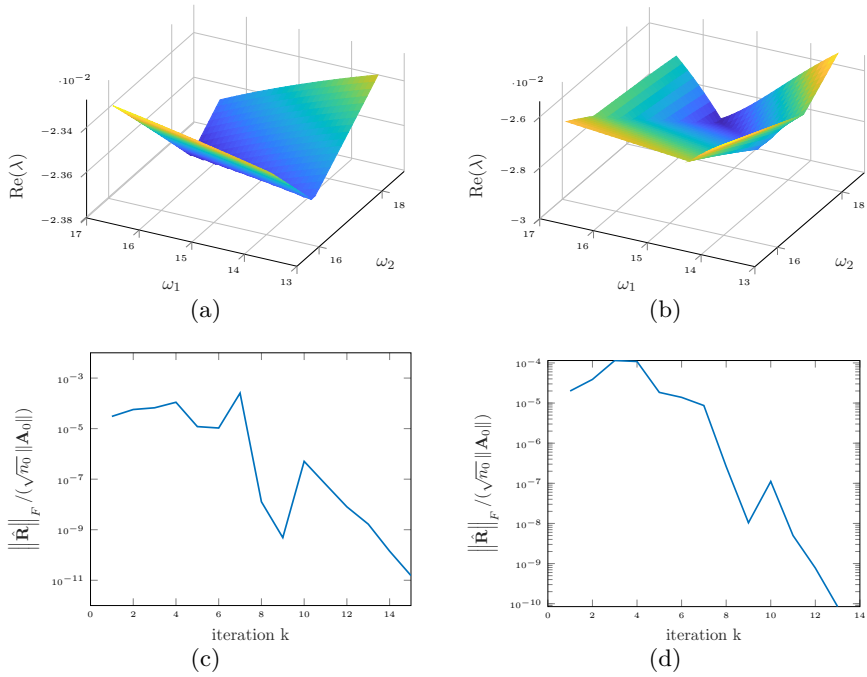



Figure 4.5: In (a)-(b), The spectral abscissa over the parameters for the two different cases in Example 11. The convergence results for the two different cases in Example 11 can be seen in (c)-(d).

other. In our experience, projection methods for calculating eigenvalues of non-normal matrices can fail to converge if the searched eigenvalue is close to other eigenvalues. Therefore, it is possible that due to the projection an eigenvalue is selected which is not the one we look for. The technique that we apply, is to save for every parameter sample in iteration $k$ five eigenvectors of the projected problem that best satisfies our criterium. In the folowing iteration $k+1$ only an eigenvalue with associated eigenvector is selected if the angle between this eigenvector and one of the five selected eigenvectors in iteration $k$ is below a prescribed tolerance.

## 4.7 Conclusion

We proposed a method to calculate eigenvalues of a parametrized matrix $\mathbf{A}(\omega)$ for a number of parameter values at the same time. We started with introducing

the Residual Arnoldi algorithm, which is a version of the Arnoldi algorithm that is robust against introduced errors. We applied the Residual Arnoldi Algorithm on a large number of sample points at the same time. To keep the computations feasible, we made low-rank approximations. We made low-rank approximations of the searched eigenvectors over the parameter (with relative tolerance $\eta_X$) and of the residual over the parameter (with relative tolerance $\eta_R$). We saw that the total error relative to the residual needs to be smaller than $10^{-3}$, from which we deduce that $\eta_X$ needs to be chosen proportional to the residual and that $\eta_R$ can be chosen as high as $10^{-3}/(2 + 10^{-3})$.

We illustrated with numerical experiments that a higher $\epsilon_R$ even improves performance, since a higher tolerance leads to a slower increase of the subspace dimension and consecutively fewer restarts. At the end of Section 4.6, we made the extension to multiple parameters and to shift-inverted Arnoldi for a specific range of problems. A possible extension is to make a version of the parametric shift-inverted Arnoldi method when more than one matrix has rank higher than one. The difficulty in this problem is that we need to be able to solve parametrized linear systems efficiently. Another possibility for further research is the calculation of statistical moments of certain eigenvalues over the parameter.

# Chapter 5

# Solving multiparameter eigenvalue problems via tensor-train representations

*"Je moet heel diep nadenken over niets.*
*Dan komt het opeens", zei Kees*
*"Dat heet inspiratie"*
*(Daan Zonderland, De avonturen van Jeroen en zijn vrienden)*

Parts of this chapter are copied from:

RUYMBEEK K., MEERBERGEN K., AND MICHIELS W.
"Subspace method for multiparameter-eigenvalue problems based on tensor-train representations", submitted for publication to Numerical Linear Algebra with Applications

## 5.1 Introduction

In this chapter we solve a different problem than in the previous chapters but we use the same type of method, i.e. a projection method. Recall from Section 1.4 the problem of computing solutions $((\mu_1, \mu_2, \ldots, \mu_m), (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m))$ of

the *m-parameter eigenvalue problem* (mEP)

$$\begin{cases} \mathbf{A}_1\mathbf{x}_1 &= \mu_1\mathbf{B}_{11}\mathbf{x}_1 + \ldots + \mu_m\mathbf{B}_{1m}\mathbf{x}_1 \\ \mathbf{A}_2\mathbf{x}_2 &= \mu_1\mathbf{B}_{21}\mathbf{x}_2 + \ldots + \mu_m\mathbf{B}_{2m}\mathbf{x}_2 \\ & \vdots \\ \mathbf{A}_m\mathbf{x}_m &= \mu_1\mathbf{B}_{m1}\mathbf{x}_m + \ldots + \mu_m\mathbf{B}_{mm}\mathbf{x}_m \end{cases}, \tag{5.1}$$

where $\mathbf{A}_i, \mathbf{B}_{ij} \in \mathbb{R}^{n_i \times n_i}, i, j = 1, \ldots, m$. We call the $m$-tuple $(\mu_1, \mu_2, \ldots, \mu_m)$ an *eigenvalue-tuple* and $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ the associated *eigenvector-tuple*. Here, we aim for the eigenvalue-tuples whose $k$th component, $\mu_k$ has smallest modulus. Without loss of generality we assume in this chapter that $k = m$.

As seen in Section 1.4, it can be proven that (5.1) is equivalent with the generalised eigenvalue problems

$$\boldsymbol{\Delta}_i\mathbf{x} = \mu_i\boldsymbol{\Delta}_0\mathbf{x}, \quad i = 0, 1, \ldots m, \tag{5.2}$$

where these $\boldsymbol{\Delta}$-matrices are defined as

$$\boldsymbol{\Delta}_i := \left| \begin{array}{ccccccc} \mathbf{B}_{11} & \ldots & \mathbf{B}_{1,i-1} & \mathbf{A}_1 & \mathbf{B}_{1,i+1} & \ldots & \mathbf{B}_{1m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{m1} & \ldots & \mathbf{B}_{m,i-1} & \mathbf{A}_m & \mathbf{B}_{m,i+1} & \ldots & \mathbf{B}_{mm} \end{array} \right|_{\otimes}, i = 1, \ldots, m. \tag{5.3}$$

The eigenvalue $\mu_i$ is the $i$th element of an eigenvalue-tuple of (5.1) and eigenvectors $\mathbf{x}$ of (5.2) are of the form $\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \ldots \otimes \mathbf{x}_m$ where the $\mathbf{x}_i, i = 1, \ldots, m$, together form an eigenvector-tuple of (5.1). Note that we assume here that $\boldsymbol{\Delta}_0$ is non-singular.

The algorithm that we present, uses the same ideas as the algorithm in [12]. To this end, we prove that the $\boldsymbol{\Delta}$-matrices in (5.3) can be written in the Tensor Train-format (TT). The aforementioned paper only presented a subspace algorithm for symmetric standard eigenvalue problems (with the matrix written in TT-format) which implies that the smallest eigenvalues can be interpreted via a variational characterisation as the result of an optimisation problem. This approach does not apply to the generalised eigenvalue problem $(\boldsymbol{\Delta}_m, \boldsymbol{\Delta}_0)$. We need to pay extra attention in selecting the wanted eigenvalues of the projected problem. Furthermore we have the additional information that in our problem the eigenvectors are of rank one, which we can exploit to increase efficiency. We show the effectiveness of our algorithm by numerical examples.

The structure of this chapter is as follows. In Section 5.2 we explain the method in [12] that solves a related problem. We explain and motivate the changes to this method in Section 5.4. In Section 5.5 we show with some

numerical experiments the effectiveness of our algorithm and the dependency of our algorithm on $m$ and the size of the matrices in a $m$EP. We close this chapter with conclusions. The Matlab code can be downloaded following

`http://twr.cs.kuleuven.be/research/software/delay-control/mEP/`.

## 5.2 Calculating eigenvalues of symmetric matrices as TT-operators

We describe the method proposed in [12] to calculate the smallest eigenvalues of a symmetric standard eigenvalue problem where the matrix is represented in Tensor-Train (TT) format. A general description of tensors and this format can be found in Section 2.2.

Let $\mathbf{X} \in \mathbb{R}^{n_1 n_2 \ldots n_m \times b}$ be the current estimate of the eigenvectors for $b$ eigenvalues in block-TT-format with index $k$, so

$$\mathbf{X}(:, i_b) = \sum_{j_1=1}^{r_1} \sum_{j_2=1}^{r_2} \ldots \sum_{j_{m-1}=1}^{r_{m-1}} \mathcal{G}_1(:, j_1) \otimes \mathcal{G}_2(j_1, :, j_2) \otimes \ldots \tag{5.4}$$

$$\otimes \hat{\mathcal{G}}_k(j_{k-1}, :, i_b, j_k) \otimes \ldots \otimes \mathcal{G}_m(j_{m-1}, :).$$

The idea behind the method is that we update one mode, leaving the other modes fixed. We start with updating the first mode, then we update the second mode and we continue this process until the $m$th mode. When this is finished, we go back in reversed order towards the first mode. This process is called a *sweep*. Using (5.4), this means in practice that for the $k$th mode we search for $b$ solutions in the subspace

$$\left\{ \sum_{j_1=1}^{r_1} \sum_{j_2=1}^{r_2} \ldots \sum_{j_{m-1}=1}^{r_{m-1}} \mathcal{G}_1(:, j_1) \otimes \mathcal{G}_2(j_1, :, j_2) \otimes \ldots \otimes \mathcal{G}_{k-1}(j_{k-2}, :, j_{k-1}) \otimes \mathbf{Y} \right.$$

$$\left. \otimes \mathcal{G}_{k+1}(j_k, :, j_{k+1}) \otimes \ldots \otimes \mathcal{G}_m(j_{m-1}, :) | \mathbf{Y} \in \mathbb{C}^{n_k \times r_{k-1} r_k} \right\}. \tag{5.5}$$

Let $\mathbf{X}_{\neq k}$ be an orthonormal basis for (5.5), then via the variational characterization of eigenvalues, this is equivalent to solving following optimization

problem

$$\min_{\mathbf{X}_k^X \in \mathbb{R}^{r_{k-1} n_k r_k \times b}} (\mathbf{X}_k^X)^T \mathbf{X}_{\neq k}^T \mathbf{A} \mathbf{X}_{\neq k} \mathbf{X}_k^X \text{ subject to } (\mathbf{X}_k^X)^T \mathbf{X}_k^X = \mathbf{I}. \tag{5.6}$$

We discuss now how we can construct a basis for (5.5). In order to achieve this, we need the concept of *interfaces*. In TT-notation, interfaces are defined as the matrices

$$\mathbf{X}^{>k}(:, (j_{k+1}, \ldots, j_m)) = \mathbf{G}_{k+1}(j_{k+1}) \ldots \mathbf{G}_m(j_m) \tag{5.7}$$

$$\mathbf{X}^{<k}((j_1, \ldots, j_{k-1}), :) = \mathbf{G}_1(j_1) \ldots \mathbf{G}_{k-1}(j_{k-1}). \tag{5.8}$$

These matrices are of size $r_k \times (n_{k+1} \ldots n_m)$ resp. $(n_1 \ldots n_{k-1}) \times r_{k-1}$. We observe that $\mathbf{X}^{<k}$ is completely defined by the first $k-1$ modes of the decomposition, while $\mathbf{X}^{>k}$ is completely defined by the last $m-k$ modes. When in the next step we need $\mathbf{X}^{<k\pm1}$ and $\mathbf{X}^{>k\pm1}$, we just update the cores. It can be seen that a basis $\mathbf{X}_{\neq k}$ for (5.5) is

$$\mathbf{X}_{\neq k} = \mathbf{X}^{<k} \otimes \mathbf{I}_{n_k} \otimes (\mathbf{X}^{>k})^T. \tag{5.9}$$

In the literature $\mathbf{X}_{\neq k}$ is called a *frame matrix*. We want a condition for the frame matrix to be orthonormal. We make an orthonormal basis by orthonormalizing the interfaces $\mathbf{X}^{<k}$ and $\mathbf{X}^{>k}$. We introduce *left*-and *right orthonormality*.

**Definition 4.** *[12, 32] A TT-core $\mathcal{G}_k$ is* left- *resp.* right orthonormal *if*

$$\sum_{j_k=1}^{n_k} \mathbf{G}_k(j_k)^T \mathbf{G}_k(j_k) = \mathbf{I}_{r_k}$$

*resp.*

$$\sum_{j_k=1}^{n_k} \mathbf{G}_k(j_k) \mathbf{G}_k(j_k)^T = \mathbf{I}_{r_{k-1}}.$$

Property 8 puts a condition on the interfaces such that the frame matrix is orthonormal.

