Dissertation

Computation of Approximate Border Bases and Applications

Dipl.-Inf. Univ. Jan Limbeck

Eingereicht an der Fakultät für Informatik und Mathematik
der Universität Passau als Dissertation zur Erlangung des
Grades eines Doktors der Naturwissenschaften

Submitted to the Department of Informatics and Mathematics
of the University of Passau in Partial Fulfilment of the Requirements
for the Degree of a Doctor in the Domain of Science

Betreuer / Supervisor:
Prof. Dr. Martin Kreuzer
Lehrstuhl für Mathematik mit Schwerpunkt Symbolic Computation
Universität Passau

Zweitgutachter / Second Assessor:
Prof. Dr. Tomas Sauer
Lehrstuhl für Mathematik mit Schwerpunkt Digitale Bildverarbeitung
Universität Passau

January 2014

Computation of Approximate Border Bases and Applications
Dissertation

Dipl.-Inf. Univ. Jan Limbeck

4th January 2014

Supervisor: Professor Dr. Martin Kreuzer
Chair of Symbolic Computation
University of Passau, Germany

Second Assessor: Professor Dr. Tomas Sauer
Chair of Mathematics
University of Passau, Germany

Contents

1 Introduction

Contents

This thesis deals with challenges associated with the computation of approximate border bases, a generalisation of border bases, in the context of the Algebraic Oil Research Project ([1]). The concept of approximate border bases was introduced by Kreuzer, Poulisse et al. in [28], as an effective mean to derive physically relevant polynomial models from measured data. The main advantage of this approach compared to alternative techniques currently in use in the (oil) industry is its power to derive polynomial models without additional a priori knowledge about the underlying physical system and its robustness with respect to noise in the measured data. One main result of [28], the so-called Approximate Vanishing Ideal (AVI) algorithm which can be used to compute approximate border bases, served as a starting point for further research which was conducted in this thesis. An aim of our work is to broaden its applicability to additional areas in the oil industry, like seismic imaging and the description of unconventional geological structures. For this purpose several new algorithms, where each one focuses on a specific task, are developed. The numerical aspects of the algorithms, which are a major concern for practical applications, are analysed in detail and a solid foundation of the underlying mathematical concepts is provided. A further contribution to the subject of approximate border bases is the achievement to improve the runtime of the core algorithms significantly, by modifying strategies known from literature that are capable of updating matrix factorisations in such a way that they are applicable in our setting. Finally, we compare our approach to the approximate H-basis algorithm of Sauer, which was presented in [41], and to the SOI and NBM methods which were proposed by Abbott, Fassino, et al. in [34] and [35], three other state of the art algorithms that are concerned with similar problems. For all three algorithms we compare the computed results and in the case of the SOI and NBM methods we also compare the computational performance. Further details and some conclusions are contained in Section 1.2, where we present our main results.

Approximate border bases - no matter how useful they are in practice - have one important disadvantage: they currently lack an extensive body of theory behind them. One approach to

mitigate this problem is addressed in this work, namely the approach to construct "close by" exact border bases. To achieve this, we establish a link between this task, called the rational recovery problem here, and the problem of simultaneously quasi-diagonalising a set of complex matrices. As simultaneously quasi-diagonalising a set of matrices is not a standard topic in numerical linear algebra, we introduce and study a new algorithm for this purpose that is based on the classical Jacobi eigenvalue algorithm. Additionally, we motivate a second solution of the rational recovery problem via the minimisation of a sum of squares expression.

1.1 Motivation

The modelling of complex physical systems and their mathematically accurate description are a major concern for industrial companies that want to understand, influence, and ultimately control, physical processes and systems. The determining quantities in such processes can include temperatures, pressures, concentrations of chemical substances, conductivity and valve settings to name just a few. Understanding the relations between these quantities in such a system is in general a non-trivial task and requires profound knowledge about the laws of physics and the problem at hand. The more complex these systems become in terms of the number of parameters they involve, the more difficult it gets to understand their behaviour. Therefore, it is desirable if some of these relations, e.g. the polynomial ones, can be found in an automated way. Polynomial approximation is a classical attempt in this direction, but it is unable to find back all polynomial relations that exist between the measured quantities. Successful attempts in this direction were made by Sauer in [41], using the concept of approximate H-Bases, and by Kreuzer, Poulisse et al. in [28], using the concept of approximate border bases. Based on the work of the latter, we further improve the AVI algorithm, in form of the ABM and extended ABM algorithms, which are both numerically more robust and are also applicable to new contexts.
One such context that is for instance very relevant to the oil industry is the field of seismic imaging. It is mostly concerned with visualising the geological structures in the subsurface. This process is usually carried out by performing a seismic survey, which is created using an artificial acoustic source and an array of "geophones" which measure and record the reflections of the acoustic waves from the underground. These shot records are then processed to generate pictures of the subsurface. Most of the methods that are in use today and that do not rely on a full inversion of the wave equation, which models the propagation of the acoustic wave in the subsurface, make strong assumptions about the geometry of the underground. These assumptions are mainly necessary in current methods to achieve convergence and a reasonable runtime. However, they may be unrealistic in practice and lead to wrong results. Thus it is desirable to explore new techniques that are more data driven than the procedures which are currently available. For this purpose we investigate in detail how our new algorithms, the ABM and the extended ABM algorithm, can be applied to seismic imaging.

Of both theoretical and practical interest is the rational recovery problem. As mentioned before, it is concerned with the construction of an exact border basis in the vicinity of a given approximate one. Once we have computed an exact border basis we are in the favourable situation that we can apply standard techniques from computer algebra. However, the techniques which

we propose for this purpose are also of value when we are dealing with essentially exact border bases that were computed in floating point arithmetic (for instance when speed is a major concern). This specific topic was also investigated by Stetter in his book Numerical Polynomial Algebra ([49]), who also gave an overview of the currently existing techniques. It turns out that our method of simultaneous quasi-diagonalisation is numerically superior compared to the approaches that are advocated by Stetter in his book.

1.2 Main Results

Before we can present the main results of this thesis we need to clarify the setting in which we operate. Let $P = K[x_1, ..., x_n]$ with $K = \mathbb{R}$ or $K = \mathbb{C}$ be the polynomial in n indeterminates over the real or complex numbers and let $\mathbb{X} = \{p_1, ..., p_s\} \subset K^n$ be a finite set of points. It is well-known that once we fix a term ordering σ on the monoid of terms $\mathbb{T}^n$ in P, we can compute a Gröbner basis or a border basis of the vanishing ideal $\mathcal{I}(\mathbb{X})$ of $\mathbb{X}$ in P with the help of suitable variants of the Buchberger-Möller (BM) algorithm. Without a term ordering it is also possible to compute an H-basis of $\mathcal{I}(\mathbb{X})$ with a specific variant of the BM algorithm. In particular, we are interested in a system of generators of $\mathcal{I}(\mathbb{X})$ because all polynomial relations between the coordinates of the points in $\mathbb{X}$ can be expressed in this way. Suppose that $\mathbb{X}$ contains some "characteristic" measurements of an unknown or partially unknown physical process that can be described (or at least well approximated) by polynomials in the coordinates of the points in $\mathbb{X}$. Then it is possible to study this process via $\mathcal{I}(\mathbb{X})$. Specifically, we are interested in low degree and possibly sparse polynomials in the vanishing ideal of $\mathbb{X}$, as these can be more easily "interpreted" in the context of the physical system. However, in real world applications, we are facing a significant complication, due to the presence of measurement errors in $\mathbb{X}$ which makes it virtually impossible to find low degree polynomials in $\mathcal{I}(\mathbb{X})$. This has the consequence that the classical versions of the BM algorithm are not particularly well-suited if the points in $\mathbb{X}$ are "noisy". For this purpose Kreuzer, Poulisse et al. have extended the concept of (exact) border bases to approximate border bases in [28]. This extension to approximate input data allows to retrieve low degree polynomial relations among the coordinates of $\mathbb{X}$ that hold only approximately. An algorithm that is capable of computing such an approximate border basis is the Approximate Vanishing Ideal (AVI) algorithm that was proposed in [28]. Before we are able to sketch the main results of this thesis concerning approximate border basis, let us introduce them more formally (compare also Definition 4.1.4). Let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal consisting of terms in $\mathbb{T}^n$, let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be its border, and let $G = \{g_1, ..., g_\nu\}$ be an $\mathcal{O}$-border prebasis of the ideal $I = \langle g_1, ..., g_\nu \rangle$ in P. So, each g_j is of the form $g_j = b_j - \sum_{i=1}^{\mu} c_{ij} t_i$ with $c_{ij} \in K$. For every pair (i, j) such that b_i and b_j are neighbours in $\partial\mathcal{O}$, compute the normal remainder $S'_{ij} = \mathrm{NR}_{\mathcal{O},G}(S_{ij})$ of the S-polynomial of g_i and g_j with respect to G. The set G is an ε-approximate $\mathcal{O}$-border basis of the ideal I if $\|S'_{ij}\| \leq \varepsilon$ for all such pairs (i, j), where $\|S'_{ij}\|$ is the Euclidean norm of the coefficient vector of S'_{ij}.

In Chapter 4 of this thesis, three newly developed algorithms that are capable of computing approximate border bases, namely the Approximate Buchberger-Möller (ABM), the Border Basis (BB) ABM, and the extended ABM algorithm are presented and discussed in detail.

These algorithms are, just like the AVI algorithm, also variants of the BM algorithm for border bases (18) but adapted to inexact input data. The most important differences and improvements over specific properties of the AVI algorithm are pointed out, proven, and illustrated with examples. To be able to better present these results, we briefly explain some basic properties of the AVI and of the BM algorithm.

The AVI algorithm processes iteratively all polynomials of a degree at a time and uses the singular value decomposition (see Definition 2.5.31) extensively in order to compute the almost vanishing polynomials that make up the resulting approximate border basis. The original version of the Buchberger-Möller algorithm that was proposed in [21] proceeds term by term and uses the (exact) kernel of a matrix in order to compute the (exactly) vanishing polynomials. The modification to proceed degree by degree was introduced in the AVI algorithm mainly to achieve competitive performance for industrial size datasets, as the computation of the SVD is a rather expensive tool of numerical linear algebra. One consequence of this approach is that the ε-approximate kernel (see Definition 4.1.5) may have dimension greater than one. The border prebasis structure (see Definition 3.1.7) of the polynomials in the approximate border basis makes it necessary to compute a reduced row echelon form of the vectors in the approximate kernel. This is the only way to make sure that each polynomial in the approximate border basis contains a unique border term. The computation of such a reduced row echelon form is numerically a delicate matter. In order to overcome this issue, Kreuzer et al. have suggested a method (compare Algorithm 20) that is built upon QR decomposition via Gram-Schmidt orthonormalisation. One consequence of this numerically stabilised RREF computation is that it is only possible to state an upper bound in terms of the input parameters for the Euclidean norm of the evaluation residual of the polynomials in the approximate border basis with respect to $\mathbb{X}$. Given $\mathbb{X} = \{p_1, ..., p_s\}$, small numbers $\varepsilon > \tau > 0$ and a degree compatible term ordering σ on $\mathbb{T}^n$, the AVI algorithm computes two sets $G = \{g_1, ..., g_\nu\}$ and $\mathcal{O} = \{t_1, ..., t_\mu\}$. The authors of [28] could show that for each $g_i \in G$ the inequality $\|\text{eval}_{\mathbb{X}}(g_i)\|_2 < \varepsilon\sqrt{\nu} + \tau\nu(\mu+\nu)\sqrt{s}$ holds. This bound depends on the input parameters ε and τ, the number of input points, the size of the order ideal and the size of G. For example, if G contains 16 elements, which is a rather modest size, then the evaluation residual may already grow by a factor of 4. It should be noted that these are worst case estimates. Nevertheless, the bound obtained in this way may be impractical for some applications that require us to choose large values of ε. Furthermore, the deviation of the computed approximate border basis from an exact border basis depends significantly on $\|\text{eval}_{\mathbb{X}}(g_i)\|_2$ (compare Claim 4 of Theorem 4.2.2). If the terms are processed term by term, it is possible to state a clean and simple upper bound. However, as pointed out before, this results in a high computational cost because the SVD has to be computed for each term that is considered.

The ABM algorithm addresses this problem by recasting the problem of computing the ε-approximate kernel of a matrix via the SVD to the problem of solving a related homogeneous least squares problem (see Definition 2.10.2 and Section 2.13). The problem can thus be stated as a Hermitian eigenvalue/eigenvector problem. This different characterisation allows us to treat each term individually with only a minor performance penalty. This is possible by utilising the special structure present in the Hermitian matrices which have to be decomposed in each itera-

tion in which we have to solve a homogeneous least squares problem (see Remark 4.3.10). With this "updating" procedure we achieve a cubic runtime in the number of input points, whereas the total runtime would have been in $O\left(s^4\right)$ if we would have computed a SVD for each new term. So, while we maintain competitive performance, we manage to establish a tight direct bound on the polynomials in the set G computed by the ABM algorithm. In Theorem 4.3.1 we manage to show that $\|\text{eval}_{\mathbb{X}}(g_i)\|_2 \leq \varepsilon$ for each $g_i \in G$, given the same input data as in the AVI algorithm. This also yields a sharper upper bound for the deviation of the computed approximate border basis from an exact one.
Additionally, the numerical stability of the method and of the computed solutions is analysed in detail. The corresponding result is contained in Theorem 4.3.7. This analysis is based on the foundations that are laid in Chapter 2 and more specifically in Subsection 2.9. Note, that these numerical aspects were not considered in [28]. Nevertheless they are important for practical applications, as unstable solutions need to be detected and handled with care in the presence of inexact input data.
Furthermore, the ABM algorithm is applicable to complex input data $\mathbb{X} \subset \mathbb{C}^n$, a case which was also not covered in [28]. The necessary results from the literature are also introduced in Chapter 2. This generalisation is rather straightforward, but it is also an important requirement for practical applications, for instance when working with Fast Fourier transformed (FFT) input data, which allows us to find polynomial relations in the frequency domain.
The AVI and the ABM algorithm are particularly useful if we want to investigate the approximate polynomial relations that exist between the coordinates of the points in $\mathbb{X}$. However, the application of both algorithms is no longer straightforward if one wants to decide if one specific measurement is a (multivariate) polynomial function in other measurements. The complications that may arise, and a possible solution, the extended ABM algorithm, are discussed in Section 4.4. The extended ABM algorithm takes as input again $\mathbb{X}$ and $\varepsilon \geq 0$ but additionally a set of 1-dimensional points $\mathbb{V} = \{v_1, ..., v_s\} \subset \mathbb{C}$ and an additional parameter $\tau \geq 0$. The algorithm can be seen as a true "extension" of the ABM algorithm, as it still computes an approximate border basis with respect to $\mathbb{X}$ in form of sets G and $\mathcal{O}$ but additionally it also computes a (possibly empty) set of polynomials $H = \{h_1, ..., h_\kappa\}$ such that for each h_i we have $\left\|\text{eval}_{\mathbb{X}}(h_i) - \mathbb{V}^{\text{tr}}\right\| \leq \tau$ (see Theorem 4.4.3). So, the polynomials in H together with the polynomials in G allow us to characterise essentially all polynomials that approximate $\mathbb{V}$ when evaluated on $\mathbb{X}$. In this way, the extended ABM algorithm provides additional control over the modelling process by guaranteeing a tight bound on the residual error of the computed model polynomials for $\mathbb{V}$. Therefore, it increases the chances, compared to the AVI and ABM algorithms, to obtain a physically sensible polynomial model. Consider Section 6.1 for further implications and details.
Another important accomplishment of the extended ABM algorithm is that it enables us to reuse the matrix factorisations that are utilised to compute the polynomials in the set G also to construct the polynomials in the set H (compare Remark 4.4.9). In this way the extended ABM algorithm does not suffer a significant slow down compared to the ABM algorithm and therefore, it can still be applied to industrial size datasets. Moreover, we could show in Proposition 4.4.8 that the algorithm is in $O\left(s^3\right)$ if it is implemented in a sensible way. The details about this

specific aspect are contained in Subsection 4.3.1.
Furthermore, we were also able to establish a theoretical bound that concerns the sensitivity of the polynomials h_i with respect to perturbations in the input data. In Theorem 4.4.3 we show that the condition number (see Definition 2.7.9) for the solution of the inhomogeneous least squares problem, via which we obtain the coefficients of the polynomials h_i, is bounded by $\frac{s\cdot\|\mathbb{X}\|_{\max}^D}{\varepsilon} + \left(\frac{s\cdot\|\mathbb{X}\|_{\max}^D}{\varepsilon}\right)^2 \sqrt{1-\frac{(\|\mathbb{V}\|-\tau)^2}{\|\mathbb{V}\|^2}} \Big/ \frac{\|\mathbb{V}\|-\tau}{\|\mathbb{V}\|}$, where $\|\mathbb{X}\|_{\max}$ denotes the maximal absolute coordinate of the points in $\mathbb{X}$ and where D is an additional input parameter of the algorithm that specifies the maximally allowed degree for the resulting polynomials in H.
In Section 4.5, we present a third algorithm, the so-called Border Basis (BB) ABM algorithm. It is built around the idea to use (ordinary) least squares to construct the almost vanishing polynomials that was originally proposed by C. Fassino in [29]. However, Fassino described an algorithm that computes an approximate Gröbner basis, which turned out to be a numerically challenging undertaking. In contrast, our BB ABM algorithm tries to compute an approximate border basis in form of sets G and $\mathcal{O}$. The algorithm takes as input a finite set of points $\mathbb{X}$, $\varepsilon \geq 0$, and a term ordering σ on $\mathbb{T}^n$. By construction, we have for each polynomial g_i in G the bound $\|\text{eval}_{\mathbb{X}}(g_i)\| \leq \varepsilon$. Furthermore, we were able to show in Theorem 4.5.3 that the polynomials in G form a δ-approximate border basis with $\delta = \frac{\varepsilon}{\zeta}(2\|\mathbb{X}\|_{\max} + \gamma\nu)$, where $\zeta = \frac{\varepsilon^{s-1}}{\sqrt{s}^{s-2}\Pi_{i=2}^{s}(\sqrt{i}\|\mathbb{X}\|_{\max}^{i-1})}$ and $\gamma = \frac{\sqrt{s}^{s-1}\Pi_{i=2}^{s}(\sqrt{i}\|\mathbb{X}\|_{\max}^{s+i-1})}{\varepsilon^{s-1}}$. Even though an upper bound in terms of the input parameters exists, it is unfortunately most of the time impractically large.
Finally, we compare the algorithms that we have developed to some other state of the art approaches in Section 4.7 that are concerned with the solution of similar problems. For this purpose, we briefly explain the approximate H-basis of algorithm of Sauer in Subsection 4.7.1 and the SOI and NBM of C. Fassiono in Subsections 4.7.2 and 4.7.3. We evaluate the quality of the computed results of the ABM algorithm, of the approximate H-basis algorithm, and of the SOI and NBM algorithm. In the case of the SOI and NBM algorithm, we also compare the computational performance, as there are implementations of the algorithms publicly available in the CoCoA library ([19]). The experimental data (see Tables 4.1 and 4.2) give strong evidence that the ABM algorithm significantly outperforms the SOI and NBM algorithm while delivering comparable or even superior results.

In Chapter 5 two novel solutions to the rational recovery problem, which is concerned with constructing an exact $\mathcal{O}$-border bases in the vicinity of a given approximate $\mathcal{O}$-border basis, are presented. More specifically this means that for a given approximate $\mathcal{O}$-border basis $G = \{g_1, ..., g_\nu\}$ we want to construct an exact $\mathcal{O}$-border basis $\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_\nu\}$ such that $\Sigma_{i=1}^{\nu}\|g_i - \tilde{g}_i\|_2$ is "possibly" small, where $\|\cdot\|$ is the norm of the coefficient vector of a polynomial. Even though it is possible to formulate this problem as a constrained optimisation problem which can be solved with global minimisation techniques, this procedure becomes already too time consuming for rather small values of n and ν. Therefore we do not require to find a $\tilde{G}$ such that $\Sigma_{i=1}^{\nu}\|g_i - \tilde{g}_i\|_2$ is minimal, we are rather satisfied if we obtain a value of $\Sigma_{i=1}^{\nu}\|g_i - \tilde{g}_i\|_2$ which is small enough for a particular application.
Before we are able to state our contributions to this subject, let us briefly mention a characterisation of border bases in terms of commuting matrices. Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$

be an order ideal, and let G be an $\mathcal{O}$-border basis for a 0-dimensional ideal I. It was originally shown by Mourrain in [25] that an $\mathcal{O}$-border basis can also be characterised via its associated (formal) multiplication matrices $A_1, ..., A_n \in \mathrm{Mat}_\mu(\mathbb{C})$ (see Definition 5.1.1). Theorem 5.1.3 states that an $\mathcal{O}$-border prebasis G is a border basis of $\langle G \rangle$ if $A_i A_j = A_j A_i$ for all $1 \leq i < j \leq n$. Furthermore, if $\langle G \rangle$ is a radical ideal it is well-known that the multiplication matrices can be simultaneously diagonalised (see Theorem 5.1.10). With the help of this simultaneous diagonalisation of $A_1, ..., A_n$ it is possible to obtain a numerical approximations of the roots of $\langle G \rangle$. This idea was originally brought up by Auzinger and Stetter in [48]. It can be used to solve the rational recovery problem if G is (essentially) an exact border basis that was computed in floating point arithmetic, i.e. G is a δ-approximate border basis where δ is in the order of the machine accuracy $\varepsilon_{machine}$ (see Definition 2.7.4). For this purpose we present the so-called Eigenvector algorithm (30) that is based upon the idea of simultaneous diagonalisation. Furthermore, we show by example that it produces inadequate solutions for larger values of δ.
Similarly, we show in Theorem 5.1.15 that approximate border bases can also be characterised via their associated multiplication matrices. Suppose that G is an $\mathcal{O}$-border prebasis, then G is a δ-approximate $\mathcal{O}$-border basis if we have $\|(A_j A_i - A_i A_j)\, e_k\| \leq \delta$ for all $1 \leq i < j \leq n$ and $1 \leq k \leq \mu$, where e_k is the k-th unit vector in $\mathbb{C}^\mu$. Consequently we have adapted the concept of simultaneous diagonalisation to approximate border bases. Our first major contribution to this subject is the idea to compute a simultaneous quasi-diagonalisation of the multiplication matrices associated with an approximate $\mathcal{O}$-border basis. As it is no longer possible to simultaneously diagonalise the involved matrices, we try to construct iteratively a basis transformation matrix $V \in \mathrm{GL}_\mu(\mathbb{C})$ such that $V^{-1} A_i V$ is close to a diagonal matrix for all $1 \leq i \leq n$. Please consider Subsection 5.3.1 for details. Via similar techniques as in the Eigenvector algorithm we can extract a set of points $\mathbb{X} = \{p_1, ..., p_\mu\}$ from the quasi-diagonal matrices $V^{-1} A_i V$ that can be used as input for the Buchberger-Möller algorithm for border bases. The resulting border basis is then transformed using the border basis transformation algorithm (19). In this way, we can construct a new exact border basis with respect to the given order ideal $\mathcal{O}$.
As the computation of a simultaneous quasi-diagonalisation for a set of general matrices is a non-standard subject in numerical linear algebra, we have investigated that problem in more detail in Section 5.3. Algorithms in the literature are scarce and most of the time they can only be applied to special cases such as Hermitian or normal matrices (compare [62]). Other algorithms, like the one proposed by Fu and Gao in [53], are only applicable to real matrices and tend to lack proper proofs of convergence. Therefore it was unfortunately not possible to just pick one algorithm from the literature and to apply it to our problem. Hence, we propose a new algorithm, based on the classical Jacobi eigenvalue algorithm, that is capable of computing a simultaneous quasi-diagonalisation of a set of general complex matrices. Furthermore, we also prove convergence of this method in Theorem 5.3.23. The central idea of the algorithm is to apply a sequence of unitary and non-unitary similarity transformations simultaneously to the matrices A_i such that after each iteration the common squared departure from normality (see Definition 5.2.11) is reduced via the non-unitary transformations and the common squared departure from diagonality (see Definition 5.3.7) of the matrices is reduced via the unitary transformations. If we accumulate the similarity transformations we obtain the matrices V and V^{-1}. Like Eberlein in [59] for the

case of one matrix, we apply a sequence of shear (non-unitary) and unitary rotation matrices. The exact definitions of those matrices can be found in 5.3.8 and 5.3.9. Both are determined by the row p and column q that they operate on, as well as by two real valued parameters. A major achievement of this thesis was to show via some lengthy computations how to choose those parameters in such a way that each iteration decreases the common squared departure from normality and from diagonality. The result concerning the shear rotation matrix is contained in Theorem 5.3.17, and the result concerning the unitary rotation matrix is contained in Theorem 5.3.22.
The simultaneous quasi-diagonalisation algorithm is not limited to cases where the matrices cannot be exactly simultaneously diagonalised. As it turns out, it also provides additional numerical stability where traditionally a diagonalise-one-then-diagonalise-the-others (DODO) approach would have been used. This method first diagonalises one matrix and then applies the same similarity transformations to the other matrices. However, if we use all matrices simultaneously to determine the similarity transformations, it is clear that we gain additional numerical stability compared to working on one matrix alone. Of course, our method is computationally more expensive than the DODO approach, therefore, a tradeoff between runtime and accuracy has to be made.
So, if we come back to the rational recovery problem, in the case that the involved multiplication matrices are exact or are computed in floating point arithmetic and therefore only almost exact, we can use the simultaneous diagonalisation algorithm as a stable and efficient means to compute an approximation of the zero set of the underlying ideal.
Finally, we give some numerical evidence which demonstrates that the presented algorithm is both fast and numerically stable in comparison with current state of the art simultaneous diagonalisation algorithms like the one of Fu and Gao ([53]).
The second solution to the rational recovery problem that we present in Section 5.4 is a heuristic that tries to find the local minima of a sum of squares expression, which we obtain directly from the polynomials in the approximate $\mathcal{O}$-border basis $G = \{g_1, ..., g_\nu\}$. The idea is rather straightforward and can be explained easily if we assume for a moment that $K = \mathbb{R}$. First, we form the sum of squares expression $S = \sum_{i=1}^{\nu} g_1^2$. Then, the local minima of S serve as candidates for the zero set of the exact $\mathcal{O}$-border basis that we again reconstruct with the help of the Buchberger-Möller algorithm and the border basis transformation algorithm. Even though S is a sum of squares expression, it is computationally a non-trivial task to find all local minima. For this purpose, we present a local optimisation approach that is specifically tailored to our situation and based on the Polak-Ribière conjugate gradient algorithm (34). A more general version of this idea also working for $K = \mathbb{C}$ is explained in Section 5.4. Finally, we present a few computations that show the potential of the heuristic method.

Some of the theoretical improvements that we achieve with the new algorithms were necessary prerequisites to allow new industrial applications. In Section 6.2, for instance, we present a new data driven approach to seismic imaging that is based on the extended ABM algorithm. In contrast to existing methods, it requires hardly any a priori assumptions about the subsurface structure. We explain the basic methodology and demonstrate it on a few synthetic examples. Even though the method is promising, it still requires the development of a surrounding frame-

work in order to be fully competitive with established techniques.