**Property 8.** *[32, 62] If $\mathbf{G}_1, \ldots, \mathbf{G}_{k-1}$ are left-orthonormal and $\mathbf{G}_{k+1}, \ldots, \mathbf{G}_m$ are right-orthonormal, then $\mathbf{X}_{\neq k}$ is orthonormal.*

In the solvers for (5.6), we only need to know how to compute $\mathbf{X}_{\neq k}^T \mathbf{A} \mathbf{X}_{\neq k} \mathbf{v}$ for a vector $\mathbf{v}$. For this, we see the matrix $\mathbf{A}$ as a TT-operator. We refer to Section 2.2.2 for its definition and main properties. In Section 5 of [32] they made an algorithm with complexity $\mathcal{O}(r^3 n^2 (r^A)^2)$ where $r^A$ is the maximal internal rank of the TT-operator $\mathbf{A}$ and $r$ is the maximal rank of the block-TT tensor $\mathbf{X}_{\neq k}$.

Once we have solved (5.6), the new estimate of the eigenvectors is $\mathbf{X} = \mathbf{X}_{\neq k}\mathbf{X}_k^X$. We want to extract from this a basis $\mathbf{X}_{\neq k\pm 1}$ for the subspace used in the next step. To ease the notation we restrict ourselves to the case $m = 3$. Assume that we have just solved (5.6) for $k = 1$ and that we want to form $\mathbf{X}_{\neq 2}$.

We first reshape $\mathbf{X}_1^X$ to a matrix $\mathbf{X}_{\text{resh}}^X$ of size $n_1 \times r_1 b$, then we perform an SVD-decomposition $\mathbf{X}_{\text{resh}}^X = \mathbf{U}\mathbf{S}\mathbf{Z}^T$ and truncate it to a rank $r$ matrix. This implies that

$$\mathbf{X}_1^X(:, i_b) \approx \sum_{i=1}^{r} \mathbf{U}(:, i) \otimes \mathbf{S}(i, i)\mathbf{Z}((:, i_b), i) = \sum_{i=1}^{r} \mathbf{U}(:, i) \otimes \hat{\mathbf{Z}}((:, i_b), i). \quad (5.10)$$

Then we proceed with

$$\begin{aligned}
\mathbf{X}(:, i_b) &= \mathbf{X}_{\neq 1}\mathbf{X}_1^X(:, i_b) && (5.11)\\
&= (\mathbf{I}_{n_1} \otimes (\mathbf{X}^{>1})^T)\mathbf{X}_1^X(:, i_b)\\
&\approx \left( \sum_{j_2=1}^{r_2} \mathbf{I}_{n_1} \otimes \mathcal{G}_2(:, :, j_2)^T \otimes \mathcal{G}_3(j_2, :) \right) \sum_{i=1}^{r} \mathbf{U}(:, i) \otimes \hat{\mathbf{Z}}((:, i_b), i) && (5.12)\\
&= \sum_{i=1}^{r} \sum_{j_2=1}^{r_2} \mathbf{U}(:, i) \otimes \sum_{j_1=1}^{r_1} \mathcal{G}_2(j_1, :, j_2)\hat{\mathbf{Z}}((j_1, i_b), i) \otimes \mathcal{G}_3(j_2, :). && (5.13)
\end{aligned}$$

So by defining

$$\hat{\mathcal{G}}_1(:, j_1) = \mathbf{U}(:, j_1), \hat{\mathcal{G}}_2(j_1, :, i_b, j_2) = \sum_{i=1}^{r_1} \mathcal{G}_2(i, :, j_2)\hat{\mathbf{Z}}((i, i_b), j_1),$$

$$\hat{\mathcal{G}}_3(j_2, :) = \mathbf{G}_3(j_2, :),$$

we have found a block-TT-decomposition of $\mathbf{X}(:, i_b)$ with index 2. Furthermore, note that by the orthonormality of the SVD-decomposition, the core $\hat{\mathbf{G}}_1$ is left-orthonormal by construction, so $\mathbf{X}_{\neq 2}$ is via Property 8 an orthonormal basis by construction. In [12] the author posed the idea to add extra random vectors and to use in (5.12) and (5.13),

$$\tilde{\mathbf{U}} = [\mathbf{U}, \mathbf{U}_1] \in \mathbb{R}^{n_1 \times (r+p)}, \tilde{\mathbf{Z}} = [\hat{\mathbf{Z}}, 0] \in \mathbb{R}^{r_1 b \times (r+p)} \quad (5.14)$$

with $\mathbf{U}_1$ a random matrix such that the columns of $\tilde{\mathbf{U}}$ remain orthonormal against each other, instead of $\mathbf{U}$ and $\mathbf{Z}$. Adding extra randomness improves the robustness of the algorithm for finding the wanted eigenvalues but makes it also more computationally demanding. The algorithm for calculating $b$ eigenvalues of a symmetric matrix $\mathbf{A}$ in TT-format is stated in Algorithm 8.

**Algorithm 8** Algorithm from [12] to compute $b$ eigenpairs of a symmetric matrix $\mathbf{A}$ in TT-format.

**Input:** TT-decomposition of $\mathbf{A}$
**Output:** Eigenpairs of $\mathbf{A}$.

Make initial guess $\mathbf{X}$ in block-TT format.
Orthogonalise $\mathbf{X}_{\neq 1}$ by modifying each core using Property 8.
**while** Stopping criterion not fulfilled **do**
  **for** $k = 1, \dots, m-1$ (left-to-right half-sweep) **do**
    Solve (5.6) and store the solution in $\mathbf{X}^X$
    Reshape $\mathbf{X}^X$ to $r_{k-1} \times n_k \times b \times r_k$
    Retrieve $\mathbf{X}_{\neq k+1}$ analogously to (5.11)-(5.13)
  **end for**
  **for** $k = m, m-1, \dots, 2$ (right-to-left half-sweep) **do**
    Solve (5.6) and store the solution in $\mathbf{X}^X$
    Reshape $\mathbf{X}^X$ to $r_{k-1} \times n_k \times b \times r_k$
    Retrieve $\mathbf{X}_{\neq k-1}$ analogously to (5.11)-(5.13)
  **end for**
**end while**

We want to use the same ideas to calculate the eigenvalues of the generalized eigenvalue problem in (5.2). We first prove that the $\mathbf{\Delta}$-matrices can be interpreted as TT-operator. In contrast to a symmetric eigenvalue problem, the characterization as an optimization problem does no longer hold for the generalized eigenvalue problem

$$\left(\mathbf{X}_{\neq k}^T \mathbf{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \mathbf{\Delta}_0 \mathbf{X}_{\neq k}\right). \tag{5.15}$$

This makes the selection of $b$ eigenpairs of (5.15) more difficult as it is not clear which eigenvalue to take. As we design the algorithm for solving $m$EPs, we use the induced structure to make it as efficient as possible.

## 5.3 Tensor-algorithm for multiparameter eigenvalue problems

In this and the following section we describe our algorithm that is implemented for $m$EPs. This section is devoted to the more basic aspects of the algorithm, the next section is more focused on some more technical issues like the convergence criterion and the selection of $b$ eigenvalues of (5.15). We start this section by motivating why a method similar to the algorithm in previous section can be applied for solving a $m$EP. The algorithm is given in pseudo-code in Algorithm 9.

### 5.3.1 Δ-matrices as TT-operators

We have seen in the previous section that the complexity depends on the ranks of the involved matrix $\mathbf{A}$ written as TT-operator. In order to use the TT-decomposition in our application, the ranks of the $\boldsymbol{\Delta}$-matrices, expressed as TT-operators must be modest. Without loss of generality we focus on $\boldsymbol{\Delta}_0$. As described in (5.3), $\boldsymbol{\Delta}_0$ is a $m \times m$-operator determinant. This means that also each element of $\boldsymbol{\Delta}_0$ is in fact the determinant of a matrix

$$\boldsymbol{\Delta}_0(i_1,\dots,i_m,j_1,\dots,j_m) = \det\left(\begin{bmatrix} \mathbf{B}_{11}(i_1,j_1) & \mathbf{B}_{12}(i_1,j_1) & \dots & \mathbf{B}_{1m}(i_1,j_1) \\ \mathbf{B}_{21}(i_2,j_2) & \mathbf{B}_{22}(i_2,j_2) & \dots & \mathbf{B}_{2m}(i_2,j_2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{m1}(i_m,j_m) & \mathbf{B}_{m2}(i_m,j_m) & \dots & \mathbf{B}_{mm}(i_m,j_m) \end{bmatrix}\right),$$

so the cores $\mathbf{G}_k^{\Delta}(i_k,j_k)$ for the operator $\boldsymbol{\Delta}_0$ need to fulfil

$$\det\left(\begin{bmatrix} \mathbf{B}_{11}(i_1,j_1) & \mathbf{B}_{12}(i_1,j_1) & \dots & \mathbf{B}_{1m}(i_1,j_1) \\ \mathbf{B}_{21}(i_2,j_2) & \mathbf{B}_{22}(i_2,j_2) & \dots & \mathbf{B}_{2m}(i_2,j_2) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{m1}(i_m,j_m) & \mathbf{B}_{m2}(i_m,j_m) & \dots & \mathbf{B}_{mm}(i_m,j_m) \end{bmatrix}\right) = \mathbf{G}_1^{\Delta}(i_1,j_1)\dots\mathbf{G}_m^{\Delta}(i_m,j_m),$$

$$i_k, j_k = 1,\dots,n_k, k = 1,\dots,m.$$

This means we need to find a way to write the determinant as a product of matrices. This can be done using following theorem.

**Theorem 6.** *[34] Let $\mathbf{A} \in \mathbb{C}^{n\times n}$ and denote with $a_{i:}$ the $i$th row of $\mathbf{A}$, then*

$$\det(\mathbf{A}) = \mathbf{D}_{1,n}(a_{1:})\mathbf{D}_{2,n}(a_{2:})\dots\mathbf{D}_{n,n}(a_{n:}), \tag{5.16}$$

*where $\mathbf{D}_{i,n}, i = 1,\dots,n$ is calculated recursively using following expressions*

$$\mathbf{D}_{1,n}(\mathbf{a}) = [\mathbf{a}(1),\mathbf{a}(2),\dots,\mathbf{a}(n)] \in \mathbb{C}^{1\times n}$$

$$\mathbf{D}_{n,n}(\mathbf{a}) = [\mathbf{a}(n),-\mathbf{a}(n-1),\dots,(-1)^{n-1}\mathbf{a}(1)]^T \in \mathbb{C}^{n\times 1}$$

$$\mathbf{D}_{k,n}(\mathbf{a}) = \begin{bmatrix} \mathbf{D}_{k-1,n-1}(\mathbf{a}(2),\dots,\mathbf{a}(n)) & 0 \\ (-1)^{k-1}\mathbf{a}(1)\mathbf{I} & \mathbf{D}_{k,n-1}(\mathbf{a}(2),\dots,\mathbf{a}(n)) \end{bmatrix} \in \mathbb{C}^{\binom{n}{k-1}\times\binom{n}{k}},$$

$$k = 2,\dots,n-1.$$

**Corollary 1.** *A possible TT-decomposition of $\boldsymbol{\Delta}_0$ is given by*

$$\mathbf{G}_k^{\Delta}(i_k,j_k) = \mathbf{D}_{km}([\mathbf{B}_{k1}(i_k,j_k),\dots,\mathbf{B}_{km}(i_k,j_k)]), k = 1,\dots,m.$$

*The ranks $r_i^{\Delta}, i = 1,\dots,m$ are described by the $m+1$-st line of Pascal's triangle. Note that these are upper bounds for the internal rank.*

Corollary 1 gives rise to a recursive algorithm to compute a TT-decomposition of the $\boldsymbol{\Delta}$-matrices. It follows that the ranks are on average $\frac{2^m}{m}$ which becomes high if $m$ increases. In Section 5.5 we devote a numerical experiment to the influence of $m$ on the computational time. We see that a significant reduction in time can be achieved if we first round the TT-decomposition upto a certain low tolerance.

The requirement to keep the ranks of the TT-cores small is also the reason why we did not include deflation in Algorithm 9, even though the preservation of the TT-format can be easily achieved using Hotelling's deflation. If we have calculated $q$ eigenvalues, then we can deflate them towards infinity by defining

$$\boldsymbol{\Delta}_0^{\text{defl}} = \boldsymbol{\Delta}_0 - \sum_{i=1}^{q} \frac{1}{\mathbf{y}_i^T \boldsymbol{\Delta}_m \mathbf{x}_i} \boldsymbol{\Delta}_0 \mathbf{x}_i \mathbf{y}_i^T \boldsymbol{\Delta}_m, \tag{5.17}$$

where $\mathbf{y}_i$ is the left-eigenvector (which is also of rank one in TT-format). However it follows that $\boldsymbol{\Delta}_0^{\text{defl}}$ has internal ranks $r_i + q(r_i^{\Delta})^2$ which is too large for practical use.

Note that in the context of multiparameter eigenvalue problems, finding a deflation method that respects the TT-rank structure is still an open problem.

### 5.3.2 Size of the projected eigenvalue problem and the rank of $\mathbf{X}$

The subspaces $\mathbf{X}_{\neq k}$ are of dimension $r_{k-1} n_k r_k$ and so is the projected eigenvalue problem $\left(\mathbf{X}_{\neq k}^T \boldsymbol{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \boldsymbol{\Delta}_0 \mathbf{X}_{\neq k}\right)$. This means we can keep the eigenvalue problem small if the internal ranks of $\mathbf{X}_{\neq k}$ are low. These ranks are determined by the number of vectors taken in the SVD-decomposition when constructing $\mathbf{X}_{\neq k\pm 1}$ from $\mathbf{X}_{\neq k}$ and the computed eigenvectors of the projected eigenvalue problem (see (5.10)). We know that all eigenvectors of $(\boldsymbol{\Delta}_m, \boldsymbol{\Delta}_0)$ are of rank one. For estimating which rank is appropriate, we consider the case where the eigenvectors stored in $\mathbf{X}$ have converged. Note that when having a complex-valued eigenvector we split it into its real-and complex part as we want to keep the subspace real. A block-TT decomposition of $\mathbf{X}$ has internal ranks at most $b$. Let $\mathbf{X}$ consist of $p$ real eigenvectors $\mathbf{x}_1^i \otimes \mathbf{x}_2^i \otimes \ldots \otimes \mathbf{x}_m^i$ and $\frac{p-b}{2}$ complex-valued, then a possible block-TT decomposition of $\mathbf{X}$ of index one is

$$
\begin{aligned}
\mathcal{G}_1(:, i_b, i_b) &= \mathbf{x}_1^{i_b}, && i_b = 1, \ldots, p, \\
\mathcal{G}_2(i_b, :, i_b) &= \mathbf{x}_2^{i_b}, \\
\mathbf{G}_m(i_b, :) &= \mathbf{x}_m^{i_b}, \\
\mathcal{G}_1(i_b, :, i_b : i_b + 1) &= [\mathrm{Re}(\mathbf{x}_1^{i_b}), -\mathrm{Im}(\mathbf{x}_1^{i_b})], && i_b = p+1, p+3, \ldots, b-1, \\
\mathcal{G}_1(:, i_b + 1, i_b : i_b + 1) &= [\mathrm{Im}(\mathbf{x}_1^{i_b}), \mathrm{Re}(\mathbf{x}_1^{i_b})], \\
\mathcal{G}_j(i_b, :, i_b : i_b + 1) &= [\mathrm{Re}(\mathbf{x}_j^{i_b}), -\mathrm{Im}(\mathbf{x}_j^{i_b})], && j = 2, \ldots, m-1 \\
\mathcal{G}_j(i_b + 1, :, i_b : i_b + 1) &= [\mathrm{Im}(\mathbf{x}_j^{i_b}), \mathrm{Re}(\mathbf{x}_j^{i_b})], \\
\mathbf{G}_m(i_b : i_b + 1, :) &= [\mathrm{Re}(\mathbf{x}_m^{i_b}), \mathrm{Im}(\mathbf{x}_m^{i_b})],
\end{aligned}
$$

We consider $r$ to be maximal $b + 1$ as mostly not all eigenvalues are converged at the same time.

### 5.3.3 Stopping criterion

Usually after a few sweeps, we find already converged eigenvalues. These are eigenvalues with small modulus, but usually not among the smallest. To make our algorithm more robust in finding the smallest eigenvalues, we do extra full sweeps. We stop after a fixed number of sweeps or when we did not find new eigenvalues in the last sweeps. We noticed that the more clustered the eigenvalues are, the more sweeps it needs to find them all.

## 5.4 Selecting eigenvalues and convergence criterion

In the previous section we showed that it is possible to make an algorithm using the same ideas as in [12]. There are two main aspects that are completely different: the selection of eigenvalues of the projected problem and the convergence criterion. To achieve this we rely on the fact that eigenvectors of $(\boldsymbol{\Delta}_m, \boldsymbol{\Delta}_0)$ are always of rank one in their TT-representation.

**Algorithm 9** Algorithm to calculate eigenvalue-tuples with smallest $\mu_m$ in absolute value of a $m$EP. Line 7 is further discussed in Section 5.4.

**Input:** $m$EP

**Output:** Eigenvalue-tuples $(\mu_1, \mu_2, \ldots, \mu_m)$ with corresponding eigenvector-tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ which are smallest in absolute value.

1: Construct $\mathbf{\Delta}_m$ and $\mathbf{\Delta}_0$
2: Make initial guess $\mathbf{X}$ in block-TT format
3: Orthogonalise $\mathbf{X}_{\neq 1}$ by modifying each core using Property 8
4: **while** Stopping criterion not fulfilled **do**
5: **for** $k = 1, \ldots, m-1$ (left-to-right half-sweep) **do**
6: Update $\mathbf{X}_{\neq k}$ using (5.11)-(5.13) (except in the very first iteration)
7: Select $b$ eigenpairs of $\left(\mathbf{X}_{\neq k}^T \mathbf{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \mathbf{\Delta}_0 \mathbf{X}_{\neq k}\right)$ and check their convergence (see Section 5.4)
8: Store the associated eigenvectors in $\mathbf{X}_k^X$ and make real
9: Reshape $\mathbf{X}_k^X$ to $r_{k-1} \times n_k \times b \times r_k$
10: **end for**
11: Update cores in the right-to-left half-sweep (analogous to left-to-right half-sweep)
12: **end while**

### 5.4.1 Selection of eigenvalues

For standard eigenvalue problems with symmetric matrices, calculating the $b$ smallest eigenvalues of $\mathbf{A}$ corresponds to minimizing a cost function inferred from the variational characterization of eigenvalues. Projection methods work well [60, 66] for symmetric eigenvalue problems as the eigenvalues of the projected eigenvalue problem are bounded by the eigenvalues of the large problem [52]. For our non-Hermitian generalised eigenvalue problem

$$\left(\mathbf{X}_{\neq k}^T \mathbf{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \mathbf{\Delta}_0 \mathbf{X}_{\neq k}\right) \tag{5.18}$$

the convergence is much harder to understand. In particular, it is not always clear which Ritz pairs should be selected.

The higher we take $b$, the more eigenvalues we find. Moreover we need to take $b$ significantly high such that if we select an eigenvalue that is not a good estimate of an eigenvalue, it hardly harms the convergence. On the other hand, in Section 5.3.2 we saw that the maximal rank $r$ is chosen as $b+1$, so as the size of (5.18) is $r^2 n$, we cannot take $b$ too large. Via numerical experiments, we concluded that $b$ at least 5 gives a good balance between the two. It mainly depends on the size of the involved matrices which value of $b$ is appropriate. The lower $n$ is, the higher $b$ may be.

The following method for selecting $b$ eigenvalues, is heuristic and uses the same idea as the method in Section 4.6.1.

1. Check if the Ritz value is converged (see Section 5.4.2)
2. If it has not converged, we check if the Ritz vector is close enough to one of the Ritz vectors in the previous iteration, see around (5.19).
3. If there are too many Ritz values that satisfy 2., we select among them the Ritz values with smallest residual norm.
4. If there are not enough Ritz values that fulfil 2., we complete the selection with the Ritz values with the smallest residual norms. If there are still not enough, then we add random vectors.

We do not lock converged Ritz vectors in our subspace nor deflate them as they do not necessarily correspond with the smallest eigenvalues (see Section 5.3.3). Let $q$ be the number of already found eigenvalues, then we select the $2b + q$ Ritz values via the `eigs` (ARPACK) command in Matlab that have smallest modulus and positive real part. Here, one of the two projection matrices need to be formed explicitly which has a cost of $\mathcal{O}(r^4 n^2 (r^\Delta)^2)$, while for the other projection matrix only the matrix-vector product is performed with cost $\mathcal{O}(r^3 n^2 (r^\Delta)^2)$.

There are two problems with this procedure:

1. Within a left-to-right sweep we project the eigenvalue problem on $\mathbf{X}_{\neq k}$ in step $k$ and in step $k+1$ we project on $\mathbf{X}_{\neq k+1}$. The eigenvectors of the projected eigenvalue problem in step $k$ are not comparable to the eigenvectors of the projected eigenvalue problem in step $k+1$. (an analogous problem arises in right-to-left sweeps)
2. Calculating the norm of residuals in every step for all $2b + q$ eigenvalues is expensive.

Without loss of generality we consider the case that we update the $k$th mode and that we update the $(k+1)$-th mode in the next step. We can solve both problems at the same time. Also for checking the convergence we use the same ideas. Given a Ritz pair $\left(\mu^X, \mathbf{X}_{\neq k}\mathbf{x}_k^x\right)$ of $\left(\mathbf{X}_{\neq k}^T \boldsymbol{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \boldsymbol{\Delta}_0 \mathbf{X}_{\neq k}\right)$ that is selected in iteration $k$, we make a frame matrix $\hat{\mathbf{X}}_{\neq k+1}$ based on only this eigenpair in the same way as in (5.11)-(5.13). Our estimate $\hat{\mathbf{x}}_{k+1}^x$ is then the vectorisation of $\hat{\mathcal{G}}_{k+1}$. In the next iterate, we compare an eigenvector $\mathbf{x}_{k+1}^x$

with $\hat{\mathbf{x}}_{k+1}^x$ by seeing $\mathbf{x}_{k+1}^x$ resp. $\hat{\mathbf{x}}_{k+1}^x$ as a tensor of size $r_k \times n_{k+1} \times r_{k+1}$ resp. $\hat{r}_k \times n_{k+1} \times r_{k+1}$, make a rank-one TT-approximation

$$\mathbf{x}_{k+1}^x \approx \mathbf{x}^{<k+1} \otimes \overline{\mathbf{x}}_{k+1} \otimes \mathbf{x}^{>k+1}$$

resp.

$$\hat{\mathbf{x}}_{k+1}^x \approx \hat{\mathbf{x}}^{<k+1} \otimes \hat{\overline{\mathbf{x}}}_{k+1} \otimes \hat{\mathbf{x}}^{>k+1} \tag{5.19}$$

of these tensors and we consider them equal if

$$\cos(\theta(\overline{\mathbf{x}}_{k+1}, \hat{\overline{\mathbf{x}}}_{k+1})) > 0.99. \tag{5.20}$$

In words, we check whether the angle between the second mode (i.e. the mode that contains the approximation of one of the vectors in an eigenvector-tuple) of $\hat{\mathbf{x}}_{k+1}^x$ and $\mathbf{x}_{k+1}^x$ is low enough. Certainly if convergence starts, from the following Property 9 we deduce that both vectors can be well approximated by a rank-one tensor.

**Property 9.** *Assume that an eigenvector* $\mathbf{x} = \mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \ldots \otimes \mathbf{x}_m$ *of* $(\boldsymbol{\Delta}_m, \boldsymbol{\Delta}_0)$ *is such that it lays in the subspace spanned by* $\mathbf{X}_{\neq k}$*. This means that there is a* $\mathbf{x}_k^x$ *such that*

$$\mathbf{X}_{\neq k}\mathbf{x}_k^x = \mathbf{x}.$$

*It holds for all* $p = 1, \ldots, m$ *that there is a frame matrix* $\mathbf{X}_{\neq p}$ *such that*

$$\mathbf{x}_p^x = \mathbf{X}_{\neq p}^T \mathbf{x} = \mathbf{x}^{<p} \otimes \mathbf{x}_p \otimes \mathbf{x}^{>p}.$$

*The vector* $\mathbf{x}_p$ *is the pth element of an eigenvector-tuple of the mEP.*

*Proof.* We have seen that if a vector $\mathbf{x}$ lays in the subspace spanned by the columns of $\mathbf{X}_{\neq k}$, we can build a frame matrix for mode $k+1$ that also contains this $\mathbf{x}$. Therefore, it is enough to prove the result for mode $k$. The latter follows from

$$
\begin{aligned}
\mathbf{x}_k^x &= \mathbf{X}_{\neq k}^T \left(\mathbf{x}_1 \otimes \mathbf{x}_2 \otimes \ldots \otimes \mathbf{x}_m\right) \\
&= \left((\mathbf{X}^{<k})^T \otimes \mathbf{I}_k \otimes (\mathbf{X}^{>k})\right) \left(\mathbf{x}_1 \otimes \ldots \otimes \mathbf{x}_m\right) \\
&= \sum_{j_1=1}^{r_1} \sum_{j_2=1}^{r_2} \cdots \sum_{j_{k-2}=1}^{r_{k-2}} \sum_{j_{k+2}=1}^{r_{k+2}} \cdots \sum_{j_{m-1}=1}^{r_{m-1}} \mathbf{G}_1(:, j_1)^T \mathbf{x}_1 \otimes \mathcal{G}_2(j_1, :, j_2)^T \mathbf{x}_2 \otimes \ldots \otimes \\
&\quad \mathcal{G}_{k-2}(j_{k-3}\, :, j_{k-2})^T \mathbf{x}_{k-2} \otimes \mathcal{G}_{k-1}(j_{k-2}, :, :)^T \mathbf{x}_{k-1} \otimes \mathbf{x}_k \otimes \\
&\quad \Big(\mathcal{G}_{k+1}(:, :, j_{k+1})^T \mathbf{x}_{k+1} \otimes \mathcal{G}_{k+2}(j_{k+1}, :, j_{k+2})^T \mathbf{x}_{k+2} \otimes \ldots \otimes \\
&\quad \mathcal{G}_{m-1}(j_{m-2}, :, j_{m-1})^T \mathbf{x}_{m-1} \otimes \mathbf{G}_m(j_{m-1}, :)^T \mathbf{x}_m \Big) \\
&= \underbrace{\sum_{j_{k-2}=1}^{r_{k-2}} a_{j_{k-2}}^{k-2} \mathcal{G}_{k-1}(j_{k-2}, :, :)^T \mathbf{x}_{k-1}}_{:=\mathbf{x}^{<k}} \otimes \mathbf{x}_k \otimes \underbrace{\sum_{j_{k+1}=1}^{r_{k+1}} a_{j_{k+1}}^{k+1} \mathcal{G}_{k+1}(:, :, j_{k+1}) \mathbf{x}_{k+1}}_{:=\mathbf{x}^{>k}},
\end{aligned}
$$

where the constants $a_{j_{k-2}}^{k-2}$ and $a_{j_{k+1}}^{k+1}$ are obtained from the projections of $\mathcal{G}_{j_i}(j_{i-1}, :, j_i)$ on $\mathbf{x}_i$ for $i = 1, \ldots, k-2$ resp. $i = k+1, \ldots, m$. □

To make an estimation of the residual norm for eigenpair $(\mu^X, \mathbf{x}_k^x)$ of the projected problem, we use the same frame matrix $\hat{\mathbf{X}}_{\neq k+1}$ and we compute the norm of the residual projected on the subspace spanned by the columns of $\hat{\mathbf{X}}_{\neq k+1}$, i.e

$$
\|\hat{\mathbf{r}}\| = \left\| \hat{\mathbf{X}}_{\neq k+1}^T \boldsymbol{\Delta}_m \hat{\mathbf{X}}_{\neq k+1} \hat{\mathbf{x}}_{k+1}^x - \mu_i^X \hat{\mathbf{X}}_{\neq k+1}^T \boldsymbol{\Delta}_0 \hat{\mathbf{X}}_{\neq k+1} \hat{\mathbf{x}}_{k+1}^x \right\|, \tag{5.21}
$$

where $\hat{\mathbf{X}}_{\neq k+1}^T \boldsymbol{\Delta}_0 \hat{\mathbf{X}}_{\neq k+1} \hat{\mathbf{x}}_{k+1}^x$ and $\hat{\mathbf{X}}_{\neq k+1}^T \boldsymbol{\Delta}_m \hat{\mathbf{X}}_{\neq k+1} \hat{\mathbf{x}}_{k+1}^x$ are directly computed using the TT-representations. This is a much cheaper alternative than explicitly calculating the residual norm. As (5.21) is always smaller than the residual norm of the full size problem, there is no guarantee that if (5.21) is small it also holds for the residual norm. But if this is already high, we know that the residual norm is even higher and it is for sure not a converged eigenvalue. The calculation of (5.21) is used to decide which eigenpairs to select if the criterion in (5.20) is not restrictive enough and to decide whether it is a possible converged eigenpair, see Section 5.4.2. As we need to do this in every step of a sweep for all $2b + q$ eigenpairs, it would be too expensive to calculate the residual norm for each eigenpair.