1.3 Outline of the Thesis

In Chapter 2 we try to give a rather comprehensive overview of all the mathematical concepts and results which are used for the development and analysis of the main results of this thesis.
After introducing some basic notations in Section 2.1, we continue with recalling some fundamental concepts and definitions from algebra in Section 2.2. Starting with Section 2.3, we refresh the knowledge of the reader about concepts of numerical linear algebra which are crucial for the development and analysis of the algorithms presented later on. This includes the definitions of vector and matrix norms together with their properties as well as properties of Hermitian matrices and some first basic results about the eigenvalues and eigenvectors of both general and Hermitian matrices.
The efficiency of an algorithm and its precise analysis are also crucial for industrial applications and applications in general that deal with large input datasets. For this reason we briefly recall the so-called Big O notation in Section 2.4 that allows us to measure the asymptotic complexity of algorithms.
In Section 2.5 we introduce the most common matrix factorisations that are used in numerical linear algebra, and which we will either use in the proofs of some theoretical properties or which we will use as an algorithmic component inside some algorithms like the singular value decomposition (SVD). We also present the not so well-known PLURQ decomposition which is closely related to the reduced row echelon form of a matrix and which we will use to update the kernel of a matrix once a new column is added. This allows us to improve the runtime of the Buchberger-Möller algorithm for border bases.
As a next step we recall in Section 2.6 the definition and the properties of the Moore-Penrose pseudoinverse of a matrix. It will turn out to be relevant to us as it generalises the concept of the inverse of a matrix to non-square matrices and is closely related to the solution of the homogeneous least squares problem. The numerical stability of algorithms plays a crucial role in industrial algorithms, therefore we want to put our analysis on a solid foundation and we introduce the relevant concepts from numerical linear algebra that relate both to the condition of a problem and to the accuracy of an algorithm in Section 2.7.
Next, we explain the concept of Householder reflectors and how they can be utilised to compute a QR decomposition of a general matrix in an efficient and stable way. As eigenvalues and eigenvectors and the question how they can be economically and stably computed are crucial for understanding and analysing the numerical properties of the ABM family of algorithms, we cover this topic extensively in Section 2.9. We present the basic version of the QR-algorithm that is based on the QR decomposition and which can be applied to general matrices as well as the Divide&Conquer eigenvalue algorithm that proves useful for Hermitian matrices. These details are important as some of the optimisations which we propose for the ABM family of algorithms require a deep understanding of the available techniques and in which context they can be applied.
In Section 2.10 we present the inhomogeneous and the homogeneous least squares problem. The

following section explains in detail how the inhomogeneous least squares problem can be solved and how it is related to the Moore-Penrose pseudoinverse. Finally, we present a well-known theorem about the conditioning of the inhomogeneous least squares problem in Section 2.12. This result will be used later in the analysis of the properties of the extended ABM algorithm. The introductory chapter is concluded by some elaborations about the solution of the homogeneous least squares problem and how it can be computed effectively. The algorithm that follows from the theoretical characterisation forms the core of the standard version of the ABM algorithm.

Chapter 3 explains the concepts behind exact border bases, their characterisation and properties in detail, as they are essential for understanding the generalisation to approximate border bases. Even though this also serves as preparation for Chapters 4, 5, and 6, which contain our main results, we have separated this part from Chapter 2 as border bases play a central role in this work such that they deserve their own chapter.
Section 3.1 starts with the basic definitions that are necessary prerequisites before we can finally introduce border bases. Next, we explain why border bases behave numerically more stable than, for instance, Gröbner bases and hence, are more suitable for practical applications. As we want to compute border bases of the vanishing ideal of finite point sets, we become more specific with respect to these concepts in Section 3.3. As the conclusion of this chapter, we present the Buchberger-Möller algorithm for border bases, which is an efficient polynomial time algorithm that computes a border basis for the vanishing ideal of a given finite point set. Additionally, we analyse the runtime of the algorithm in detail and explain how the PLURQ decomposition can be used to speed up the computation.

The beginning of Chapter 4 deals with the concept of approximate border bases. In Section 4.2 we present the AVI algorithm from [28] and we additionally discuss its runtime and some of its shortcomings. The ABM algorithm is introduced in Section 4.3. We prove finiteness of the algorithm and we also show how far away the computed result is at most from an exact border basis. Additionally, we elaborate on the numerical properties of the algorithm and analyse its runtime. In Subsection 4.3.1 it will be explained how to update the underlying matrix factorisations of the ABM algorithm to achieve better runtime and how the algorithm could be implemented using the BLAS and LAPACK libraries. This result can be seen as an analogue to using the PLURQ decomposition in the Buchberger-Möller algorithm to improve its runtime. We start the following section with a small example that demonstrates that the standard version of the AVI and ABM algorithm cannot be used directly if polynomials that "model" one specific quantity are desired. To overcome this issue, we introduce the extended ABM algorithm and also discuss its algebraic and numeric properties as well as its runtime. Its main advantage over the ABM algorithm for modelling is the direct control over the residual error in the model polynomials that we can establish. In Section 4.5 we present the BB ABM algorithm, a border basis variant of an algorithm which was originally proposed by C. Fassino in [29]. As a follow up, we explain how the mentioned algorithms can be adapted to produce results that are of higher practical relevance, for instance by cleaning the polynomials from terms which do not contribute in a significant way to the evaluation of the polynomial with respect to the given input data. Furthermore, we show how the ideas about sub-ideal border bases from [32] can be combined with the new algorithms developed in this thesis. In Section 4.7 we try to put our work into

context. For this purpose we briefly mention H-Bases which have also been applied successfully by Sauer to numerical problems in [41]. Additionally, we discuss their computation and how they differ conceptually from border bases. The chapter is concluded by a comparison with the Stable Order Ideal (SOI) and the Numerical Buchberger-Möller (NBM) algorithm (published in [34] and [35]), two alternative state of the art approaches.

The topic of Chapter 5 is the so-called rational recovery problem. The output of the ABM family of algorithms is essentially an approximate $\mathcal{O}$-border basis. However, a lot of concepts from algebra like the computation of syzygies cannot be transferred easily to approximate border bases. To be able to use the full "toolbox" of algebra it is therefore desirable to construct an exact border basis which is as close as possible, in terms of the differences of the coefficient vectors, to a given approximate border basis. For this purpose, we explain to the reader in Section 5.1 how exact and also approximate border bases can be characterised with the help of multiplication matrices. In case the multiplication matrices are almost exact, e.g. if they have been computed by a floating point implementation of the Buchberger-Möller algorithm, we present a known solution to the problem in the form of the Eigenvector algorithm in Section 5.2. We make clear how this translates to computing a simultaneous diagonalisation of the involved multiplication matrices. This is our starting point for Section 5.3, where we explain in detail how a simultaneous quasi-diagonalisation for a given set of matrices can be effectively computed. For this purpose, we present the Simultaneous Quasi-Diagonalisation algorithm (31), a variant of the classical Jacobi eigenvalue algorithm, and show that it is capable of computing a simultaneous quasi-diagonalisation for a given set of real or complex matrices. Via some examples we also demonstrate the workings of the algorithm. To conclude this chapter, we present in Section 5.4 a heuristic method which forms a sum of squares expression from the given approximate border bases and uses the local minima of this expression as candidates for the zeros of an exact border basis. We give numerical evidence in the form of example computations that show the great potential of this approach.

After laying out the theoretical foundations, possible applications of the presented algorithms within the (oil) industry are discussed in Chapter 6. This includes, but is not limited to, the traditional application of the AVI algorithm, the discovery of polynomial relations in noisy physical data and the construction of polynomial models for one specific physical quantity. In Section 6.2 we detail the challenges of seismic imaging and explain the underlying physical principles of wave propagation. Using this knowledge, we propose an application of the extended ABM algorithm to seismic data that is able to reconstruct an image of the subsurface. Additionally, we describe in the next section how the ABM algorithm can be used for the modelling of unconventional oil bodies via algebraic surfaces. The Chapter is concluded by an application of the shear rotation algorithm that is used to compute the zero set of a border basis in a numerically stable way. Based on example computations, we present some evidence that the shear rotation algorithm is numerically superior to the Eigenvector algorithm implemented with a state of the art version of LAPACK.

In Chapter 7 we give a short outlook on additional improvements that could be added to the algorithms. More specifically, we sketch how the ABM algorithm could be modified to provide a more practical upper bound for the deviation from an exact border basis of the compute

approximate one. Furthermore, we hint at some potential new applications in sparse signal approximation that are currently being explored in the Geometric Exploration Project, a joint research effort between Shell International Exploration and Production and the University of Passau.

Finally, an overview of the functions, which were implemented by the author in C++ in the ApCoCoA [20] library, and of their syntax are given in the Appendix. Small examples are provided to illustrate their usage. Additionally, we explain in Appendix 8.2 how to read the pseudo code which is used throughout this thesis.

2 Mathematical and Algorithmic Foundation

Contents

In the following chapters we will make use of the basic notation which we now introduce. Additionally, some information about the pseudo code which we utilise to write down algorithms can be found in Appendix 8.2.

2.1 Notation

$\mathbb{N}$ denotes the set of natural numbers beginning with 1.

$\mathbb{N}_0$ denotes the set of natural numbers beginning with 0.

$\mathbb{Z}$ denotes the ring of integers.

$\mathbb{Q}$ denotes the field of rational numbers.

$\mathbb{R}$ denotes the field of real numbers.

$\mathbb{R}^+$ denotes all real numbers greater than 0.

$\mathbb{R}_0^+$ denotes all real numbers greater than or equal to 0.

$\mathbb{C}$ denotes the field of complex numbers. For a given complex number c we denote by $\Re(c)$ the real and by $\Im(c)$ the imaginary part of c.

Let $x \in \mathbb{R}$. By $\lfloor x \rfloor$ we denote the largest element in $\mathbb{Z}$ which does not exceed x. Similarly, we denote by $\lceil x \rceil$ the smallest element in $\mathbb{Z}$ which is not less than x.

$\mathrm{Mat}_{m,n}(K)$ denotes all matrices with m rows and n columns over a field K.

$\mathrm{Mat}_m(K)$ denotes all square matrices with m rows and columns over a field K.

$\mathrm{GL}_m(K)$ denotes all invertible matrices over a field K.

If no ambiguity can arise and the dimensions m and n are known from the context we will slightly abuse notation and denote the zero vector in K^m and the zero matrix in $\mathrm{Mat}_{m,n}(K)$ by 0. Otherwise we will use the notation 0_m respectively $0_{m,n}$.

If $A \in \mathrm{Mat}_{m,n}(K)$, then $A_{o:p,q:r} \in \mathrm{Mat}_{p-o+1,r-q+1}(K)$ with $1 \leq o \leq p \leq m$ and $1 \leq q \leq r \leq n$ denotes the submatrix of A which includes rows o to p and columns q to r of A. If $o = p$ and/or $q = r$ we abbreviate our notation by writing $A_{o,q:r}$, $A_{o:p,q}$ or $A_{o,q}$, respectively. Furthermore, we denote by $a_{ij} \in K$ with $1 \leq i \leq m$ and $1 \leq j \leq n$ the entry in the i-th row and j-th column of A.

$I_m(K)$ denotes the identity matrix with m rows and columns over a field K. If K is clear from the context we may omit K and write I_m. Furthermore, we will denote a rectangular matrix that has $1 \in K$ as entries in $A_{1,1}, A_{2,2}, ..., A_{\min(m,n),\min(m,n)}$ and $0 \in K$ in all other locations by $I_{m,n}(K)$ or $I_{m,n}$.

For a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ we let A^{tr} be the transposed of A, we let $\bar{A}$ be the complex conjugate of A, and we let A^* be the complex conjugate transposed of A. The matrix A^* is called the **adjoint matrix** of A. For a vector $v \in \mathbb{C}^m$ we denote by $\bar{v}$ the complex conjugate of v.

$A^+ \in \mathrm{Mat}_{n,m}(\mathbb{C})$ denotes the Moore-Penrose pseudoinverse of a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$.

A matrix $A \in \mathrm{Mat}_{1,n}(K)$ is called **row vector** and a matrix $A \in \mathrm{Mat}_{m,1}(K)$ is called **column vector**. Just like ordinary vectors they will be denoted by lower-case letters. If no ambiguity can arise, we silently identify a tuple $v \in K^m$ with the corresponding column vector in $\mathrm{Mat}_{m,1}(K)$.

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix. Then we denote by $\lambda_k(A)$ the k-th eigenvalue of A assuming that the m eigenvalues of A are ordered descendingly with respect to their values, meaning that $\lambda_1(A) \geq ... \geq \lambda_m(A)$.

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be a complex matrix. Then we denote by $\sigma_k(A)$ the k-th singular value of A assuming that the $\min(m,n)$ singular values of A are ordered descendingly with respect to their values, meaning that $\sigma_1(A) \geq ... \geq \sigma_{\min(m,n)}(A)$.

2.2 Basic Definitions from Algebra

In the following we recall some standard terminology from [45].

Definition 2.2.1. [Monoid]

Let S be a set, let $\cdot$ be a binary operation $S \times S \to S$ and let $1_S \in S$. The triple $(S, \cdot, 1_S)$ is called a **monoid** if the following conditions are satisfied:

1. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in S$,
2. $a \cdot 1_S = 1_S \cdot a = a$ for all $a \in S$.

Suppose that $\bar{1}_S \in S$ is another identity element. Then we have $1_S = 1_S \cdot \bar{1}_S = \bar{1}_S$. So as 1_S is uniquely determined for S, it is custom to omit 1_S and to write $(S, \cdot)$. Additionally, instead of 1_S we just write 1.
A monoid is called **commutative** if $a \cdot b = b \cdot a$ for all $a, b \in S$.

Definition 2.2.2. [Group]

Let $(S, \cdot)$ be a monoid. We call $(S, \cdot)$ a group if each element in S is invertible with respect to $\cdot$, which means that for all $a \in S$ there exists $a^{-1} \in S$ such that

$$a \cdot a^{-1} = a^{-1} \cdot a = 1.$$

A group is called **commutative** if $(S, \cdot)$ is a commutative monoid.