The algorithm is given in pseudo-code in Algorithm 10.

### 5.4.2 Convergence criterion

We use the same convergence criterion as used in [30] for $m = 3$,

$$\left\| \begin{bmatrix} (\mathbf{A}_1 - \mu_1\mathbf{B}_{11} - \ldots - \mu_m\mathbf{B}_{1m})\mathbf{x}_1 \\ (\mathbf{A}_2 - \mu_1\mathbf{B}_{21} - \ldots - \mu_m\mathbf{B}_{2m})\mathbf{x}_2 \\ \vdots \\ (\mathbf{A}_m - \mu_1\mathbf{B}_{m1} - \ldots - \mu_m\mathbf{B}_{mm})\mathbf{x}_m \end{bmatrix} \right\|_\infty < \epsilon \tag{5.22}$$

so an eigenvalue is considered converged if the residual norm for all equations together is below a tolerance $\epsilon$. We choose $\epsilon$ equal to $10^{-6}$ as this is the tolerance that was used in the aforementioned chapter. s

Without loss of generality, we only consider the case where we just have updated the $k$th mode and will update the $k+1$st mode in the next step. In Section 5.4.1, we explained how we estimate the residual norm for $\mathbf{x}_k^x$ by making a frame matrix $\hat{\mathbf{X}}_{\neq k+1}$ and to compute the residual projected on the subspace spanned by the columns of this frame matrix (see (5.21)). The idea is now that if this residual norm is below $\epsilon_1$, to continue this process and to construct the frame matrix $\hat{\mathbf{X}}_{\neq k+2}$, estimate $\hat{\mathbf{x}}_{k+2} \in \mathbb{C}^{n_{k+2}}$ and check whether the residual projected on the subspace spanned by the columns of this frame matrix is below $\epsilon_1$ too. We can continue this process until all modes are visited or till the moment we find a mode for which this approximation of the residual norm is too high. If we reach $m$ and not all modes are visited, then we start again with $k$ and go back in reverse order to mode one. As we always check the residual whithin a subspace, $\epsilon_1$ is chosen low. We use in our algorithm $\epsilon_1 = 10^{-8}$. If the residual projected on all subspaces $\hat{\mathbf{X}}_{\neq 1}, \hat{\mathbf{X}}_{\neq 2}, \ldots, \hat{\mathbf{X}}_{\neq m}$ is below $\epsilon_1$, we may have found a converged eigenvalue. In this case we have an estimate for the vectors $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_m$ with eigenvalue $\hat{\mu}_m$. In order to check the criterion (5.22), we need associated values $\hat{\mu}_1, \ldots, \hat{\mu}_{m-1}$ to complete the eigenvalue-tuple. In [30, 53] a Tensor Rayleigh Quotient iteration (TRQI) was described to refine $\hat{\mu}_m$ and to give associated values $\hat{\mu}_1, \ldots, \hat{\mu}_{m-1}$.

If we have found a new eigenvalue, it is still possible that it is an eigenvalue we already found. In [30] a method was described to check this for 3EP. We use an analogous tool for the general $m$EP case. A new eigenvector $\hat{\mathbf{x}} = \hat{\mathbf{x}}_1 \otimes \hat{\mathbf{x}}_2 \otimes \ldots \otimes \hat{\mathbf{x}}_m$ is accepted if

$$\max_{p=1\ldots,q} \frac{(\mathbf{y}^p)^* \boldsymbol{\Delta}_0 \hat{\mathbf{x}}}{(\mathbf{y}^p)^* \boldsymbol{\Delta}_0 \mathbf{x}^p} < \xi \tag{5.23}$$

with $\mathbf{x}^p = \mathbf{x}_1^p \otimes \mathbf{x}_2^p \otimes \ldots \otimes \mathbf{x}_m^p$ resp. $\mathbf{y}^p = \mathbf{y}_1^p \otimes \mathbf{y}_2^p \otimes \ldots \otimes \mathbf{y}_m^p$ already found right- and left eigenvectors. For calculating the left eigenvector-tuple $(\mathbf{y}_1^p, \ldots, \mathbf{y}_m^p)$ we use the same tool as used in [30]. The tolerance $\xi$ is usually chosen as $10^{-4}$.

Note that the expression in (5.23) can be calculated by considering $\mathbf{x}, \mathbf{y}$ and $\mathbf{\Delta}_0$ in TT-format. Remark that in theory $(\mathbf{y}^p)^* \mathbf{\Delta}_0 \hat{\mathbf{x}} = 0$ if $\hat{\mathbf{x}}$ and $\mathbf{y}^p$ are associated to a different eigenvalue. Note that this is a different criterion than in (5.20). In (5.20) we only have estimations of a solution of the $k+1$st equation in the $m$EP (5.1). As this is not the solution of a generalized or standard eigenvalue problem, it is not possible to use a similar criterion as in (5.23).

The implemented algorithm based on the ingredients described above, can be found in Algorithm 11. This algorithm is the algorithm to which we refer in Algorithm 10 line 4.

## 5.5 Numerical experiments

In this section we test our algorithm on some of the examples discussed in [30] to prove the algorithm works. The second goal of this section is to see how the computational time depends on $m$ and $n$ (the size of the involved matrices). The code for Algorithm 8 which was the basis for our algorithm, is the code `dmrg_eig` available in the TT-Toolbox [13]. We choose $b = 5$ and we add in every step of a sweep one extra random vector as explained in (5.14).

### 5.5.1 Performance

**Example 12.** *Consider Example 4 in Section 1.4.1, used in the recent paper [30] on multiparameter eigenvalue problems. We take as radii of the ellipsoid, the values* $x_0 = 1, y_0 = 1.5, z_0 = 2$. *For the configuration of the solution in* (1.18) *we take* $\rho = \sigma = \tau = 0$. *For the discretisation, we use* 200 *Chebyshev nodes as collocation points. We search for the eigenvalues such that* $\mu_3$ *is closest to* 0 *and* $\mu_3$ *closest to* 200. *If the searched eigenvalues are exterior, usually, it is sufficient to do only* 20 *full sweeps. For computing interior eigenvalues, we do* 100 *full sweeps. In Figure 5.1 we show the results. In the first case we found almost the* 15 *eigenvalue-tuples with smallest* $\mu_3$. *In the second case, we found almost half of the first* 20 *eigenvalues. We note that the algorithm in [30, Section 5.1] can find more eigenvalues for an equal amount of time, but the method [30] was especially designed for 3EP and the extension to* $m > 3$ *is not feasible as was discussed in the introduction (Section 5.1).*

In the literature there are, to the best of our knowledge, no examples for $m > 3$ and reasonable size of the matrices. Therefore, we generate some random examples to test our algorithm. To be able to test how many of the searched eigenvalues we have found, we need to be able to compute all eigenvalue-tuples.

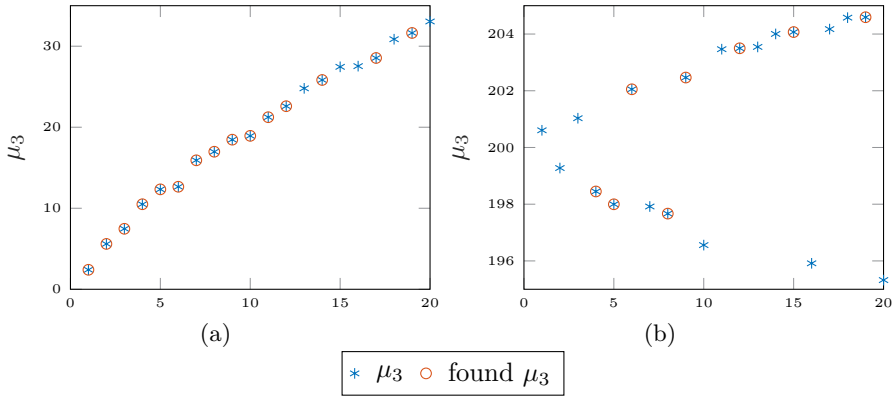



Figure 5.1: We plot the wanted eigenvalues together with the eigenvalues that we found for Example 12. In (a) the eigenvalues are well separated and we can find the first 10 eigenvalues. In (b) the eigenvalues are interior and we see the algorithm has much more difficulties in finding the wanted eigenvalues. We note that different runs can give diverent converged eigenvalues as the algorithm is subject to random numbers.

The technique described in the next lines is also the way random examples were created in [30].

We make first real vectors $\mathbf{a}_i, \mathbf{b}_{ij}, i, j = 1, \ldots, m$, and random linear independent $n \times n$ matrices $\mathbf{U}_i, \mathbf{Z}_i, i = 1, \ldots, m$, and construct

$$\mathbf{A}_i = \mathbf{U}_i \text{diag}(\mathbf{a}_i)\mathbf{Z}_i$$

$$\mathbf{B}_{ij} = \mathbf{U}_i \text{diag}(\mathbf{b}_{ij})\mathbf{Z}_i.$$

An eigenvalue-tuple $\mu = (\mu_1, \ldots, \mu_m)$ of the associated $m$EP is then found as the solution of the linear systems

$$\begin{bmatrix} \mathbf{b}_{11}(i_1) & \mathbf{b}_{12}(i_1) & \ldots & \mathbf{b}_{1m}(i_1) \\ \mathbf{b}_{21}(i_2) & \mathbf{b}_{22}(i_2) & \ldots & \mathbf{b}_{2m}(i_2) \\ \vdots & & \ddots & \vdots \\ \mathbf{b}_{m1}(i_m) & \mathbf{b}_{m2}(i_m) & \ldots & \mathbf{b}_{mm}(i_m) \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_m \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1(i_1) \\ \mathbf{a}_2(i_2) \\ \vdots \\ \mathbf{a}_m(i_m) \end{bmatrix}, i_1, \ldots, i_m = 1, \ldots, n. \tag{5.24}$$

This means that if we want to be sure that we compute the eigenvalue-tuples with smallest $|\mu_m|$ we need to solve $n^m$ linear systems.

Here we choose to randomly generate $\mathbf{b}_{i2}$ and to define $\mathbf{b}_{ij} = \mathbf{b}_{i2}^{j-1}$. The following code is used to construct $\mathbf{a}_i, \mathbf{b}_{i2}, \mathbf{U}_i$ and $\mathbf{Z}_i, i = 1, \ldots, m$. We give here the Matlab-code instead of giving it in pseudo-code, to distinguish it from code contributing to our proposed algorithm.

```
U_c = cell(m,1); V_c = cell(m,1);
B_c = cell(m, m); A_c = cell(m, 1);
for i = 1:m
    U_c{i} = 0.3*rand(n,n)+eye(n);
    V_c{i} = 0.3*rand(n,n)+eye(n);
end
[x, ~] = chebdif_mp( n, 2, 'double');
limit = linspace( -1.9, 2, 2*m+1);
limit = limit(1:2*m);
a_m = -5*randn(n,m); b_m = zeros(n,m);
for i = 1:m
    b_m(:,i) = (x/2*(limit(2*i)-limit(2*i-1))+ ...
                    1/2*(limit(2*i-1)+limit(2*i)));
end
for i = 1:m
    A_c{i} = V_c{i}*diag(a_m(:,i))*U_c{i};
    for j = 1:m
        B_c{i,j+1} = V_c{i}*diag( b_m(:,i).^(j-1))*U_c{i};
    end
end
```

where `chebdiff_mp` can be found in the package `MultiParEig` [54]. This code is inspired on the construction of the matrices in Example 12. We can shift the $m$EP such that all $\mu_m$ are positive-valued by defining a new $m$EP with matrices

$$\hat{\mathbf{A}}_i := \mathbf{A}_i + \eta \mathbf{B}_{im}, \quad \hat{\mathbf{B}}_{ij} := \mathbf{B}_{ij}, \quad i,j = 1,\ldots,m, \tag{5.25}$$

for a certain shift $\eta \in \mathbb{R}$. It can be seen that this means in the new $m$EP only $\mu_m$ is shifted to $\mu_m + \eta$.