Definition 2.2.3. [Ring]

Let R be a set which is equipped with two binary operations $+ : R \times R \to R$ (addition) and $\cdot : R \times R \to R$ (multiplication). The triple $(R, +, \cdot)$ is called a **ring** if the following conditions are satisfied:

1. $(R, +)$ is a commutative group with identity element $0 \in R$.
2. $(R, \cdot)$ is a monoid with identity element $1 \in R$.
3. $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = ac + bc$ for all $a, b, c \in R$.

If $+$ and $\cdot$ are clear from the context we may omit them and just denote the ring $(R, +, \cdot)$ by R.

In case $(R, \cdot)$ is a commutative monoid we call R a **commutative ring**.

Definition 2.2.4. [Field]

A **field** K is a commutative ring $(K, +, \cdot)$ for which additionally $(K \setminus \{0\}, \cdot)$ is a group.

Example 2.2.5. The rational, real, and complex numbers together with the usual addition and multiplication are fields.

From now on let K be a field and let $P = K[x_1, ..., x_n]$ be the polynomial ring in n indeterminates. The elements of P are called **polynomials**, in particular, the element $0 \in P$ is called the **zero polynomial**.

Definition 2.2.6. [Term]

A polynomial $t \in P$ of the form $t = \prod_{i=1}^{n} x_i^{e_i}$ with $e_i \in \mathbb{N}_0$ is called a **term**. In particular this means that also 1 is a term. By $\mathbb{T}(x_1, ..., x_n)$ or $\mathbb{T}^n$ we denote the **set of all terms** in P.

Remark 2.2.7. The set $\mathbb{T}^n$ together with the usual multiplication $\cdot$ forms a commutative monoid with identity element $1 = x_1^0 ... x_n^0$.

Definition 2.2.8. [Log]

The map $\log : \mathbb{T}^n \to \mathbb{N}_0^n$ given by $x_1^{e_1} ... x_n^{e_n} \mapsto (e_1, ..., e_n)$ is called the **logarithm**.

Definition 2.2.9. [Monomial]

A **monomial** m is a term t multiplied by an element $c \in K \setminus \{0\}$, such that $m = ct$. The element c is called the **coefficient** of the monomial m. Naturally, every term is a monomial with coefficient 1.

Definition 2.2.10. [Degree of a monomial]

We denote the **degree of a monomial** $m = c \prod_{i=1}^{n} x_i^{e_i}$ with $c \in K$ by $\deg(m)$. It is defined as follows:

$$\deg(m) = \sum_{i=1}^{n} e_i.$$

Definition 2.2.11. [Support of a polynomial]

The set of all terms occurring in a polynomial $p \neq 0$ is denoted by $\operatorname{supp}(p)$ and is called the **support** of the polynomial p. If $p = 0$ we let $\operatorname{supp}(p) = \emptyset$.

The **degree of a polynomial** $p \neq 0$ is denoted by $\deg(p)$ and defined as the maximum of $\{\deg(t) \,|\, t \in \operatorname{supp}(p)\}$.

By $\mathbb{T}_k(x_1, ..., x_n)$ or $\mathbb{T}_k^n$ we denote the **set of all terms of degree k** in P.

Definition 2.2.12. [Term ordering]

Let us consider the monoid $(\mathbb{T}^n, \cdot)$. A complete relation σ on $\mathbb{T}^n$ is called a **term ordering on** $\mathbb{T}^n$ if the following conditions are satisfied for all $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{T}^n$. For convenience we shall write $\gamma_1 \geq_\sigma \gamma_2$ if $(\gamma_1, \gamma_2) \in \sigma$.

1. $\gamma_1 \geq_\sigma \gamma_1$.
2. $\gamma_1 \geq_\sigma \gamma_2$ and $\gamma_2 \geq_\sigma \gamma_1$ imply $\gamma_1 = \gamma_2$.
3. $\gamma_1 \geq_\sigma \gamma_2$ and $\gamma_2 \geq_\sigma \gamma_3$ imply $\gamma_1 \geq_\sigma \gamma_3$.
4. $\gamma_1 \geq_\sigma \gamma_2$ implies $\gamma_1 \cdot \gamma_3 \geq_\sigma \gamma_2 \cdot \gamma_3$.
5. $\gamma_1 \geq_\sigma 1$.

If additionally $\gamma_1 \geq_\sigma \gamma_2$ implies that $\deg(\gamma_1) \geq \deg(\gamma_2)$, we say that σ is a **degree compatible term ordering**.

Example 2.2.13. [DegRevLex term ordering]

Let $t_1, t_2 \in \mathbb{T}^n$. We write $t_1 \geq_{\text{DegRevLex}} t_2$ if $\deg(t_1) > \deg(t_2)$, or if $\deg(t_1) = \deg(t_2)$ and the last non-zero component of $\log(t_1) - \log(t_2)$ is negative, or if $t_1 = t_2$. This ordering is called the **degree-reverse-lexicographic term ordering** and is denoted by `DegRevLex`. It can be easily verified that `DegRevLex` is a degree compatible term ordering.

Definition 2.2.14. [Ideal]

Let $I \subseteq P$ and let $a, b \in P$. We call I an **ideal** if the following conditions are satisfied.

1. $0 \in I$.
2. If $a, b \in I$, then $(a + b) \in I$.
3. If $a \in I$, then $ab \in I$.

2.3 Basic Definitions from (Numerical) Linear Algebra

At the beginning of this section we recall some basic definitions from linear algebra. The concept of vector and matrix norms will turn out to be useful when we investigate the numerical properties of algorithms which deal with floating point vectors and matrices. Then we discuss the concept of unitary transformations, as these play an important role in computing certain matrix factorisations which we will actively use in the Chapters 4 and 5. This introduction ends with some definitions from eigenvalue theory which will allow us to study the homogeneous least squares problem in more detail. All results presented in this chapter are well-known unless explicitly stated otherwise. The base field is (in most cases) $\mathbb{C}$ as it will be imperative later on that we work over an algebraically closed field.

Definition 2.3.1. [Inner product]

The **inner product** of two vectors $a, b \in \mathbb{C}^m$ is defined as

$$\langle a, b \rangle = \sum_{i=1}^{m} \overline{a}_i b_i,$$

where $\bar{a}_i$ is the complex conjugate of a_i. Whenever there is no danger of confusion we abbreviate $\langle a, b \rangle$ by writing ab.

Remark 2.3.2. Let $a, b \in \mathbb{C}^m$ be vectors and let $A, B \in \text{Mat}_{m,1}(\mathbb{C})$ be the column vectors, which we obtain if we interpret a and b as column vectors. Then we note that $\langle a, b \rangle = A^* \cdot B$ where $\cdot$ is the standard matrix multiplication.

Definition 2.3.3. [Vector norm]

A map $\|\cdot\| : \mathbb{C}^m \to \mathbb{R}$ is called a **vector norm** if it satisfies the following three conditions. For all $a, b \in \mathbb{C}^m$ and $\alpha \in \mathbb{C}$,

$$\begin{aligned} \|a\| &\geq 0, \text{ and } \|a\| = 0 \text{ if and only if } a = 0_m, \\ \|a + b\| &\leq \|a\| + \|b\|, \\ \|\alpha a\| &= |\alpha| \, \|a\|. \end{aligned}$$

Definition 2.3.4. [p-norm]

Let $a = (a_1, ..., a_m) \in \mathbb{C}^m$ and $p \geq 1$. The non-negative real number

$$\|a\|_p = \sqrt[p]{\sum_{i=1}^{m} |a_i|^p}$$

is called the **p-norm** of a. The 2-norm

$$\|a\|_2 = \sqrt{\langle a, a \rangle} = \sqrt{\sum_{i=1}^{m} |a_i|^2}$$

is also known as the **Euclidean norm**. If there is no danger of confusion, we abbreviate the 2-norm by $\|\cdot\|$.

Proposition 2.3.5. *For every $p \geq 1$, the p-norm $\|\cdot\|_p : \mathbb{C}^m \to \mathbb{R}_0^+$ is a vector norm.*

Proof. See, for example, [3, Theorem 15.50]. □

Definition 2.3.6. [Maximum norm]

Let $a = (a_1, ..., a_m) \in \mathbb{C}^m$. We call the non-negative real number

$$\|a\|_\infty = \max\{|a_1|, ..., |a_m|\}$$

the **maximum norm** of a.

Proposition 2.3.7. *The maximum norm $\|\cdot\|_\infty : \mathbb{C}^m \to \mathbb{R}_0^+$ is a vector norm.*

Proof. A proof is also contained in [3, Theorem 15.50]. □

Theorem 2.3.8 (Hölder inequality)**.** *The inner product can be bounded via the p-norms. If p and q satisfy the condition $\frac{1}{p} + \frac{1}{q} = 1$, with $1 \leq p, q \leq \infty$, then*

$$|\langle a, b \rangle| \leq \|a\|_p \|b\|_q$$

holds for all $a, b \in \mathbb{C}^m$.

Proof. A proof can be found in [3, Theorem 15.48]. □

In the special case $p = q = 2$, the Hölder inequality is also called the **Cauchy-Schwarz inequality.**

Definition 2.3.9. [Matrix norm]

A map $\|\cdot\| : \mathrm{Mat}_{m,n}(\mathbb{C}) \to \mathbb{R}$ is called a **matrix norm** if it satisfies the following three conditions. For all $A, B \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $\alpha \in \mathbb{C}$,

$$\begin{aligned} \|A\| &\geq 0, \text{ and } \|A\| = 0 \text{ if and only if } A = 0_{m,n}, \\ \|A + B\| &\leq \|A\| + \|B\|, \\ \|\alpha A\| &= |\alpha| \|A\|. \end{aligned}$$

A matrix norm is thus a vector norm on $\mathrm{Mat}_{m,n}(\mathbb{C})$.

Definition 2.3.10. [Induced matrix norm]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, $1 \leq p \leq \infty$ and let us, by a slight abuse of notation, denote by $\|\cdot\|_p$ both the p-vector norm on $\mathbb{C}^n$ and $\mathbb{C}^m$. In this case the **induced matrix norm** is defined as

$$\|A\|_p = \max\left\{ \|Ax\|_p \,\middle|\, x \in \mathbb{C}^n \text{ with } \|x\|_p = 1 \right\}.$$

Proposition 2.3.11. *For every $1 \leq p \leq \infty$, the induced matrix norm $\|\cdot\|_p : \mathrm{Mat}_{m,n}(\mathbb{C}) \to \mathbb{R}_0^+$ is a matrix norm.*

Proof. See [6, Section 2.3.1]. □

Definition 2.3.12. [Consistent matrix norm]

A matrix norm $\|\cdot\|_{(mn)}$ on $\mathrm{Mat}_{m,n}(\mathbb{C})$ is called **consistent** with a vector norm $\|\cdot\|_{(m)}$ on $\mathbb{C}^m$ and a vector norm $\|\cdot\|_{(n)}$ on $\mathbb{C}^n$, if for all $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and for all $x \in \mathbb{C}^n$ the inequality

$$\|Ax\|_{(m)} \leq \|A\|_{(mn)} \|x\|_{(n)}$$

holds.

Proposition 2.3.13 (Properties of the induced matrix norm)**.** *Let $1 \leq p \leq \infty$ and let $\|\cdot\|_p$ be the induced matrix norm.*

1. *The induced matrix norm $\|\cdot\|_p$ is consistent with the inducing p-vector norms, which means that for all $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, $B \in \mathrm{Mat}_{n,k}(\mathbb{C})$, and $x \in \mathbb{C}^k$, the inequalities*

   $$\|ABx\|_p \leq \|A\|_p \|Bx\|_p \leq \|A\|_p \|B\|_p \|x\|_p$$

   hold.

2. *Additionally, for all $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, $B \in \mathrm{Mat}_{n,k}(\mathbb{C})$ the relation*

   $$\|AB\|_p \leq \|A\|_p \|B\|_p$$

   is satisfied.

Proof. To prove the first claim let us consider the case that $Bx = 0_n$. In this situation

$$\|ABx\|_p = 0 \leq \|A\|_p 0 \leq \|A\|_p \|B\|_p \|x\|_p$$

holds. So let us now consider the case where $Bx \neq 0_n$. As $\|Bx\|_p \in \mathbb{R}^+$ we can now write

$$\|ABx\|_p = \left\| A \frac{Bx}{\|Bx\|_p} \right\|_p \|Bx\|_p .$$

From the definition of the induced matrix norm it follows that

$$\left\| A \frac{Bx}{\|Bx\|_p} \right\|_p \|Bx\|_p \leq \|A\|_p \|Bx\|_p .$$

Using the same arguments we obtain

$$\|Bx\|_p = \left\| B \frac{x}{\|x\|_p} \right\|_p \|x\|_p \leq \|B\|_p \|x\|_p .$$

Combining the above inequalities, we get

$$\|ABx\|_p \leq \|A\|_p \|Bx\|_p \leq \|A\|_p \|B\|_p \|x\|_p .$$

In order to prove the second claim, we use the first claim and write

$$\begin{aligned} \|AB\|_p &= \max \left\{ \|ABx\|_p \,\middle|\, x \in \mathbb{C}^n \text{ with } \|x\|_p = 1 \right\} \\ &\leq \max \left\{ \|A\|_p \|B\|_p \|x\|_p \,\middle|\, x \in \mathbb{C}^n \text{ with } \|x\|_p = 1 \right\} \\ &= \|A\|_p \|B\|_p . \end{aligned}$$

□

Proposition 2.3.14. *Let $1 \leq p, q \leq \infty$ and let $\|\cdot\|_p$ and $\|\cdot\|_q$ be the corresponding induced matrix norms. Then $\|A\|_p \leq \|A\|_q$ for all $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ if and only if $\|A\|_p = \|A\|_q$ for all $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$.*

Proof. See [9, Corollary 5.6.25]. □

This proposition tells us that no induced matrix norm can be uniformly bounded by a different induced matrix norm.

Definition 2.3.15. [Frobenius matrix norm]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. Then

$$\|A\|_F = \|A\|_E = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

is called the **Frobenius** (or **Hilbert-Schmidt**) **norm** of A.

Proposition 2.3.16. *The Frobenius norm $\|\cdot\|_F : \mathrm{Mat}_{m,n}(\mathbb{C}) \to \mathbb{R}_0^+$ is a matrix norm.*

Proof. Compare again [6, Section 2.3.1]. □

Proposition 2.3.17. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $B \in \mathrm{Mat}_{n,l}(\mathbb{C})$. Then the inequality*

$$\|AB\|_F \leq \|A\|_F \|B\|_F$$

holds.

Proof. To prove the claim let us have a closer look at $\|AB\|_F^2$. With the help of the Cauchy-Schwarz inequality we compute

$$\begin{aligned}
\|AB\|_F^2 &= \sum_{i=1}^{m}\sum_{k=1}^{l}\left|\sum_{j=1}^{n} a_{ij}b_{jk}\right|^2 = \sum_{i=1}^{m}\sum_{k=1}^{l}\left|A_{i,1:n}B_{1:n,k}\right|^2 \\
&= \sum_{i=1}^{m}\sum_{k=1}^{l}\left|\left\langle A_{i,1:n}^*, B_{1:n,k}\right\rangle\right|^2 \leq \sum_{i=1}^{m}\sum_{k=1}^{l}\left\|A_{i,1:n}^*\right\|_2^2\left\|B_{1:n,k}\right\|_2^2 \\
&= \left(\sum_{i=1}^{m}\left\|A_{i,1:n}^*\right\|_2^2\right)\left(\sum_{k=1}^{l}\left\|B_{1:n,k}\right\|_2^2\right) = \|A\|_F^2\,\|B\|_F^2\,.
\end{aligned}$$

Taking the square roots on both sides concludes the proof. □

Definition 2.3.18. [Max matrix norm]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let us denote by a_{ij} with $1 \leq i \leq m$ and $1 \leq j \leq n$ the entries of A. Then

$$\|A\|_{\max} = \max_{i,j}|a_{ij}|$$

is called the **max matrix norm** of A.

Proposition 2.3.19. *The max matrix norm* $\|\cdot\|_{\max} : \mathrm{Mat}_{m,n}(\mathbb{C}) \to \mathbb{R}_0^+$ *is a matrix norm.*

Proof. See [7, page 21]. □

Proposition 2.3.20. *Let* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. *Then the inequality*

$$\|A\|_2 \leq \sqrt{mn}\,\|A\|_{\max}$$

holds.

Proof. Using the definition of the induced matrix norm 2.3.10 we obtain

$$\begin{aligned}
\|A\|_2 &= \max\left\{\|Ax\|_2 \,|\, x \in \mathbb{C}^n \text{ with } \|x\|_2 = 1\right\} \\
&= \max_{\|x\|_2=1}\left\{\sqrt{\left|\sum_{i=1}^{n} a_{1i}x_i\right|^2 + \ldots + \left|\sum_{i=1}^{n} a_{mi}x_i\right|^2}\right\} \\
&\leq \max_{\|x\|_2=1}\left\{\sqrt{\sum_{i=1}^{n}|a_{1i}x_i|^2 + \ldots + \sum_{i=1}^{n}|a_{mi}x_i|^2}\right\} \\
&\leq \sqrt{\sum_{i=1}^{n}|a_{1i}|^2 + \ldots + \sum_{i=1}^{n}|a_{mi}|^2} \leq \sqrt{mn\max_{ij}|a_{ij}|^2} \\
&= \sqrt{mn}\max_{ij}|a_{ij}|\,.
\end{aligned}$$

□

Definition 2.3.21. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let us denote by $\bar{A} \in \mathrm{Mat}_{m,n}(\mathbb{C})$ the complex conjugate of A. We call the matrix $\bar{A}^{\mathrm{tr}}$ the **adjoint matrix** of A and denote it by A^*.

Remark 2.3.22. [Basic properties of the adjoint matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let A^* be its adjoint matrix. Then the equations $A^{**} = A$ and $(AB)^* = B^*A^*$ hold.

Proof. The first equation is trivially true. For the second one we have

$$(AB)^* = \overline{AB}^{\mathrm{tr}} = \overline{(AB)^{\mathrm{tr}}} = \overline{B^{\mathrm{tr}}A^{\mathrm{tr}}} = B^*A^*.$$

□

Definition 2.3.23. [Hermitian matrix]

A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **Hermitian** if $A = A^*$ holds.

Proposition 2.3.24. *For every matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, the matrices A^*A and AA^* are Hermitian matrices.*

Proof. First we observe that $A^*A \in \mathrm{Mat}_n(\mathbb{C})$ and $AA^* \in \mathrm{Mat}_m(\mathbb{C})$ are both square matrices. We verify the claim by computing

$$(A^*A)^* = A^*A^{**} = A^*A$$

and

$$(AA^*)^* = A^{**}A^* = AA^*.$$

□

Definition 2.3.25. [Normal matrix]

A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **normal** if $A^*A = AA^*$ holds.

Example 2.3.26. All Hermitian and real symmetric matrices are normal. First we observe that real symmetric matrices are Hermitian. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix. From the definition of Hermitian matrices it follows immediately that $A^*A = AA = AA^*$. So we can conclude that A is normal.

Remark 2.3.27. The set of Hermitian matrices over $\mathbb{C}$ is a proper subset of the set of normal matrices over $\mathbb{C}$, as the matrix

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

is normal but not Hermitian.

Proposition 2.3.28. *A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is normal and (upper or lower) triangular if and only if A is a diagonal matrix.*

Proof. Let $A \in \mathrm{Mat}_m(\mathbb{C})$. In order to avoid notational ambiguities, let us denote the entries of A by $a_{i,j}$ (instead of a_{ij}) with $1 \leq i \leq m$ and $1 \leq j \leq m$. If A is a diagonal matrix, then A is obviously triangular and normal. To show the non-trivial implication "$\Rightarrow$" let us assume w.l.o.g. that A is normal and upper triangular. Note that for a lower triangular matrix essentially the same arguments can be used, as only the indices have to be adapted. Now let us consider the entries on the diagonal of $M = A^*A - AA^*$, the commutator of A, which are of the form $m_{i,i} = \sum_{j=1}^{i} |a_{j,i}|^2 - \sum_{j=i}^{m} |a_{i,j}|^2 = \sum_{j=1}^{i-1} |a_{j,i}|^2 - \sum_{j=i+1}^{m} |a_{i,j}|^2$. We start with $m_{1,1} = -\sum_{j=2}^{m} |a_{1,j}|^2$. As A is normal $m_{i,i}$ needs to be zero, and consequently $\sum_{j=2}^{m} |a_{1,j}|^2 = 0$ must hold. This equation can only be satisfied if all $a_{1,j}$ with $2 \leq j \leq m$ are zero. For $m_{2,2}$ we obtain $m_{2,2} = \sum_{j=1}^{2} |a_{j,2}|^2 - \sum_{j=2}^{m} |a_{2,j}|^2 = |a_{1,2}|^2 - \sum_{j=3}^{m} |a_{2,j}|^2$. From our previous observation we know that $a_{1,2} = 0$, which means that we can simplify the last equation to $m_{2,2} = -\sum_{j=3}^{m} |a_{2,j}|^2$. So we have shown that also $a_{2,j}$ with $3 \leq j \leq m$ has to be zero. If we reuse this argument consecutively for every $m_{i,i}$ with $i \in \{3, ..., m\}$ we can make use of the knowledge that all entries $a_{k,j}$ with $1 \leq k < i$ and $k \leq j \leq m$ are zero. We thus obtain $m_{i,i} = -\sum_{j=i+1}^{m} a_{i,j}\bar{a}_{i,j} = -\sum_{j=i+1}^{m} |a_{i,j}|^2$. So we can conclude in the i-th step that all $a_{i,j}$ with $i < j \leq m$ have to be zero as well. This means that all elements above the diagonal of A have to be zero. So A is diagonal. □

Definition 2.3.29. [Orthogonal sets]

Two sets of vectors $S_1 \subseteq \mathbb{C}^m$ and $S_2 \subseteq \mathbb{C}^m$ are called **orthogonal**, and we write $S_1 \perp S_2$, if the inner product $\langle s_1, s_2 \rangle$ of every $s_1 \in S_1$ with every $s_2 \in S_2$ is zero.

Definition 2.3.30. [Unitary matrix]

A matrix $U \in \mathrm{Mat}_m(\mathbb{C})$ is called **unitary** (or **orthogonal** if $U \in \mathrm{Mat}_m(\mathbb{R})$) if

$$U^*U = UU^* = I_m,$$

where $I_m \in \mathrm{Mat}_m(\mathbb{C})$ is the identity matrix.

Remark 2.3.31. Note that the row vectors of unitary matrices are pairwise orthogonal and the column vectors are pairwise orthogonal as well.

For the convenience of the reader, we collect some well-known properties of unitary matrices in the following propositions.

Proposition 2.3.32 (Product of unitary matrices)**.** *The product of unitary matrices is unitary.*

Proof. Let $U_1, U_2 \in \mathrm{Mat}_m(\mathbb{C})$ be unitary matrices. Then we calculate

$$U_1U_2\,(U_1U_2)^* = U_1U_2U_2^*U_1^* = I_m$$

and conclude that the claim is true. □

A special property of unitary matrices is that they preserve the Euclidean norm of a vector under matrix-vector multiplication.

Proposition 2.3.33 (Preservation of length under unitary multiplication)**.**

Let $Q \in \mathrm{Mat}_m(\mathbb{C})$ *be a unitary matrix and* $x \in \mathbb{C}^m$*. Then* $\|Qx\|_2 = \|x\|_2$*.*

Proof. The claim follows from

$$\|Qx\|_2 = \sqrt{(Qx)^* Qx} = \sqrt{x^* Q^* Q x} = \sqrt{x^* x} = \|x\|_2 .$$

□

Proposition 2.3.34. *Let* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ *be an arbitrary matrix, and let* $U \in \mathrm{Mat}_m(\mathbb{C})$ *and* $V \in \mathrm{Mat}_n(\mathbb{C})$ *be unitary matrices. Then the matrix norm induced by the Euclidean norm and the Frobenius norm are invariant under unitary transformations. In other words, the equations*

$$\|UAV\|_2 = \|A\|_2 \quad \textit{and} \quad \|UAV\|_F = \|A\|_F$$

hold.

Proof. See [6, page 70]. □

Definition 2.3.35. [Kernel and image of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. By $\ker(A)$ we denote the set

$$\ker(A) = \{x \in \mathbb{C}^n \,|\, Ax = 0_m\},$$

and by $\mathrm{im}(A)$ we denote the set

$$\mathrm{im}(A) = \{Ax \,|\, x \in \mathbb{C}^n\}.$$

Then $\ker(A)$ is called the **kernel** (or **null space**) of A, and $\mathrm{im}(A)$ is called the **image** (or **range**) of A. If $\ker(A) = \{0_n\}$ we say that the kernel of A is **trivial**.

Definition 2.3.36. [Projector]

A matrix $P \in \mathrm{Mat}_m(\mathbb{C})$ is called a **projector** if the condition $P^2 = P$ holds.

Proposition 2.3.37 (Complementary projector)**.** *If* $P \in \mathrm{Mat}_m(\mathbb{C})$ *is a projector, then the matrix* $I_m - P$ *is also a projector and the following properties hold:*

$$\mathrm{im}(I_m - P) = \ker(P)$$

and

$$\mathrm{im}(P) = \ker(I_m - P).$$

Proof. First of all, we prove that $(I_m - P)$ is also a projector. This follows from $(I_m - P)^2 = I_m^2 - 2I_m P + P^2 = I_m - 2P + P = I_m - P$.

In order to show the first equality, we start by proving $\mathrm{im}\,(I_m - P) \subseteq \ker(P)$. Let $v \in \mathbb{C}^m$. Then the inclusion follows from

$$P\left((I_m - P)\,v\right) = P\,(v - Pv) = Pv - P^2 v = Pv - Pv = 0_m.$$

Now we show that $\mathrm{im}\,(I_m - P) \supseteq \ker(P)$ also holds. This is true, as for every $v \in \ker(P)$ we have $(I_m - P)\,v = v$. By writing $P = I_m - (I_m - P)$ we can derive the second claim. □

Proposition 2.3.38. *Let $P \in \mathrm{Mat}_m(\mathbb{C})$ be a projector. Then*

$$\mathrm{im}\,(P) \cap \ker(P) = \{0_m\}.$$

Proof. Using Proposition 2.3.37 we observe that $\mathrm{im}\,(P) \cap \ker(P) = \ker(I_m - P) \cap \ker(P)$. So, for every v which is contained in both kernels, we have $(I_m - P)\,v = 0_m$ and $Pv = 0_m$. Hence, $0_m = (I_m - P)\,v = v - Pv = v$. □

This means that every projector divides $\mathbb{C}^m$ in two linear subspaces S_1 and S_2, such that $S_1 \cap S_2 = \{0_m\}$ and $S_1 \oplus S_2 = \mathbb{C}^m$. So we can express every vector $v \in \mathbb{C}^m$ in a unique way as a sum of an element $v_1 \in S_1$ and $v_2 \in S_2$, such that $v = v_1 + v_2$.