**Example 13.** *(Randomly generated 4EP and 5EP) We generate a random 4EP and 5EP with $n = 100$. We shift it such that all $\mu_m$ are positive valued via* (5.25)*. We perform two experiments, on the one hand we compute $\mu_m$'s that are closest to* 0 *and on the other hand we compute $\mu_m$'s that are closest to* 5*. In the former we perform 20 full sweeps and in the latter we perform 100 full sweeps. We see in Figure 5.2 (a) resp. (c) the* 20 *smallest $|\mu_m|$ of the 4EP resp. 5EP and we plot the $\mu_m$'s that our algorithm found that were among these* 20 *eigenvalues. In Figure 5.2 (b) resp. (d) the* 20 *$\mu_m$ closest to* 5 *were plotted for the 4EP resp. 5EP together with $\mu_m$'s that we found in our algorithm. We see that if the eigenvalues are exterior we can find almost all of the wanted values, if the eigenvalues are interior and more close to each other we find only a few.*

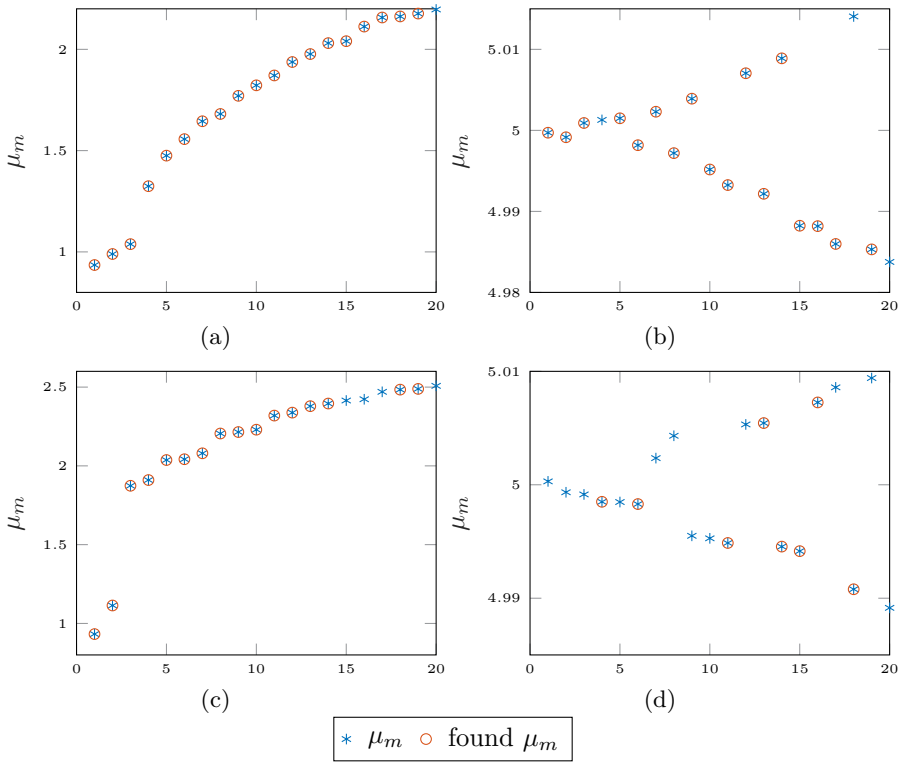



Figure 5.2: Figures (a)-(b) resp. (c)-(d) the results for the random $m$EP in Example 5.2. We plot the wanted eigenvalue $\mu_m$ in blue and the found eigenvalues in red. We see that if the wanted eigenvalues are exterior we can find them. In case they are interior or/and more close to each other, we see that it is far more difficult for the algorithm to find the wanted eigenvalues. Even after 100 sweeps we can only find a few.

## 5.6 Dependency on parameters

In this section, we discuss how the computational time depends on $m$ and $n$ (size of the involved matrices). We divide the algorithm in four parts from which we do separate timings: explicit construction of $\mathbf{X}_{\neq k}^T \mathbf{\Delta}_0 \mathbf{X}_{\neq k}$, calculating eigenvalues of $\left(\mathbf{X}_{\neq k}^T \mathbf{\Delta}_m \mathbf{X}_{\neq k}, \mathbf{X}_{\neq k}^T \mathbf{\Delta}_m \mathbf{X}_{\neq k}\right)$, selecting and checking convergence (Algorithm 10) and update of the modes of $\mathbf{X}$ (see line 9 and 6 of Algorithm 9). We note that each timing represents the total time spend on that part of the algorithm.

**Example 14.** *(Increasing $m$) We fix $n = 10$, make random mEP's for $m = 3, 4, \ldots, 11$ and we shift each mEP via* (5.25) *such that the searched eigenvalues are exterior. We time the different parts as discussed above when performing* 20 *full sweeps and we take the average over ten separate runs. The internal ranks of the $\boldsymbol{\Delta}$-matrices are the $m+1$st row of Pascal's*[1] *triangle but we know that the actual ranks are maximal $n^2$. So for $m > 8$, the maximal internal rank can be reduced. To see what happens in the worst case (no reduction possible for the ranks), we do first an experiment without reducing the internal ranks of the $\boldsymbol{\Delta}$-matrices. We perform the same experiment but now we reduce the internal rank by rounding the $\boldsymbol{\Delta}$-matrices as TT-operator to a tolerance of $10^{-13}$. In Figure 5.3 we see the timings for the different parts when we round the $\boldsymbol{\Delta}$-matrices.*

We see in Figure 5.3 (a)-(c) exponential dependency on $m$, when the $\boldsymbol{\Delta}$-matrices are not rounded. As well for the matrix-vector product $\boldsymbol{\Delta}_j\mathbf{v}$ or $\mathbf{X}_{\neq k}^T\boldsymbol{\Delta}_j\mathbf{X}_{\neq k}\mathbf{v}$ as well for the calculation of a projection $\mathbf{X}_{\neq k}^T\boldsymbol{\Delta}_0\mathbf{X}_{\neq k}$, the number of operations depend on $m$ as $m(r^{\Delta})^2$. On average $r^{\Delta}$ is $\frac{2^m}{m}$ as the ranks are the m+1st line in Pascal's triangle. This means that if $m$ is increased by one, we need to perform almost four times as much work. When we round the $\boldsymbol{\Delta}$-matrices first, we see that the computational time is reduced by a factor ten for $m = 11$. In the update of the modes of $\mathbf{X}$, the number of operations does only depend linearly on $m$ as no matrix-vector operations with one of the $\boldsymbol{\Delta}$-matrices is performed here. This explains Figure 5.3 (d) where we see that the rounding of the $\boldsymbol{\Delta}$-matrices does not affect the computational time for the update of the modes of $\mathbf{X}$.

**Example 15.** *(Increasing $n$) We fix $m = 4$, make random mEP's for $n = 10, 30, \ldots, 190$ and we shift it such that the searched eigenvalues are exterior using* (5.25)*. We time the different parts in the same way as in Example 14 when performing* 20 *sweeps and we take the average over ten separate runs. The results are plotted in Figure 5.4. The computational time needed for the selection and the convergence, depend on how many eigenvalues are converged and at which sweep it converges, explaining the decrease in the computational time in Figure 5.4 (c). In all other parts we see a quadratic increase explained by the fact that there are parts for which the complexity depends quadratically on $n$. We see that the solving of the projected eigenvalue problems takes most of the time.*

[1]Blaise Pascal (1623-1662) [78] was a French mathematician, inventor, physicist, philosopher, writer and Catholic theologician. There is a university and a programming language named after him.

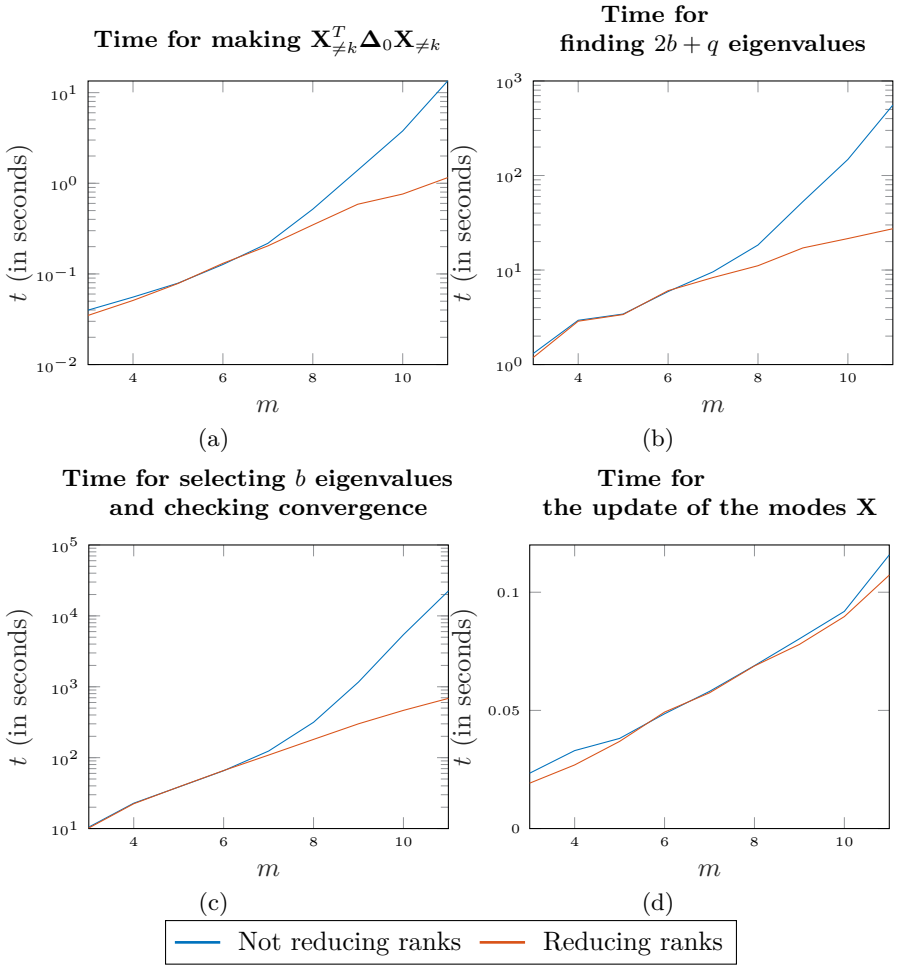



Figure 5.3: Figures corresponding to Example 5.4. We plot the time needed to perform the different parts of the algorithm when we increase the size $n$ of the matrices. We see that solving of the projected eigenvalue problem is computationally the most demanding. The computational time needed for the selection and the convergence, depends on how many eigenvalues are converged and at which sweep it converges, explaining the decrease in the computational time in (c).

## 5.7 Conclusion

In this chapter we presented a method that makes use of tensors to calculate a selection of the eigenvalues of a $m$EP with $m > 3$ and $n$ reasonably large. We showed that the Tensor-Train format can efficiently represent the $m$EP and showed that the associated $\mathbf{\Delta}$-matrices can be seen as Tensor-Train operators. This allows us to modify the algorithm in [12] to solve $m$EPs. Making modifications to the selection- and convergence criterion is necessary as we need to compute the eigenvalues of a non-symmetric generalised eigenvalue problem. We describe our modifications and motivate why we made them.

We illustrate with numerical experiments that our algorithm is able to find eigenvalues near zero and if they are well separated, we find all wanted eigenvalues.

**Algorithm 10** Algorithm to select $b$ eigenvalues among the $2b+q$ eigenvalues with positive imaginary part that we have calculated from (5.18). This algorithm is used on line 7 of Algorithm 9. The method for line 4 is explained in Section 5.4.2.

**Input:**

1. Frame matrix $\mathbf{X}_{\neq k}, k = 1, \ldots, m$ and direction $d = 1$ or $-1$ (left-to-right half sweep or right-to-left half sweep)
2. $\mathbf{\Delta}$-matrices $\mathbf{\Delta}_m, \mathbf{\Delta}_0$
3. $2b+q$ eigenpairs of (5.15), with $q$ the number of already found eigenvalues
4. The list $\hat{\mathbf{X}}^k_{\text{selected}}$ with estimates in (5.19), selected in the previous iterate.

**Output:**

1. Selection of $b$ eigenpairs of (5.15). This means we store its eigenvectors in the columns of the matrix $\mathbf{X}^X_k$, make a rank-one decomposition of $\hat{\mathbf{x}}^x_{k+d}$ and add $\hat{\overline{\mathbf{x}}}_{k+d}$ to $\hat{\mathbf{X}}^{k+d}_{\text{select}}$ (see (5.19)) .

1: **for** $i = 1$ to $2b+q$ **do**
2: Select the $i$th eigenpair $(\mu^X, \mathbf{x}^x_k)$ of (5.15)
3: Make subspace $\hat{\mathbf{X}}_{\neq k+d}$ and vector $\hat{\mathbf{x}}^x_{k+d}$ (See close to (5.19))
4: Estimate the residual norm (5.21) and check the convergence of $\mathbf{x}^x_k$ (see Section 5.4.2)
5: Make a rank-one decomposition (5.19) of $\mathbf{x}^x_k$ and retrieve $\overline{\mathbf{x}}_k \in \mathbb{C}^{n_k}$
6: **if** $\max_{\mathbf{x} \in \hat{\mathbf{X}}^k_{\text{select}}} (\cos(\theta(\mathbf{x}, \overline{\mathbf{x}}_k)) > 0.99$ (check if vectors are close to each other) and not yet converged **then**
7: Select this eigenpair
8: **end if**
9: **end for**
10: **if** we select not enough eigenvectors **then**
11: Select eigenvectors that were not selected neither converged with the smallest residual
12: **end if**
13: **if** we select too much eigenvectors **then**
14: Select among the vectors that we selected the eigenvectors with the smallest residual
15: **end if**

**Algorithm 11** Algorithm to check if a Ritz value has converged. This algorithm is the algorithm to which we refer in Algorithm 10 line 4. In the experiments we choose to keep the $p$ found eigenvalues that best fulfill our criterion. We choose in the numerical experiments $p = 4b$.