Definition 2.3.39. An **orthogonal projector** $P \in \mathrm{Mat}_m(\mathbb{C})$ is a projector for which the following additional property holds:

$$\mathrm{im}\,(P) \perp \ker(P).$$

Proposition 2.3.40. *If a projector $P \in \mathrm{Mat}_m(\mathbb{C})$ is Hermitian, which means that $P = P^*$, then P is an orthogonal projector.*

Proof. Let $P \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian projector. By Proposition 2.3.37 we know that $\ker(P) = \mathrm{im}\,(I_m - P)$. First we note that the condition that $\mathrm{im}\,(P) \perp \ker(P)$ is equivalent to $\langle Pv, (I_m - P)\,w\rangle = 0$ for all $v, w \in \mathbb{C}^m$. So let $v, w \in \mathbb{C}^m$ be arbitrary vectors. We verify the claim by computing

$$\langle Pv, (I_m - P)\,w\rangle = (Pv)^*\,(I_m - P)\,w = v^* P^*\,(I_m - P)\,w = v^*\left(P - P^2\right)w = 0.$$

□

Theorem 2.3.41. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, and let $Q \in \mathrm{Mat}_{m,k}(\mathbb{C})$, with $1 \leq k \leq n$, such that the columns of Q form an orthonormal basis of $\mathrm{im}\,(A)$. Then*

$$P_A = QQ^* \in \mathrm{Mat}_m(\mathbb{C})$$

is an orthogonal projector onto $\mathrm{im}\,(A)$, which means that P_A is an orthogonal projector and $\mathrm{im}\,(P_A) = \mathrm{im}\,(A)$.

Proof. First we show that P_A is a projector. For this purpose we compute

$$P_A^2 = QQ^*QQ^* = QI_mQ^* = QQ^* = P_A.$$

The next property we show is that P_A is Hermitian and thus an orthogonal projector:

$$P_A^* = (QQ^*)^* = Q^{**}Q^* = QQ^* = P_A.$$

Finally, we prove that P_A projects onto $\mathrm{im}\,(A)$. From the definition of P_A it follows immediately that $\mathrm{im}\,(P_A) \subseteq \mathrm{im}\,(Q) = \mathrm{im}\,(A)$. As $\mathrm{im}\,(P_A) = \ker\,(I_m - P_A)$ (compare Proposition 2.3.37), it suffices to show that $\ker\,(I_m - P_A) \supseteq \mathrm{im}\,(Q)$. For every $v \in \mathbb{C}^m$ we observe that

$$(I_m - P_A)\,Qv = Qv - QQ^*Qv = Qv - Qv = 0_m,$$

which concludes the proof. □

Proposition 2.3.42 (Uniqueness)**.** *For each $A \in \mathrm{Mat}_{m,n}\,(\mathbb{C})$ the orthogonal projector P_A onto $\mathrm{im}\,(A)$ is unique.*

Proof. See, for instance, [6, Subsection 2.6.1]. □

Example 2.3.43. If $A \in \mathrm{Mat}_m\,(\mathbb{C})$ has full rank, then every matrix $Q \in \mathrm{Mat}_m\,(\mathbb{C})$ which contains as its columns an orthonormal basis of $\mathrm{im}\,(A) = \mathbb{C}^m$ is unitary. Consequently, the projector P_A is identical to the unit matrix I_m.

Corollary 2.3.44. *Let $v \in \mathrm{Mat}_{m,1}\,(\mathbb{C})$ be a non-zero column vector. Then $P_v = \frac{vv^*}{v^*v}$ is the orthogonal projector onto $\mathrm{im}\,(v)$.*

Proof. This is a direct consequence of Theorem 2.3.41. □

Remark 2.3.45. An immediate consequence is that $I_m - P_v$ is the orthogonal projector onto $P_{v^\perp}$, where we denote by $v^\perp$ the set of all vectors in $\mathbb{C}^m$ which are orthogonal to v.

Definition 2.3.46. [Eigenvalue]

Let $A \in \mathrm{Mat}_m\,(\mathbb{C})$. A scalar $\lambda \in \mathbb{C}$ is called an **eigenvalue** of A if

$$\ker\,(A - \lambda I_m) \neq \{0_m\}.$$

Given an eigenvalue λ of A, the dimension of $\ker\,(A - \lambda I_m)$ is called the **geometric multiplicity** of λ and will be denoted by $\mathrm{gmult}\,(A, \lambda)$.

Definition 2.3.47. Let $A \in \mathrm{Mat}_m\,(\mathbb{C})$ and let $\mathbb{C}\,[x]$ be the polynomial ring over $\mathbb{C}$ in the indeterminate x. The polynomial

$$p_A\,(x) = \det\,(A - xI_m) \in \mathbb{C}\,[x]$$

is called the **characteristic polynomial** of A.

Proposition 2.3.48. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$. A number $\lambda \in \mathbb{C}$ is an eigenvalue of A if and only if λ is a solution of the polynomial equation*

$$p_A(x) = 0$$

over $\mathbb{C}$.

Proof. We observe that, whenever $\lambda \in \mathbb{C}$ is a root of $p_A(x)$ the fact that $\det(A - \lambda I_m) = 0$ implies that $\ker(A - \lambda I_m) \neq \{0_m\}$. Additionally, if we assume that λ is an eigenvalue of A, then $\ker(A - \lambda I_m) \neq \{0_m\}$ implies that $\det(A - \lambda I_m) = 0$ and consequently λ has to be a root of $p_A(x)$. □

Corollary 2.3.49. *A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ has at most m (distinct) eigenvalues.*

Proof. As $p_A(x)$ is a polynomial of degree m in $\mathbb{C}[x]$, it has m roots over $\mathbb{C}$ and therefore at most m distinct roots. □

Definition 2.3.50. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $p_A(x)$ be the associated characteristic polynomial. The multiplicity of each (complex) root λ of $p_A(x)$ is called the **algebraic multiplicity** of λ in A.

Definition 2.3.51. [Spectrum]

The set of all eigenvalues of a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called its **spectrum** and is denoted by $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$ with $1 \leq q \leq m$.

Definition 2.3.52. [Eigenvector/Eigenspace]

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$ be the spectrum of A. A non-zero column vector $v \in \mathrm{Mat}_{m,1}(\mathbb{C})$ is called a **(right-hand) eigenvector** of A corresponding to λ_i if the equation $(A - \lambda_i I_m)\, v = 0_m$ holds. Similarly, a non-zero row vector $v \in \mathrm{Mat}_{1,m}(\mathbb{C})$ is called a **left-hand eigenvector** of A corresponding to λ_i if $v(A - \lambda_i I_m) = 0_m$ is satisfied. We call the set of all right-hand eigenvectors v together with 0_m belonging to λ_i the **eigenspace** of A for λ_i and denote it by $\mathrm{Eig}(A, \lambda_i)$.

An intuitive interpretation of an eigenvector is a vector which is only scaled, i.e. multiplied by a scalar, by the linear transformation given through the transformation matrix A.

Remark 2.3.53. For a given matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ with spectrum $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$, each set $\mathrm{Eig}(A, \lambda_i)$ is a linear subspace of $\mathbb{C}^m$.

Proof. First, we observe that $0_m \in \mathrm{Eig}(A, \lambda_i)$ by definition. If $v, w \in \mathrm{Eig}(A, \lambda_i)$, then $v + w$ is again an eigenvector of A with respect to the eigenvalue λ_i and therefore $v + w \in \mathrm{Eig}(A, \lambda_i)$. Finally, if $c \in \mathbb{C}$ and $v \in \mathrm{Eig}(A, \lambda_i)$, then vc is also an eigenvector of A with respect to λ_i and consequently $vc \in \mathrm{Eig}(A, \lambda_i)$. □

Proposition 2.3.54. *A row vector $v \in \mathrm{Mat}_{1,m}(\mathbb{C})$ is a left-hand eigenvector of a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ if and only if v^{tr} is a right-hand eigenvector of A^{tr}.*

Proof. Let $v \in \mathrm{Mat}_{1,m}(\mathbb{C})$ be a left-hand eigenvector of A. Then we can compute

$$\begin{aligned} vA = \lambda v \quad &\Longleftrightarrow \\ (vA)^{\mathrm{tr}} = \lambda v^{\mathrm{tr}} \quad &\Longleftrightarrow \\ A^{\mathrm{tr}} v^{\mathrm{tr}} = \lambda v^{\mathrm{tr}} & \end{aligned}$$

and have thus shown that v^{tr} is a right-hand eigenvector of A^{tr}. Furthermore, if we assume that v^{tr} is a right-hand eigenvector of A^{tr} the same arguments can be used and we obtain that v has to be a left-hand eigenvector of A. □

Definition 2.3.55. A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **invertible** if a matrix $B \in \mathrm{Mat}_m(\mathbb{C})$ exists such that

$$AB = BA = I_m.$$

Proposition 2.3.56. *If $A \in \mathrm{Mat}_m(\mathbb{C})$ is invertible and $B \in \mathrm{Mat}_m(\mathbb{C})$ is such that $AB = I_m$, then the matrix B is unique.*

Proof. Let us assume that $A \in \mathrm{Mat}_m(\mathbb{C})$ is invertible and that $B, \tilde{B} \in \mathrm{Mat}_m(\mathbb{C})$ are such that $AB = A\tilde{B} = I_m$. We have

$$B = BI_m = B(A\tilde{B}) = (BA)\,\tilde{B} = I_m\tilde{B} = \tilde{B},$$

which shows that the matrix B is unique. □

Definition 2.3.57. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be invertible and let $B \in \mathrm{Mat}_m(\mathbb{C})$ such that $AB = I_m$. Then B is called the **inverse** of A and denoted by A^{-1}.

Remark 2.3.58. The invertible matrices $A \in \mathrm{Mat}_m(\mathbb{C})$ form a group. This group is called the **general linear group** and will be denoted by $\mathrm{GL}_m(\mathbb{C})$.

Definition 2.3.59. Two matrices $A, B \in \mathrm{Mat}_m(\mathbb{C})$ are called **similar**, if there exists an invertible matrix $P \in \mathrm{GL}_m(\mathbb{C})$ such that $P^{-1}AP = B$.

Proposition 2.3.60. *Similar matrices have the same spectrum.*

Proof. Let $A, B \in \mathrm{Mat}_m(\mathbb{C})$ be similar, let $P \in \mathrm{GL}_m(\mathbb{C})$ be such that $PBP^{-1} = A$, and let $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$ be the spectrum of A. Now, let $1 \leq i \leq q$, and let $v \in \ker(A - \lambda_i I_m) \setminus \{0_m\}$ be an eigenvector of A corresponding to λ_i. Then we compute

$$\begin{aligned} 0_m &= (A - \lambda_i I_m)\, v \\ &= \left(PBP^{-1} - \lambda_i PP^{-1}\right) v \\ &= P\,(B - \lambda_i I_m)\, P^{-1} v \end{aligned}$$

which shows that λ_i is also an eigenvalue of B. □

Definition 2.3.61. [Generalised Eigenvector/Eigenspace]

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$ be its spectrum. A non-zero column vector $v \in \mathrm{Mat}_{m,1}(\mathbb{C})$ is called a **generalised eigenvector** of A corresponding to λ_i if

$$(A - \lambda_i I_m)^k v = 0_m$$

for some k with $1 \leq k \leq m$. We call the set of all generalised eigenvectors associated to λ_i together with 0_m the **generalised eigenspace** of A for λ_i and denote it by $\mathrm{Gen}(A, \lambda_i)$.

Remark 2.3.62. If $k = 1$ in the setting of Definition 2.3.61, then v is an ordinary (right-hand) eigenvector of A.

Remark 2.3.63. For a given matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ with spectrum $\Lambda(A) = \{\lambda_1, ..., \lambda_q\}$, the set $\mathrm{Gen}(A, \lambda_i)$ is a linear subspace of $\mathbb{C}^m$. Additionally, $\mathrm{Eig}(A, \lambda_i) \subseteq \mathrm{Gen}(A, \lambda_i)$ holds.

Proof. First, we observe that $0_m \in \mathrm{Gen}(A, \lambda_i)$ by definition. Now let $v, w \in \mathrm{Gen}(A, \lambda_i)$. This means that $(A - \lambda_i I_m)^{k_1} v = 0_m$ and $(A - \lambda_i I_m)^{k_2} w = 0_m$ for some $k_1, k_2 \in \{1, ..., m\}$. If we let $k = \max(k_1, k_2)$, then both

$$(A - \lambda_i I_m)^k v = 0_m \quad \text{and} \quad (A - \lambda_i I_m)^k w = 0_m$$

are still true. As the equation

$$(A - \lambda_i I_m)^k (v + w) = (A - \lambda_i I_m)^k v + (A - \lambda_i I_m)^k w = 0_m$$

holds, we can conclude that $v + w$ is again a generalised eigenvector of A with respect to the eigenvalue λ_i and therefore $v + w \in \mathrm{Eig}(A, \lambda_i)$. Finally, if $c \in \mathbb{C}$ and $v \in \mathrm{Gen}(A, \lambda_i)$, then vc is also a generalised eigenvector of A with respect to λ_i and consequently $vc \in \mathrm{Eig}(A, \lambda_i)$. □

Definition 2.3.64. [Spectral Radius]

Let $\lambda(A) = \{\lambda_1, ..., \lambda_q\}$ be the spectrum of $A \in \mathrm{Mat}_m(\mathbb{C})$. Then we call $\varrho(A) = \max\limits_{1 \leq i \leq q} (|\lambda_i|)$ the **spectral radius** of A.

The following well-known theorem allows us to bound the spectral radius of a matrix A.

Theorem 2.3.65. *Let $1 \leq p \leq \infty$, $A \in \mathrm{Mat}_m(\mathbb{C})$, and let $\|\cdot\|_p$ be the induced matrix norm. Then*

$$\varrho(A) \leq \sqrt[k]{\|A^k\|_p}$$

for all $k \in \mathbb{N}$.

Proof. Let $\lambda \in \mathbb{C}$ be an eigenvalue of $A \in \mathrm{Mat}_m(\mathbb{C})$, and let v be an associated eigenvector. We know that $|\lambda|^k \|v\|_p = \|\lambda^k v\|_p = \|A^k v\|_p$. By Proposition 2.3.13, all induced matrix norms are consistent and it follows that $|\lambda|^k \|v\|_p = \|A^k v\|_p \leq \|A^k\|_p \|v\|_p$ holds. Because v as an eigenvector is different from 0_m we can divide by $\|v\|_p$ and obtain $|\lambda|^k \leq \|A^k\|_p$. This proves, that for every eigenvalue λ of A the inequality $|\lambda| \leq \sqrt[k]{\|A^k\|_p}$ holds, and thus $\varrho(A) \leq \sqrt[k]{\|A^k\|_p}$. □

2.4 Measuring Computational Cost

Another important aspect of an algorithm is its runtime in terms of the size of the input data. In computer science, it is common to measure the runtime of an algorithm in an asymptotic way. For growing input sizes, the most costly part of an algorithm will start to dominate. Therefore, it makes sense to classify algorithms by their dominating cost factor. When comparing certain implementations of an algorithm, this kind of cost measure may be too crude and it makes sense to analyse the exact number of elementary (floating point) operations (FLOPs) that are required until the algorithm terminates. We now introduce some common definitions, most prominently the big O notation which is used to express the worst case runtime behaviour. A very detailed account of how computational cost can be measured is contained in [8, Chapter 3].

Definition 2.4.1. [Big O-Notation]

Let $f(n) : S \to \mathbb{R}$ be a function with $S \subseteq \mathbb{R}$. We denote by $O(f(n))$ the set of functions

$$O(f(n)) = \{g(n) \,|\exists c > 0, \exists n_0 > 0, \forall n > n_0 \,(|g(n)| \leq c\,|f(n)|)\}.$$

Remark 2.4.2. To express that $g(n) \in O(f(n))$, it is common in computer science to write $g(n) = O(f(n))$. Note that this represents a slight abuse of notation, as the equal sign neither is symmetric nor a true equality in the usual sense (compare [8, pages 44-45]). In this thesis we will avoid the latter notation.

Example 2.4.3. Let us assume that an algorithm G performs $g(n) = 10n^3 + 3n^2 + \lceil n \log_2 n \rceil$ basic operations depending on the size of the input data n before it terminates. Then the runtime of the algorithm G is in $O\left(n^3\right)$, and we write $g(n) \in O\left(n^3\right)$.

Even though two algorithms may have the same asymptotic runtime, their actual performance may differ drastically, as the O-notation may obfuscate large constants. Especially in numerical linear algebra, the performance difference between two algorithms is quite often just a constant factor. So, sometimes it may be desirable to compare their runtime in a more accurate way. As stated before it would be possible to count all operations, however, this is usually too tedious. Let us assume that Algorithm 1 requires in total $g_1(n) = 10n^3 + 2n$ flops, and Algorithm 2 requires $g_2(n) = 5n^3 + n$ flops to complete a certain task. Though, in fact, we could write that $g_1(n) \in O\left(n^3\right)$ and $g_2(n) \in O\left(n^3\right)$ we write $g_1(n) \in O\left(10n^3\right)$ and $g_2(n) \in O\left(5n^3\right)$ to emphasise that Algorithm 2 is in fact twice as fast as Algorithm 1.

2.4.1 Runtime of Basic Linear Algebra Algorithms

First of all we derive a bound for the complexity of computing the kernel of a matrix via Gauss-Jordan elimination.

Proposition 2.4.4. *Let $A \in \mathrm{Mat}_{m,n}(K)$. Then the cost of computing $\ker(A)$ via Gauss-Jordan elimination is in $O\left(\min(m,n)^2 \max(m,n)\right)$, if addition and multiplication of two elements in K can be performed in $O(1)$.*

Proof. Let us start with the case where $m \geq n$. The first row contains n elements and needs to be added to at most $m-1$ other rows after being multiplied with a scalar. The second row only contains $n-1$ elements but needs to be added to $m-1$ rows after being multiplied with a scalar as well. If we continue this process we have a cost of $O\left((m-1)\sum_{i=1}^{n} 2i\right) = O\left((m-1)\,n\,(n+1)\right) = O\left(mn^2+mn-n^2-n\right) = O\left(mn^2\right)$. The cost for forming the kernel vectors is in $O(mn)$ and can therefore be neglected in the big O notation. In the case $m < n$ essentially the same arguments apply and we arrive at $O\left(m^2 n\right)$. So for an arbitrary matrix the cost is in $O\left(\min(m,n)^2 \max(m,n)\right)$. □

Remark 2.4.5. Using the same reasoning, the cost for computing the reduced row echelon form of a matrix via Gauss-Jordan elimination is also in $O\left(\min(m,n)^2 \max(m,n)\right)$.

Proposition 2.4.6. *Let $R \in \mathrm{Mat}_{m,n}(K)$ be an upper triangular matrix, meaning that the not necessarily square matrix has only zero entries below its main diagonal. The cost to transfer this matrix into reduced row echelon form is in $O\left(\frac{1}{6}n^3\right)$ if $m \geq n$ and it is in $O\left(\frac{1}{2}m^2 n - \frac{1}{3}m^3\right)$ if $n < m$.*

Proof. Let us initially consider the case $m \geq n$. The first row needs to be multiplied by a constant K such that the pivot entry becomes 1, and the cost of this operation is in $O(n)$. The second row contains $n-1$ entries. Those have to be multiplied with a scalar and have to be added to the first row. This costs $O\left((n-1)+\right)$ operations. So the cumulated cost for the i-th row is in $O\left(i\,(n-i)\right)$. This means we have a total cost of $\sum_{i=1}^{n}\left(i\,(n-i+1)\right) = \frac{n(n+1)(n+2)}{6}$, which again means the cost is in $O\left(\frac{1}{6}n^3\right)$.

Now let us look at the case $m < n$. Here we obtain $\sum_{i=1}^{m}\left(i\,(n-i+1)\right) = \frac{m^2 n}{2} - \frac{m^3}{3} + \frac{m(3n+2)}{6}$, which means that the cost is in $O\left(\frac{1}{2}m^2 n - \frac{1}{3}m^3\right)$. □

2.5 Canonical Matrix Factorisations

In this section we present some well-known matrix factorisations which are of either theoretical value, like the Jordan decomposition, or play an important role in numerical linear algebra, like the QR decomposition or the Schur decomposition.

2.5.1 PLURQ Decomposition

A PLURQ decomposition essentially stores the relevant operations which transform a matrix into its reduced row echelon form. It can, for example, be used to compute the kernel of a matrix. If later on a new column is added to this matrix it is not necessary to recompute the kernel from scratch, as the PLURQ decomposition can be updated, which takes significantly less time (compare Remark 3.4.5). As the transformations which are involved are not unitary, this decomposition is mostly relevant when implemented in exact arithmetic. The first step when computing a PLURQ decomposition is to bring the matrix into row echelon form. This amounts

to the computation of a PLUQ decomposition of a matrix A (compare [6, Algorithm 3.4.2]). Afterwards the matrix U in this decomposition is further decomposed into another upper triangular matrix and the reduced row echelon form of A.

Definition 2.5.1. [Permutation matrix]

A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **permutation matrix** if each row and column of A contains exactly one 1 entry and 0s in all other locations.

Example 2.5.2. The matrix

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathrm{Mat}_3(\mathbb{C})$$

is a permutation matrix.

Definition 2.5.3. [PLURQ decomposition of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(K)$, where K is an arbitrary field. A decomposition of A such that

$$A = PLURQ$$

where $P \in \mathrm{Mat}_m(K)$ and $Q \in \mathrm{Mat}_n(K)$ are permutation matrices, $L \in \mathrm{Mat}_m(K)$ is a lower triangular matrix of full rank, $U \in \mathrm{Mat}_m(K)$ is an upper triangular matrix of full rank, and $R \in \mathrm{Mat}_{m,n}$ is in reduced row echelon form is called **PLURQ decomposition** of A.

In the following, we present an algorithm which computes a PLURQ decomposition of a matrix $A \in \mathrm{Mat}_{m,n}(K)$. Hence every matrix has a PLURQ decomposition.

Remark 2.5.4. In general, a PLURQ decomposition of a matrix is not unique. Consider, for instance, the zero matrix. In this case, the matrix R is also the zero matrix but P and Q can be arbitrary permutation matrices and L and U can be arbitrary lower- and upper triangular matrices.

Example 2.5.5. Let

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 4 \end{pmatrix}.$$

A PLURQ decomposition of A is given by

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \; L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; U = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ and } Q = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Algorithm 1: PLURQ decomposition

Input: A matrix $A \in \mathrm{Mat}_{m,n}(K)$

Output: Matrices P, L, U, R, Q, such that $A = PLURQ$

1 $k := \min(m,n)$, $P := I_m$, $L := I_m$, $U := I_m$, $R := A$, $Q := I_n$, rank $:= k$;
`// First a PLUQ decomposition of` A `is computed`
2 **for** $i := 1$ **to** k **do**
3 | **if** $isZeroMatrix(R_{i:m,i:n})$ **then** rank $:= i-1$; **break**;
4 | $[r,c] :=$ findFirstNonZeroIndex$(R_{i:m,i:n})$;
5 | swapColumns$(P, i, i+r-1)$;
6 | swapRows$(R, i, i+r-1)$;
7 | swapColumns$(R, i, i+c-1)$;
8 | swapRows$(Q, i, i+c-1)$;
9 | $R_{i+1:m,i} := R_{i+1:m,i}/R_{i,i}$;
10 | $R_{i+1:m,i+1:n} := R_{i+1:m,i+1:n} - R_{i+1:m,i}R_{i,i+1:n}$;
11 **end**
`// L and U are split into separate matrices`
12 **for** $i := 1$ **to** m **do**
13 | $L_{i,1:\min(n,i-1)} := R_{i,1:\min(n,i-1)}$; $R_{i,1:\min(n,i-1)} := 0$;
14 **end**
15 **for** $i :=$ *rank* **to** *rank* **do**
16 | $L_{1:m,i} := L_{1:m,i}R_{i,i}$; $R_{i,i:n} = R_{i,i:n}/R_{i,i}$;
17 **end**
`// R is transformed into RREF`
18 **for** $i :=$ *rank* **downto** *2* **do**
19 | $U_{1:i-1,i} := R_{1:i-1,i}$;
20 | $R_{1:i-1,rank+1:n} := R_{1:i-1,rank+1:n} - R_{1:i-1,i}R_{i,rank+1:n}$;
21 | $R_{1:i-1,i} := 0$;
22 **end**
23 **return** (P, L, U, R, Q);

Theorem 2.5.6. *Algorithm 1 is an algorithm which computes in a finite number of steps a PLURQ decomposition of a matrix* $A \in \mathrm{Mat}_{m,n}(K)$.

Proof. First a PLUQ decomposition of A is computed. Correctness of this part follows from [6, Algorithm 3.4.2] together with the observation that the arguments remain valid for rectangular matrices. The matrix R is now in upper triangular form. In lines 18 to 22 it is transformed into RREF via a series of elementary row operations and the inverse operations are stored in matrix U. □

Remark 2.5.7. In practice, it makes sense not to store all entries of the matrices P and Q as they are permutation matrices and thus sparse.

Later on, we are most interested in the matrices L^{-1} and U^{-1}, because with their help it is possible to "update" the kernel of a matrix. We will discuss this in more detail in Remark 3.4.5.

Therefore, we present an algorithm which computes L^{-1} and U^{-1} directly, without the need of first computing L and U via Algorithm 1 and afterwards inverting them. Please note that in this algorithm L^{-1}, U^{-1},... have to be interpreted as variable names and are not meant as instructions to invert a given matrix L or U.

Algorithm 2: PLURQ inverse decomposition

Input: A matrix $A \in \mathrm{Mat}_{m,n}(K)$

Output: Matrices $P^{-1}, L^{-1}, U^{-1}, R, Q^{-1}$, such that $R = U^{-1}L^{-1}P^{-1}AQ^{-1}$

$k := \min(m,n)$, $P^{-1} := L^{-1} := U^{-1} := I_m$, $R := A$, $Q^{-1} := I_n$, rank $:= k$;
// First a PLUQ decomposition of A is computed
for $i := 1$ **to** k **do**
  if *isZeroMatrix*$(R_{i:m,i:n})$ **then** rank $:= i-1$; **break**;
  $[r,c] :=$ findFirstNonZeroIndex$(R_{i:m,i:n})$;
  swapRows$(P^{-1}, i, i+r-1)$;
  swapRows$(R, i, i+r-1)$;
  swapColumns$(R, i, i+c-1)$;
  swapColumns$(Q^{-1}, i, i+c-1)$;
  $R_{i+1:m,i} := -R_{i+1:m,i}/R_{i,i}$;
  $R_{i+1:m,1:i-1} := R_{i+1:m,1:i-1} + R_{i+1:m,i}R_{i,1:i-1}$;
  $R_{i+1:m,i+1:n} := R_{i+1:m,i+1:n} + R_{i+1:m,i}R_{i,i+1:n}$;
  $R_{i,1:i-1} := R_{i,1:i-1}/R_{i,i}$;
  $R_{i,i+1:n} := R_{i,i+1:n}/R_{i,i}$;
  $R_{i,i} := 1/R_{i,i}$;
end
// L^{-1} and U^{-1} are split into separate matrices
for $i := 1$ **to** m **do**
  $L^{-1}_{i,1:\min(n,i-1)} := R_{i,1:\min(n,i-1)}$; $R_{i,1:\min(n,i-1)} := 0$;
end
for $i :=$ *1* **to** *rank* **do** $R(i,i) := 1$;
for $i :=$ *rank+1* **to** m **do** $L(i,i)^{-1} := 1$;
// R is transformed into RREF
for $i :=$ *rank* **downto** *2* **do**
  $U^{-1}_{1:i-1,i} := -R_{1:i-1,i}$;
  $R_{1:i-1,rank+1:n} := R_{1:i-1,rank+1:n} - R_{1:i-1,i}R_{i,rank+1:n}$;
  $R_{1:i-1,i} := 0$;
  $U^{-1}_{1:i-1,i+1:m} := U^{-1}_{1:i-1,i+1:m} + U^{-1}_{1:i-1,i}U^{-1}_{i,i+1:m}$;
end
return $(P^{-1}, L^{-1}, U^{-1}, R, Q^{-1})$;

Theorem 2.5.8. *Algorithm 2 is an algorithm which computes in a finite number of steps an inverse PLURQ decomposition of a matrix* $A \in \mathrm{Mat}_{m,n}(K)$ *and returns matrices* $P^{-1}, L^{-1}, U^{-1}, R$, *and* Q^{-1} *such that* $R = U^{-1}L^{-1}P^{-1}AQ^{-1}$.