**Input:**

1. Direction $d = 1$ (left-to-right sweep), $d = -1$ (right-to-left sweep)
2. Subspace $\hat{\mathbf{X}}_{\neq k+d}$
3. Vector $\hat{\mathbf{x}}^x_{k+d}$ (see line 4 in Algorithm 10)

**Output:** Estimate of residual $\hat{\mathbf{r}}$ and check if eigenvalue has converged.

1: Compute $\|\hat{\mathbf{r}}\|$ as in (5.21)
2: Decompose $\hat{\mathbf{x}}^x_{k+d}$ and compute estimate $\hat{\mathbf{x}}_{k+d} \in \mathbb{C}^{n_{k_1}}$
3: $k_1 = k + d$;
4: **while** $\|\hat{\mathbf{r}}\| < \epsilon_1$ and we do not have estimates for all modes **do**
5: **if** $k_1 == m$ or $k_1 == 1$ **then** (If we reach the first/last mode, we need to change direction)
6: $k_1 = k$; $d = -d$
7: **end if**
8: $k_1 = k_1 + d$
9: Make $\hat{\mathbf{X}}_{\neq k_1}$ and vector $\hat{\mathbf{x}}_{k_1}$
10: Decompose $\hat{\mathbf{x}}_{k_1}$ and compute estimate $\hat{\mathbf{x}}_{k_1} \in \mathbb{C}^{n_{k_1}}$
11: Compute $\|\hat{\mathbf{r}}\|$ as in (5.21)
12: **end while**
13: **if** All $\|\hat{\mathbf{r}}\| < \epsilon_1$ **then**
14: Check if $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_m$ is an eigenvector-tuple that is not found yet via (5.23)
15: **if** Condition (5.22) is satisfied and the Ritz value is one of the $p$ smallest eigenvalues we found **then**
16: Add the Ritz pair to the list of found eigenpairs
17: Compute the left eigenvector, using the tool used in[30]
18: **end if**
19: **end if**

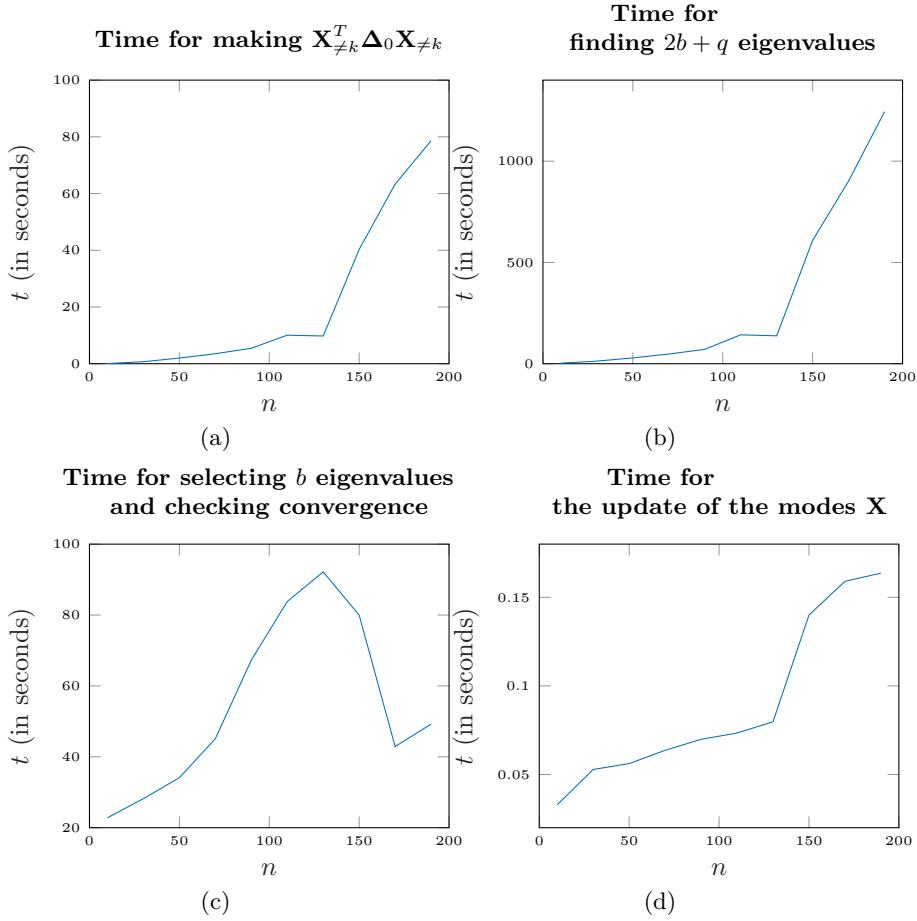



Figure 5.4: Figures corresponding to Example 5.4. We plot the time needed to perform the different parts of the algorithm when we increase the size $n$ of the matrices. We see that the solving of the projected eigenvalue problem is the most computationally demanding. The computational time needed for the selection and the convergence, depend on how many eigenvalues are converged and at which sweep it converges, explaining the decrease in the computational time in (c). We observe in (a) and (b) a kink in the curve at $n = 130$. We believe that this can be explained by the fact that the rank of $\mathbf{X}_{\neq k}$ at $n = 130$ was smaller than for other values of $n$.

# Chapter 6

# Conclusion

*Eind goed, al goed, zei Flip, en ik zei 't mee;*
*Hoe goed is dan een worst, zei Piet,*
*die heeft er twee.*
*(P.J. Harrebomée: Spreekwoordenboek der Nederlandsche taal. 1858-1862)*

In this thesis we investigate the use of projection methods for problems derived from the eigenvalue problem. For three types of problems we propose and analyse an algorithm that could solve the problem. Although in all problems we search for the same subspace, i.e. the subspace that contains a good approximation of some eigenvectors, we proved here that there are multiple ways to construct it. The idea behind the use of subspaces is that we can solve large eigenvalue problems which are expensive to compute by calculating several smaller eigenvalue problems. We give an overview of the different contributions of each chapter.After that, we formulate possible directions for further research.

**Chapter 3** In this chapter we made an algorithm for a parametrized eigenvalue problem where both matrices are symmetric and at least one of the two is positive-definite. We want to make a global approximation of the minimal eigenvalue over the parameter. The subspace is build up by several eigenvectors for a specific parameter value and its partial derivatives. We proved that adding derivatives of eigenvectors in some parameter values to the subspace leads to Hermite interpolation of the second degree in these parameter values. In each iteration of the algorithm we add eigenvectors and its derivatives for parameter values for which the residual norm is the largest. The algorithm is stopped the moment the residual norm for all parameter values is below some tolerance. The limitations ensure that (among other things) our approximations only

become increasingly better. Numerical experiments show the effectiveness of our algorithm.

**Chapter 4** As in Chapter 3, the algorithm presented in this chapter makes a global approximation of an eigenvalue over parameters. Here we limited ourselves to standard eigenvalue problems. The subspace is also built iteratively but now in each iteration we added info coming from all parameter values. The idea is that we extend a Krylov method for calculating extreme eigenvalues of a single matrix such that it can solve our problem. If we apply a Krylov method for all parameter values at the same time, the problem is that we need to add a vector to the subspace for each parameter value. These are too many vectors that we need to add, so the idea is that we make a low-rank decomposition for the tensor consisting of these vectors and to add the induced space to our subspace. This implies that we induce error in our algorithm, so we chose a Krylov method that it is robust against introduced error. We show that to a limited extent it is better to allow for a large error, as it does not harm the convergence and we need to add fewer vectors to our subspace. We validate our claims with numerical experiments.

**Chapter 5** The last problem that we tackle is computing eigenvalue-tuples of multiparameter eigenvalue problems for which one of the values in the tuple is minimal. These kind of problems are equivalent with solving a (possible very large) generalised eigenvalue problem. To the best of our knowledge, the current solvers for this type of problems are limited to three parameters for a reasonably large size of the involved matrices. We develop an algorithm based on the algorithm for calculating eigenvalues of a symmetric matrix presented as an operator on a tensor in tensor-train format described in [12]. We prove in this chapter that the associated generalised eigenvalue problem can be efficiently presented in this tensor format. We extend the algorithm in [12] such that it can solve our generalised eigenvalue problem using the fact that all eigenvectors are of rank-one seeing as a tensor. With numerical experiments we show the effectiveness of our algorithm and how the computational time depend on the the number of equations and the size of the involved matrices. It is the first time a method based on Tensor-Train approximations is used in the context of multiparameter eigenvalue problems.

## 6.1 Future research directions

We discuss a few research directions.

- In Chapter 3 we discussed the calculation of extremal eigenvalues of a symmetric matrix pair with one of the matrices positive definite. The reason why we are so restrictive is that there are theorems that estimates the error on the eigenvalues from above. If, as in the other chapters, we measure the quality of an approximation by the residual, we can potentially remove some restrictions.

- In Section 4.6 we saw how we can calculate eigenvalues close to a particular shift. The restriction was that only $\mathbf{A}_0$ was of full rank and that all other matrices where of rank one. We did this as we did not have a good way to solve parametrized systems. We performed experiments with the method in [38] and with a method similar to what was explained in Section 1.3.1. The former seemed not robust enough and the latter was quite slow. As we only came up with this idea when the paper was already written, we did not elaborate further on this topic.

- As was mentioned earlier, the tensor method proposed for solving multiparameter eigenvalue problems is the first of its kind. This opens the possibility of using similar methods for problems that can be written as large multiparameter eigenvalue problems. For example, in [31], it is proven that a quadratic two-parameter eigenvalue problem can be linearised into a five-parameter eigenvalue problem. Using the specific properties of this five-parameter eigenvalue problem, one can develop a specific solver based on the algorithm proposed in Chapter 5.

- The algorithm in Chapter 5 makes a subspace of size $r^2n$ with $n$ the size of the involved matrices and $r$ an internal rank in the TT-representation. In practice $r$ is at least 6. For practical reasons, we cannot make the subspace too large, so we restrict $n$ to 200. We think it is possible to combine our method with the method in [30] to solve multiparameter eigenvalue problems with $m > 3$ and $n > 200$.

# Appendix A

# Helmholtz equation with ellipsoidal coordinates

*All exact science is dominated*
*by the idea of approximation*
*(Bertrand Russell)*

In this appendix we explain properties of ellipsoidal coordinates and we explain how to retrieve the ordinary differential equations (1.17) from the Helmholtz equation.

## A.1 Ellipsoidal coordinates

We define the ellipsoidal coordinates $(u_1, u_2, u_3)$ implicitly by families of orthogonal confocal quadratic surfaces. These families are defined by the equation

$$\frac{x^2}{u^2 - a^2} + \frac{y^2}{u^2 - b^2} + \frac{z^2}{u^2 - c^2} = 1, a \geq b \geq c \tag{A.1}$$

with $x, y, z$ the Cartesian coordinates. Different values of $u$ lead to different types of surfaces. We only consider $u = u_1 > a$ which leads that (A.1) represents an ellipsoid. The relation between the ellipsoidal coordinates $(u_1, u_2, u_3)$ and the Cartesian coordinates $(x_1, x_2, x_3)$ are given by

$$\begin{cases} x^2 &= \dfrac{\left(u_1^2-a^2\right)\left(u_2^2-a^2\right)\left(u_3^2-a^2\right)}{\left(a^2-c^2\right)\left(a^2-b^2\right)} \\ y^2 &= \dfrac{\left(u_1^2-b^2\right)\left(u_2^2-b^2\right)\left(u_3^2-b^2\right)}{\left(b^2-c^2\right)\left(b^2-a^2\right)} \\ z^2 &= \dfrac{\left(u_1^2-c^2\right)\left(u_2^2-c^2\right)\left(u_3^2-c^2\right)}{\left(a^2-c^2\right)\left(b^2-c^2\right)} \end{cases} \tag{A.2}$$

with $u_1 > a > u_2 > b > u_3 > c$. Recall that keeping $u_1 = u$ fixed, leads to an ellipsoid surface. The radii of the ellipsoid $x_0, y_0, z_0$ relates to the variables $a, b, c$ in (A.1) via

$$\left(u^2-a^2\right)^{1/2} = x_0, \quad \left(u^2-b^2\right)^{1/2} = y_0, \quad \left(u^2-c^2\right)^{1/2} = z_0. \tag{A.3}$$

These have three equations with four variables, so we can choose $c = 0$ and we obtain

$$u = z_0, \quad a = \left(u^2-x_0^2\right)^{1/2}, \quad b = \left(u^2-y_0^2\right)^{1/2}.$$

The coordinate system $(u_1, u_2, u_3)$ can be designated by the metric coefficients, defined in general as

$$g_{ii} = \left(\frac{\partial x}{\partial u_i}\right)^2 + \left(\frac{\partial y}{\partial u_i}\right)^2 + \left(\frac{\partial z}{\partial u_i}\right)^2, \quad i = 1, 2, 3 \tag{A.4}$$

It can be seen in [103, Chapter 2] that the Laplacian is

$$\nabla^2 = g^{-\frac{1}{2}} \sum_{i=1}^{3} \frac{\partial}{\partial u_i}\left(\frac{g^{\frac{1}{2}}}{g_{ii}}\frac{\partial}{\partial u_i}\right), \tag{A.5}$$

with $g = g_{11}g_{22}g_{33}$. In the case of ellipsoidal coordinates, we obtain

$$\begin{aligned} g_{11} &= \frac{\left(u_1^2-u_2^2\right)\left(u_1^2-u_3^2\right)}{\left(u_1^2-a^2\right)\left(u_1^2-b^2\right)} \\ g_{22} &= \frac{\left(u_2^2-u_1^2\right)\left(u_2^2-u_3^2\right)}{\left(u_2^2-a^2\right)\left(u_2^2-b^2\right)} \\ g_{33} &= \frac{\left(u_3^2-u_1^2\right)\left(u_3^2-u_2^2\right)}{\left(u_3^2-a^2\right)\left(u_3^2-b^2\right)}, \end{aligned} \tag{A.6}$$

which means that

$$g^{1/2} = \frac{\left(u_1^2-u_2^2\right)\left(u_1^2-u_3^2\right)\left(u_2^2-u_3^2\right)}{\sqrt{-\left(u_1^2-a^2\right)\left(u_1^2-b^2\right)\left(u_2^2-a^2\right)\left(u_2^2-b^2\right)\left(u_3^2-a^2\right)\left(u_3^2-b^2\right)}}.$$

From this, we deduce that the Laplacian (A.5) is equal to

$$\begin{aligned}\nabla^2 &= \frac{\sqrt{(u_1^2-a^2)(u_1^2-b^2)}}{(u_1^2-u_2^2)(u_1^2-u_3^2)}\frac{\partial}{\partial u_1}\left[\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\frac{\partial}{\partial u_1}\right.\\ &+\frac{\sqrt{(u_2^2-a^2)(u_2^2-b^2)}}{(u_2^2-u_1^2)(u_2^2-u_3^2)}\frac{\partial}{\partial u_2}\left[\sqrt{(u_2^2-a^2)(u_2^2-b^2)}\frac{\partial}{\partial u_2}\right.\\ &+\left.\left.\frac{\sqrt{(u_3^2-a^2)(u_3^2-b^2)}}{(u_3^2-u_1^2)(u_3^2-u_2^2)}\frac{\partial}{\partial u_3}\left[\sqrt{(u_3^2-a^2)(u_3^2-b^2)}\frac{\partial}{\partial u_3}\right]\right]\right].\end{aligned} \tag{A.7}$$

In the next section we see how we can solve the Helmholtz equation when using ellipsoidal coordinates.

## A.2 Solving Helmholtz equation

To recall, the Helmholtz equation is given by

$$\nabla^2 v + \alpha^2 v = 0. \tag{A.8}$$

In this section we focus on how we can retrieve the three ordinary differential equations in (1.17) via the separability property. As we assume it is separable, we know there exists solution $v_1, v_2$ and $v_3$ such that

$$v = v_1(u_1)v_2(u_2)v_3(u_3).$$

If follows from (A.7) that the Helmholtz can be written in the form

$$\begin{aligned}&\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\left(u_2^2-u_3^2\right)v_2(u_2)v_3(u_3)\tfrac{\partial}{\partial u_1}\left[\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\tfrac{\partial v_1(u_1)}{\partial u_1}\right]\\ &+\sqrt{(a^2-u_2^2)(u_2^2-b^2)}\left(u_1^2-u_3^2\right)v_1(u_1)v_3(u_3)\tfrac{\partial}{\partial u_2}\left[\sqrt{(a^2-u_2^2)(u_2^2-b^2)}\tfrac{\partial v_2(u_2)}{\partial u_2}\right]\\ &+\sqrt{(a^2-u_3^2)(b^2-u_3^2)}\left(u_1^2-u_2^2\right)v_1(u_1)v_2(u_2)\tfrac{\partial}{\partial u_3}\left[\sqrt{(a^2-u_3^2)(b^2-u_3^2)}\tfrac{\partial v_3(u_3)}{\partial u_3}\right]\\ &+\alpha^2\left(u_1^2-u_2^2\right)\left(u_2^2-u_3^2\right)\left(u_1^2-u_3^2\right)v_1(u_1)v_2(u_2)v_3(u_3)=0.\end{aligned} \tag{A.9}$$

By dividing this by $v$, we get

$$\begin{aligned}&\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\left(u_2^2-u_3^2\right)\tfrac{1}{v_1(u_1)}\tfrac{\partial}{\partial u_1}\left[\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\tfrac{\partial v_1(u_1)}{\partial u_1}\right]\\ &+\sqrt{(a^2-u_2^2)(u_2^2-b^2)}\left(u_1^2-u_3^2\right)\tfrac{1}{v_2(u_2)}\tfrac{\partial}{\partial u_2}\left[\sqrt{(a^2-u_2^2)(u_2^2-b^2)}\tfrac{\partial v_2(u_2)}{\partial u_2}\right]\\ &+\sqrt{(a^2-u_3^2)(b^2-u_3^2)}\left(u_1^2-u_2^2\right)\tfrac{1}{v_3(u_3)}\tfrac{\partial}{\partial u_3}\left[\sqrt{(a^2-u_3^2)(b^2-u_3^2)}\tfrac{\partial v_3(u_3)}{\partial u_3}\right]\\ &+\alpha^2\left(u_1^2-u_2^2\right)\left(u_2^2-u_3^2\right)\left(u_1^2-u_3^2\right)v_1(u_1)v_2(u_2)v_3(u_3)=0.\end{aligned} \tag{A.10}$$

If follows immediately that the solution of following system of equations

$$\begin{cases} \sqrt{(u_1^2-a^2)(u_1^2-b^2)}\frac{\partial}{\partial u_1}\left[\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\frac{\partial v_1}{\partial u_1}\right] = \left(-\alpha^2 u_1^4+\rho u_1^2-\lambda\right)v_1 \\ \sqrt{(a^2-u_2^2)(u_2^2-b^2)}\frac{\partial}{\partial u_2}\left[\sqrt{(a^2-u_2^2)(u_2^2-b^2)}\frac{\partial v_2}{\partial u_2}\right] = -\left(-\alpha^2 u_2^4+\rho u_2^2-\lambda\right)v_2 \\ \sqrt{(a^2-u_3^2)(b^2-u_3^2)}\frac{\partial}{\partial u_3}\left[\sqrt{(a^2-u_3^2)(b^2-u_3^2)}\frac{\partial v_3}{\partial u_3}\right] = \left(-\alpha^2 u_3^4+\rho u_3^2-\lambda\right)v_3 \end{cases} \tag{A.11}$$

is also a solution of (A.10).

By expanding the term $\frac{\partial}{\partial u_1}\left[\sqrt{(u_1^2-a^2)(u_1^2-b^2)}\frac{\partial v_1}{\partial u_1}\right]$ and analogue for the other two equations, we obtain that the three equations of (A.11) are all of the form

$$\left(u_i^2-a^2\right)\left(u_i^2-b^2\right)\frac{\partial^2 v_i}{\partial u_i^2}+u_i\left[2u_i^2-\left(a^2+b^2\right)\right]\frac{\partial v_i}{\partial u_i}+\left[\alpha^2 u_i^4-\rho u_i^2+\lambda\right]v_i=0 \tag{A.12}$$

for $i=1,2,3$. Without lose of generality we assume that $i=1$ in the rest of this section. Let $s_1=u_1^2$ be a transformation of variables, then from

$$\frac{\partial v_1}{\partial u_1}=2\sqrt{s_1}\frac{\partial v_1}{\partial s_1}, \qquad \frac{\partial^2 v_1}{\partial u_1^2}=4s_1\frac{\partial^2 v}{\partial s_1^2}+2\frac{\partial v_1}{\partial s_1}$$

we obtain that (A.12) is equivalent to

$$\begin{aligned} s_1(s_1-a^2)(s_1-b^2)\frac{\partial^2 v_1}{\partial s_1^2}+\frac{1}{2}(3s_1^2-2(a^2+b^2)s_1+a^2b^2)\frac{\partial v_1}{\partial s_1}+ \\ \frac{1}{4}(\kappa-\rho s_1+\alpha^2 s_1^2)v_1=0. \end{aligned} \tag{A.13}$$

If we further do the transformation $t_1=s_1/b^2$, then from

$$\frac{\partial v_1}{\partial s_1}=\frac{\partial v_1}{\partial t_1}\frac{1}{b^2}, \qquad \frac{\partial^2 v_1}{\partial s_1^2}=\frac{\partial^2 v}{\partial t_1^2}\frac{1}{b^4}$$

we get

$$\begin{aligned} t_1(t_1-1)\left(t_1-\tfrac{a^2}{b^2}\right)\tfrac{\partial^2 v_1}{\partial t_1^2}+\tfrac{1}{2}\left[3t_1^2-2\left(1+\tfrac{a^2}{b^2}\right)t_1+\tfrac{a^2}{b^2}\right]\tfrac{\partial v_1}{\partial t_1}+ \\ \tfrac{1}{4}\left[\tfrac{\kappa}{b^2}-\rho t_1+\alpha^2 b^2 t_1^2\right]v_1=0. \end{aligned} \tag{A.14}$$

By defining

$$c=\frac{a^2}{b^2}, \quad \mu_1=\frac{1}{4}\frac{\kappa}{b^2}, \quad \mu_2=\frac{\rho}{4}, \quad \mu_3=\frac{\alpha^2 b^2}{4},$$

we get the expression in (1.17).

# Index


$\epsilon$-pseudospectral abcissa, 9
$\epsilon$-pseudospectrum, 9
$j$-mode product, 27

algebraic multiplicity, 2
Arnoldi method, 57
asymptotically stable, 5

B-orthonormal, 3
basis, 21
block-TT format, 29

CP decomposition, 28

defective eigenvalue, 2
distance to instability, 10

eigenpair, 2
eigenspace, 2
eigenvalue, 2
eigenvalue-tuple, 13, 79
eigenvector, 2
eigenvector-tuple, 13, 79

fiber, 26
flattening, 27
frame matrix, 81

Gram-Schmidt method, 22, 23

Hadamard product, 33
Hadamard-product, 28
Hermitian, 3

interface, 81

Jacobian matrix, 5

Kronecker product (left), 28
Krylov subspace, 25

left-orthogonality, 81

matricization, 27
multiparameter eigenvalue problem, 13, 79

oblique projection method, 22
orthogonal projection methods, 22
orthonormal, 3, 23

positive definite, 3
projected eigenvalue problem, 22

Rayleigh-Ritz procedure, 22
Residual Arnoldi method, 58
right-orthogonality, 81
Ritz pair, 22
Ritz vector, 22

simple eigenvalue, 2
spectral abscissa, 9
spectrum, 2
standard eigenvalue problem, 2
steady state solution, 5
subspaces, 21
sweep, 80

tensor rank, 34
Tensor-Train (TT) decomposition, 28

thick restarting, 63
TT-core, 29
TT-format, 29
TT-ranks, 29
Tucker decomposition, 28
unfolding, 27
weak formulation, 7

# Bibliography


[1] Andreev, R., and Schwab, C. Sparse Tensor Approximation of Parametric Eigenvalue Problems. Springer, Berlin, Heidelberg, 2012.

[2] Arnoldi, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics* (1951).

[3] Atkinson, F. *Multiparameter Eigenvalue Problems*, vol. 1. Elsevier, 1972.

[4] Bauer, F. L., and Fike, C. T. Norms and exclusion theorems. *Numerische Mathematik 2*, 1 (1960).

[5] Bouaricha, A. Tensor-Krylov methods for large nonlinear equations. *Computational Optimization and Applications* (1996).

[6] Bouaricha, A., and Schnabel, R. B. Tensor methods for large sparse systems of nonlinear equations. *Mathematical Programming, Series B* (1998).

[7] Breda, D., Maset, S., and Vermiglio, R. Stability of linear delay differential equations: A numerical approach with MATLAB. In *Briefs in Control, Automation and Robotics*. Springer, 2015.

[8] Burroughs, E. A., Romero, L. A., Lehoucq, R. B., and Salinger, A. G. Linear stability of flow in a differentially heated cavity via large-scale eigenvalue calculations. *International Journal of Numerical Methods for Heat and Fluid Flow* (2004).

[9] Camps, D. *Pole swapping methods for the eigenvalue problem Rational QR algorithms*. Phd thesis, KU Leuven, 2019.

[10] Carroll, J. D., and Chang, J. J. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika* (1970).

[11] Cools, R., Michiels, W., and Nuyens, D. Lecture notes in "modellering en simulatie", 2017.

[12] Dolgov, S. V., Khoromskij, B. N., Oseledets, I. V., and Savostyanov, D. V. Computation of extreme eigenvalues in higher dimensions using block tensor train format. *Computer Physics Communications* (2014).

[13] Dolgov, S. V., and Oseledets, I. V. TT-Toolbox, 2020.

[14] Dong, B., Yu, B., and Yu, Y. A homotopy method for finding all solutions of a multiparameter eigenvalue problem. *SIAM Journal on Matrix Analysis and Applications* (2016).

[15] Eckart, C., and Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* (1936).

[16] Elman, H. C., Meerbergen, K., Spence, A., and Wu, M. Lyapunov Inverse Iteration for Identifying Hopf Bifurcations in Models of Incompressible Flow. *SIAM Journal on Scientific Computing* (2012).

[17] Elman, H. C., and Tengfei, S. U. Low-rank solution methods for stochastic eigenvalue problems. *SIAM Journal on Scientific Computing* (2019).

[18] Fenzi, L., and Michiels, W. Robust stability optimization for linear delay systems in a probabilistic framework. *Linear Algebra and Its Applications* (2017).

[19] Fenzi, L., and Michiels, W. Polynomial (chaos) approximation of maximum eigenvalue functions. *Numerical Algorithms* (2019).

[20] Garg, S. Derivatives of eigensolutions for a general matrix. *AIAA Journal* (1973).

[21] Ghanem, R., and Ghosh, D. Efficient characterization of the random eigenvalue problem in a polynomial chaos decomposition. *International Journal for Numerical Methods in Engineering 72*, 4 (2007).

[22] Gibbs, T., Grilli, J., Rogers, T., and Allesina, S. Effect of population abundances on the stability of large random ecosystems. *Physical Review E* (2018).

[23] Grimes, R. G., Lewis, J. G., and Simon, H. D. Eigenvalue problems and algorithms in structural engineering. *North-Holland Mathematics Studies* (1986).

[24] Guglielmi, N., and Overton, M. L. Fast algorithms for the approximation of the pseudospectral abscissa and pseudospectral radius of a matrix. *SIAM Journal on Matrix Analysis and Applications* (2011).

[25] Hackbusch, W. *Tensor Spaces and Numerical Tensor Calculus.* Springer International Publishing, 2019.

[26] Hager, W. W. Updating the inverse of a matrix. *SIAM Review* (1989).

[27] Hakula, H., Kaarnioja, V., and Laaksonen, M. Approximate methods for stochastic eigenvalue problems. *Applied Mathematics and Computation* (2015).

[28] Harshman, R. a. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics* (1970).

[29] Hesthaven, J. S., Stamm, B., and Zhang, S. Efficient greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods. *ESAIM: Mathematical Modelling and Numerical Analysis 48*, 1 (2014).

[30] Hochstenbach, M. E., Meerbergen, K., Mengi, E., and Plestenjak, B. Subspace methods for three-parameter eigenvalue problems. *Numerical Linear Algebra with Applications* (2019).

[31] Hochstenbach, M. E., Muhič, A., and Plestenjak, B. On linearizations of the quadratic two-parameter eigenvalue problem. *Linear Algebra and Its Applications 436*, 8 (2012).

[32] Holtz, S., Rohwedder, T., and Schneider, R. The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format. *SIAM Journal on Scientific Computing 34*, 2 (2012).

[33] Jolliffe, I. T. Principal Component Analysis, Second Edition. *Encyclopedia of Statistics in Behavioral Science* (2002).

[34] Jurkat, W. B., and Ryser, H. J. Matrix factorizations of determinants and permanents. *Journal of Algebra* (1966).

[35] Kangal, F., Meerbergen, K., Mengi, E., and Michiels, W. A subspace method for large-scale eigenvalue optimization. *SIAM Journal on Matrix Analysis and Applications 39*, 1 (2018).

[36] Kato, T. On the Upper and Lower Bounds of Eigenvalues. *Journal of the Physical Society of Japan 4*, 4-6 (1949).

[37] Kolda, T. G., and Bader, B. W. Tensor Decompositions and Applications. *SIAM Review* (2009).

[38] Kressner, D., and Tobler, C. Low-Rank Tensor Krylov Subspace Methods for Parametrized Linear Systems. *SIAM Journal on Matrix Analysis and Applications* (2011).

[39] Lax, P. D. *Linear Algebra and Its Applications*. Wiley-Interscience, 2007.

[40] Lee, C.-R. *Residual Arnoldi Methods : Theory, Package and Experiments*. Phd thesis, University of Maryland, 2007.

[41] Lee, C.-R., and Stewart, G. Analysis of the residual Arnoldi method. Tech. rep., University of Maryland, 2007.

[42] Meerbergen, K., and Plestenjak, B. A Sylvester-Arnoldi type method for the generalized eigenvalue problem with two-by-two operator determinants. *Numerical Linear Algebra with Applications* (2015).

[43] Meerbergen, K., and Roose, D. Matrix transformations for computing rightmost eigenvalues of large sparse non-symmetric eigenvalue problems. *IMA Journal of Numerical Analysis* (1996).

[44] Meerbergen, K., and Spence, A. Inverse iteration for purely imaginary eigenvalues with application to the detection of hopf bifurcations in large-scale problems. *SIAM Journal on Matrix Analysis and Applications* (2009).

[45] Meidani, H., and Ghanem Prof., R. Spectral power iterations for the random eigenvalue problem. *AIAA Journal* (2014).

[46] Michiels, W. Spectrum-based stability analysis and stabilisation of systems described by delay differential algebraic equations. *IET Control Theory and Applications* (2011).

[47] Nelson, R. B. Simplified calculation of eigenvector derivatives. *AIAA Journal* (1976).

[48] Nobari, A., Ouyang, H., and Bannister, P. Uncertainty quantification of squeal instability via surrogate modelling. *Mechanical Systems and Signal Processing* (2015).

[49] Olmstead, W. E., Davis, S. H., Rosenblat, S., and Kath, W. L. Bifurcation with memory. *SIAM Journal on Applied Mathematics* (1986).

[50] Oseledets, I. V. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing* (2011).

[51] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web. *World Wide Web Internet And Web Information Systems* (1998).

[52] Parlett, B. N. *The Symmetric Eigenvalue Problem*. SIAM, 1998.

[53] Plestenjak, B. A continuation method for a right definite two-parameter eigenvalue problem. *SIAM Journal on Matrix Analysis and Applications* (2000).

[54] Plestenjak, B. MultiParEig, 2020.

[55] Plestenjak, B., Gheorghiu, C. I., and Hochstenbach, M. E. Spectral collocation for multiparameter eigenvalue problems arising from separable boundary value problems. *Journal of Computational Physics* (2015).

[56] Quarteroni, A., Manzoni, A., and Negri, F. *Reduced Basis Methods for Partial Differential Equations: An introduction*, vol. 92 of *UNITEXT*. Springer, Cham, 2016.

[57] Rodriguez, J. I., Lim, L. H., and You, Y. Fiber product homotopy method for multiparameter eigenvalue problems. *ArXiv preprint* (2018).

[58] Rommes, J., and Martins, N. Exploiting structure in large-scale electrical circuit and power system problems. *Linear Algebra and Its Applications* (2009).

[59] Rozza, G., Huynh, D. B. P., and Patera, A. T. Reduced Basis Approximation and a Posteriori Error Estimation for Affinely Parametrized Elliptic Coercive Partial Differential Equations. *Archives of Computational Methods in Engineering 15*, 3 (2008).

[60] Ruymbeek, K., Meerbergen, K., and Michiels, W. Calculating the minimal/maximal eigenvalue of symmetric parametrized matrices using projection. *Numerical Linear Algebra with Applications 26*, 5 (2019).

[61] Saad, Y. *Numerical Methods for Large Eigenvalue Problems*. 2011.

[62] Schollwöck, U. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics* (2011).

[63] Seyranian, A. P., and Mailybaev, A. A. *Multiparameter Stability Theory with Mechanical Applications*, vol. 13. World Scientific, 2003.

[64] SimilarWeb. `https://www.similarweb.com/website/google.com/#overview`. Accessed: December 4, 2020.

[65] Sirkovic, P. *Low-rank methods for parameter-dependent eigenvalue problems and matrix equations*. Phd thesis, EPFL, 2016.

[66] Sirković, P., and Kressner, D. Subspace Acceleration for Large-Scale Parameter-Dependent Hermitian Eigenproblems. *SIAM Journal on Matrix Analysis and Applications 37*, 2 (2016).

[67] Sorensen, D. C. Implicit Application of Polynomial Filters in a k -Step Arnoldi Method. *SIAM Journal on Matrix Analysis and Applications* (1992).

[68] Strang, G. *Differential Equations and Linear Algebra*. Wellesley Cambridge Press, 2015.

[69] Temple, G. The Accuracy of Rayleigh's Method of Calculating the Natural Frequencies of Vibrating Systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 211*, 1105 (1952).

[70] Thompson, R. L., Mishra, A. A., Iaccarino, G., Edeling, W., and Sampaio, L. Eigenvector perturbation methodology for uncertainty quantification of turbulence models. *Physical Review Fluids* (2019).

[71] Trefethen, L. N. *Spectral Methods in MATLAB*. 2000.

[72] Trefethen, L. N., and Embree, M. *Spectra and Pseudospectra: the behavior of non-normal matrices and operators*. Princeton University Press, 2005.

[73] Tucker, L. R. Some mathematical notes on three-mode factor analysis. *Psychometrika* (1966).

[74] Van Der Aa, N., Ter Morsche, H., and Mattheij, R. Computation of eigenvalue and eigenvector derivatives for a general complex-valued eigensystem. *Electronic Journal of Linear Algebra 16* (2007).

[75] Wikipedia. Adrien-Marie Legendre — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Adrien-Marie_Legendre` Accessed: November 11, 2020.

[76] Wikipedia. Aleksey Krylov — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Aleksey_Krylov` Accessed: November 11, 2020.

[77] Wikipedia. Andrey Markov — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Andrey_Markov` Accessed: December 5, 2020.

[78] Wikipedia. Blaise Pascal — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Blaise_Pascal` Accessed: November 24, 2020.

[79] Wikipedia. E. H. Moore — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/E._H._Moore` Accessed: November 11, 2020.

[80] Wikipedia. Erhard Schmidt — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Erhard_Schmidt` Accessed: November 11, 2020.

[81] Wikipedia. Erwin Schrödinger — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Erwin_Schr%C3%B6dinger` Accessed: December 15, 2020.

[82] Wikipedia. Ferdinand Georg Frobenius — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Ferdinand_Georg_Frobenius Accessed: November 11, 2020.

[83] Wikipedia. Friedrich L. Bauer — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Friedrich_L._Bauer Accessed: November 11, 2020.

[84] Wikipedia. Gram–Schmidt process — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process Accessed: November 11, 2020.

[85] Wikipedia. Hermann von Helmholtz — Wikipedia, The Free Encyclopedia. https://nl.wikipedia.org/wiki/Hermann_von_Helmholtz Accessed: December 11, 2020.

[86] Wikipedia. Jacques Hadamard — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Jacques_Hadamard Accessed: November 11, 2020.

[87] Wikipedia. John William Strutt, 3rd Baron Rayleigh — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/John_William_Strutt,_3rd_Baron_Rayleigh Accessed: November 11, 2020.

[88] Wikipedia. Jørgen Pedersen Gram — Wikipedia, The Free Encyclopedia. https://nl.wikipedia.org/wiki/J%C3%B8rgen_Pedersen_Gram Accessed: November 11, 2020.

[89] Wikipedia. Ledyard Tucker — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Ledyard_Tucker Accessed: November 11, 2020.

[90] Wikipedia. Leopold Kronecker — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Leopold_Kronecker Accessed: November 11, 2020.

[91] Wikipedia. Oscar Perron — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Oskar_Perron Accessed: December 5, 2020.

[92] Wikipedia. Pafnuty Chebyshev — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Pafnuty_Chebyshev Accessed: November 24, 2020.

[93] Wikipedia. Peter Gustav Lejeune Dirichlet — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Peter_Gustav_Lejeune_Dirichlet Accessed: November 24, 2020.

[94] Wikipedia. Pierre-Simon Laplace — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Pierre-Simon_Laplace Accessed: December 11, 2020.

[95] Wikipedia. René Descartes — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Ren%C3%A9_Descartes Accessed: December 31, 2020.

[96] Wikipedia. Roger Penrose — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Roger_Penrose Accessed: November 11, 2020.

[97] Wikipedia. Tosio Kato — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Tosio_Kato Accessed: November 11, 2020.

[98] Wikipedia. Victor Gustave Robin — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Victor_Gustave_Robin Accessed: December 8, 2020.

[99] Wikipedia. Walter Edwin Arnoldi — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Walter_Edwin_Arnoldi Accessed: November 11, 2020.

[100] Wikipedia. Walther Ritz — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Walther_Ritz Accessed: November 11, 2020.

[101] Wikipedia. Adolf Hurwitz — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Adolf_Hurwitz Accessed: January 26, 2021.

[102] Wikipedia. Eberhard Hopf — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Eberhard_Hopf Accessed: January 27, 2021.

[103] Willatzen, M., and Lew Yan Voon, L. C. *Separable Boundary-Value Problems in Physics*. Wiley-VCH Verlag GmbH Co. KGaA, Weinheim, Germany, apr 2011.

[104] Woodbury, M. Inverting modified matrices. Tech. rep., 1950.

[105] Wu, G., and Zhang, L. On expansion of search subspaces for large non-Hermitian eigenproblems. *Linear Algebra and Its Applications* (2014).

[106] Ye, Q. Optimal expansion of subspaces for eigenvector approximations. *Linear Algebra and Its Applications* (2008).

# Curriculum vitae

**Personal information**

1. Name: Koen Ruymbeek
2. Date of birth: March 11, 1993
3. Place of birth: Lokeren, Belgium

**Education**

1. **March 2017 - April 2021**
   Phd. student in Engineering Science, Computer Science
   KU Leuven, Belgium
2. **September 2014 - September 2016**
   Master of Mathematics: Financial Mathematics
   University of Antwerp
3. **September 2011 - September 2014**
   Bachelor of Mathematics
   University of Antwerp

**Experience**

1. **September 2016 - January 2017**
   Application Scientist at Motulus
2. **January 2017 - March 2017**
   Scientific Assistant at University of Antwerp

# Publications and talks

*If I have ever made any valuable discoveries,*
*it has been more to patient attention*
*than to any other talent.*
*(Sir Isaac Newton)*

**Articles in international reviewed journals**

1. Ruymbeek, K, Vanroose, W. Algorithm for the reconstruction of dynamic objects in CT-scanning using optical flow. Journal of Computational and Applied Mathematics 367 (2020): 112459.
2. Ruymbeek, K, Meerbergen, K, Michiels, W. Calculating the minimal/maximal eigenvalue of symmetric parameterized matrices using projection. Numer Linear Algebra Appl. 2019; 26:e2263

**Submitted articles**

1. Ruymbeek, K, Meerbergen, K, Michiels, W. Tensor-Krylov method for computing eigenvalues of parameter-dependent matrices, submitted for publication to Journal of Computational and Applied mathematics.
2. Ruymbeek, K, Meerbergen, K, Michiels, W. Subspace method for multiparameter-eigenvalue problems based on tensor-train representations, submitted for publication to Numerical Linear Algebra with Applications

**Talks**

1. NASCA'18, July 2-6 2018, Kalamata, Greece: Title: 'Projection method for approximating the minimal eigenvalue of parametrized symmetric matrices' slides
2. ICIAM'19, July 15-19 2019, Valencia, Spain: Title: 'Projection method for approximating the expected value and variance of the minimal/maximal eigenvalue'
3. ApplMath'20, September 14-18 2020, Brijuni, Croatia (given via Zoom due to the Corona pandemic): Title: 'Tensor-Krylov methods for parametrized eigenvalue problems'



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
NUMA
Celestijnenlaan 200A box 2402
B-3001 Leuven
koen.ruymbeek@kuleuven.be
https://rkuoyemnbeek.github.io/