Proof. Essentially the same arguments apply as for Algorithm 1. The operations are, however, accumulated in such a way that we obtain the inverse of the matrices P, L, U, and Q. □

2.5.2 Schur Decomposition

Another matrix decomposition which plays an important role in numerical computations is the so-called Schur decomposition. Its main advantages are that it can be computed using only unitary similarity transformations and that it exists for all square matrices.

Definition 2.5.9. [Schur Decomposition]

Let $A \in \mathrm{Mat}_m(\mathbb{C})$. Then a decomposition of A such that

$$A = QUQ^*$$

where $Q \in \mathrm{Mat}_m(\mathbb{C})$ is unitary and $U \in \mathrm{Mat}_m(\mathbb{C})$ is upper triangular, is called a **Schur decomposition** of A.

Theorem 2.5.10. *Every matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ has a Schur decomposition. The eigenvalues of A can be found on the diagonal of U and the matrix Q can be chosen to achieve any desired order of the eigenvalues on the diagonal of U. If $A \in \mathrm{Mat}_m(\mathbb{R})$ and if all its eigenvalues are real, it is possible to chose an orthogonal $Q \in \mathrm{Mat}_m(\mathbb{R})$.*

Proof. A constructive proof can be found in [6, Theorem 7.1.3]. Compare also [9, Theorem 2.3.1]. □

Remark 2.5.11. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $A = QUQ^*$ be a Schur decomposition of A. This decomposition is in general not unique. Consider for instance $A = I_m$, then any unitary matrix $Q \in \mathrm{Mat}_m(\mathbb{C})$ together with $U = I_m$ will be a Schur decomposition of A.

2.5.3 QR Decomposition

In the following subsection we now introduce the well-known definition of the QR decomposition of a matrix. It can, for example, be used to solve the linear least squares problem (2.10.1) which we discuss later.

Definition 2.5.12. [QR decomposition of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $m \geq n$. A decomposition of A such that

$$A = QR$$

where $Q \in \mathrm{Mat}_m(\mathbb{C})$ is a unitary matrix and $R \in \mathrm{Mat}_{m,n}(\mathbb{C})$ is an upper triangular matrix with $m - n$ appended zero rows, is called a **QR decomposition** of A.

In this case the columns of Q form an orthonormal basis of $\mathbb{C}^m$. Obviously it is possible to remove (at least) the last $m-n$ columns from Q and the same number of zero rows from R without changing the value of the product QR. Such a decomposition is called reduced or thin QR decomposition in the literature. As the last $m-n$ columns of Q can be chosen freely to extend the first n columns to an orthonormal basis of $\mathbb{C}^m$, a QR decomposition is in general not unique.

Definition 2.5.13. [Reduced QR decomposition of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $m \geq n$. We denote the rank of A by k. A decomposition of A such that

$$A = QR,$$

with $Q \in \mathrm{Mat}_{m,k}(\mathbb{C})$ containing an orthonormal basis of $\mathrm{im}(A)$ as its columns and an upper triangular matrix $R \in \mathrm{Mat}_{k,n}(\mathbb{C})$, is called **reduced** (or **thin**) **QR decomposition** of A.

In this thesis the most frequently occurring case is when A has full rank n. Then the matrix A can then be decomposed into $Q \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $R \in \mathrm{Mat}_{n}(\mathbb{C})$.

Theorem 2.5.14. *Every matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ has a QR decomposition and a reduced QR decomposition. If $A \in \mathrm{Mat}_{m,n}(\mathbb{R})$, then both Q and R can be chosen as real matrices.*

Proof. A proof can be found in [9, Theorem 2.6.1]. □

Remark 2.5.15. In Section 2.8 we describe an algorithm which computes a QR decomposition of a given matrix A.

Theorem 2.5.16 (Uniqueness)**.** *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, $m \geq n$, and $\mathrm{rank}(A) = n$. Then A has a unique reduced QR decomposition if we demand that the diagonal elements of R are real and positive, which can always be achieved.*

Proof. See [9, Theorem 2.6.1]. □

Remark 2.5.17. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be a rank k matrix with $m \geq n$ and let $A = QR$ be a reduced QR decomposition of A. As Q contains an orthonormal basis of $\mathrm{im}(A)$ as its columns, we observe that $Q^*Q = I_k(\mathbb{C})$. However, the matrix QQ^* is in general not the identity matrix.

There are three commonly used techniques for the calculation of the QR decomposition of a matrix. They are the modified Gram-Schmidt orthonormalisation process, the transformation using Givens rotations, and the transformation using Householder reflections. Each method has its particular advantages and disadvantages. The modified Gram-Schmidt process is the fastest algorithm, but it is numerically the least favourable. Givens rotations can be used when dealing with highly structured matrices which already contain a lot of zero entries. However, if the matrices are generic, Givens rotations require more time to compute the QR decomposition. Finally, Householder reflections represent a good compromise between numerical stability and efficiency when applied to matrices which lack special structure. That is why we will not focus on the first two methods. The interested reader is referred to [6, Section 5.2] for more details.

In Section 2.8 we will study in detail how the (reduced) QR decomposition of a matrix can be effectively computed via Householder reflections.

2.5.4 Eigendecomposition

First, we will introduce the concept of diagonalisable matrices and explain how they are related to eigenvalues and eigenvectors of a matrix. Of particular interest in numerical computations are matrices which can be unitarily diagonalised, as these eigenvalue revealing factorisations can be computed in a stable way.

Definition 2.5.18. [Diagonalisable matrix]

A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **diagonalisable**, if there exist a matrix $P \in \mathrm{GL}_m(\mathbb{C})$ and a diagonal matrix $D \in \mathrm{Mat}_m(\mathbb{C})$ such that $A = PDP^{-1}$. A decomposition of this form is called an **eigendecomposition** of A. If A is not diagonalisable, it is sometimes called **defective** in the literature.

Proposition 2.5.19. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonalisable matrix, and let $P \in \mathrm{GL}_m(\mathbb{C})$ be such that $P^{-1}AP = D$ is a diagonal matrix. Then $D = \mathrm{diag}(d_1, ..., d_m) \in \mathrm{Mat}_m(\mathbb{C})$ contains all eigenvalues $\{\lambda_1, ..., \lambda_q\}$ of A with their respective geometric multiplicity on its diagonal (in an arbitrary order). This means that for each $\lambda_i \in \Lambda(A)$ there are precisely $\mathrm{gmult}(A, \lambda_i)$ diagonal entries d_k which are equal to λ_i. Consider the rewritten equation $AP = PD$. The columns of P which are associated with λ_i in the product PD contain a basis of $\ker(A - \lambda_i I_m)$. So if $\dim(\ker(A - \lambda_i I_m)) = 1$, then the i-th column of P contains the eigenvector associated with d_i (for some $\lambda_k = d_i$).*

Proof. First, we note that the matrix P is invertible which means that the columns of P are linearly independent. By writing the equation $P^{-1}AP = D$ as $AP = DP$, we immediately observe that the columns of P are (linearly independent) eigenvectors of A. If $\dim(\ker(A - \lambda_i I_m)) = 1$ the eigenvector associated with the eigenvalue λ_i is uniquely determined. This means that the entry d_i of D is the eigenvalue associated with the eigenvector which is stored in the i-th column of P. In case $\dim(\ker(A - \lambda_i I_m)) > 1$, the columns of P which are associated with λ_i (see above) are linearly independent, therefore they form a basis of $\ker(A - \lambda_i I_m)$. □

Remark 2.5.20. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonalisable matrix. In this setting an eigendecomposition of A, such that $A = PBP^{-1}$, is in general not unique as the eigenvectors of A can be arranged in an arbitrary way as the columns of P.

Definition 2.5.21. A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **unitarily diagonalisable** if a unitary matrix $U \in \mathrm{Mat}_m(\mathbb{C})$ exists such that U^*AU is a diagonal matrix.

Theorem 2.5.22 (Spectral theorem)**.** *A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is unitarily diagonalisable if and only if it is normal.*
In this case, we write $A = UDU^*$ where $U \in \mathrm{Mat}_m(\mathbb{C})$ is unitary and where $D \in \mathrm{Mat}_m(\mathbb{C})$ is diagonal. Then the columns of U contain an orthonormal basis of $\mathbb{C}^m$ and D contains the eigenvalues of A on its main diagonal.

Proof. By the Schur decomposition theorem (2.5.10) every matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ can be decomposed into

$$A = QUQ^*$$

where $Q \in \mathrm{Mat}_m(\mathbb{C})$ is unitary and $U \in \mathrm{Mat}_m(\mathbb{C})$ is upper triangular and similar to A. We observe that

$$\begin{aligned} A^*A &= (QUQ^*)^* QUQ^* = Q^{**}U^*Q^*QUQ^* = QU^*UQ^* \\ AA^* &= QUQ^*(QUQ^*)^* = QUQ^*Q^{**}U^*Q^* = QUU^*Q^*. \end{aligned}$$

If we now assume that A is normal we obtain that $U^*U = UU^*$ must hold. So U has to be both normal and upper triangular, which can only be fulfilled if U is diagonal (see Proposition 2.3.28). In order to show the other direction of the claimed implication we assume that U is diagonal. Then $U^*U = UU^*$ follows immediately and therefore, $A^*A = A^*$ must hold as well. Finally, as $AU = UD$, it follows from the definition of eigenvalues and eigenvectors that UDU^* is an eigendecomposition of A. □

Remark 2.5.23. Note that the matrix U in Theorem 2.5.22 need not be unique. Consider $A = I_m(\mathbb{C})$. Then every unitary matrix $U \in \mathrm{Mat}_m(\mathbb{C})$ unitarily diagonalises A. However, the entries of D are unique except for their order as the eigenvalues of A are unique.

Even though not all matrices $A \in \mathrm{Mat}_m(\mathbb{C})$ are similar or even unitarily similar to a diagonal matrix, it is always possible to find similarity transformations such that A is similar to a matrix containing only Jordan blocks (compare Definition 2.5.24).

2.5.5 Jordan Normal Form

As a next step we will present the Jordan Decomposition of a matrix which generalises the eigendecomposition of diagonalisable matrices $A \in \mathrm{Mat}_m(\mathbb{C})$ to general square matrices.

Definition 2.5.24. [Jordan Block]

Let $\lambda \in \mathbb{C}$. Then a matrix of the form

$$J = \begin{pmatrix} \lambda & 1 & 0 & 0 \\ 0 & \lambda & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & \lambda \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{C})$$

is called a **Jordan block** of size m.

Remark 2.5.25. For $m \geq 2$, no Jordan block J can be diagonalised. Obviously, the matrix J has only one eigenvalue λ with algebraic multiplicity m. However, $\ker(J - \lambda I_m) = \mathbb{C} \cdot e_1$ is a 1-dimensional vector space. As $\dim(\ker(J - \lambda I_m)) = 1 \neq m$ there exists no basis of eigenvectors for J.

Definition 2.5.26. [Block Diagonal]

A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **block diagonal** if it is of the form

$$A = \begin{pmatrix} B_1 & 0 & \cdots & 0 \\ 0 & B_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & B_k \end{pmatrix}$$

where each $B_i \in \mathrm{Mat}_{m_i}(\mathbb{C})$ with $1 \leq i \leq k$ is an arbitrary square matrix.

Definition 2.5.27. [Jordan Normal Form]

We say a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is in **Jordan normal form** if it is block diagonal where each diagonal block is a Jordan block.

Theorem 2.5.28. *Every square matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is similar to a matrix in Jordan normal form. There exists an invertible matrix $P \in \mathrm{Mat}_m(\mathbb{C})$ containing as its columns (generalised) eigenvectors of A such that*

$$P^{-1}AP = J$$

is in Jordan normal form. Specifically,

$$A = PJP^{-1}$$

*is called **Jordan decomposition** of A. If A is real and has only real eigenvalues it is always possible to choose $P \in \mathrm{Mat}_m(\mathbb{R})$ as well.*

Proof. A proof can be found in [9, Theorem 3.1.11]. □

Definition 2.5.29. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. We say the matrix $\tilde{A} \in \mathrm{Mat}_{m,n}(\mathbb{C})$ is an **ε-perturbation** of A if $\|A - \tilde{A}\|_2 \leq \varepsilon$.

It should be noted that every non-diagonalisable matrix is arbitrarily close to a diagonalisable one. This is captured in the following result.

Proposition 2.5.30. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $\varepsilon > 0$. Then there exists an ε-perturbation of A which is diagonalisable.*

Proof. A proof of this claim is contained in [9, Theorem 2.4.6]. □

Compare also [6, Subsections 7.1.5 and 7.6.5] for a discussion of the practical implications.

2.5.6 Singular Value Decomposition (SVD)

Another matrix decomposition which plays a major role in numerical linear algebra is the so-called singular value decomposition of a matrix.

Definition 2.5.31. [SVD of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. A **singular value decomposition** (SVD) of A is a matrix factorisation

$$A = U\Sigma V^*$$

where both $U \in \mathrm{Mat}_m(\mathbb{C})$ and $V \in \mathrm{Mat}_n(\mathbb{C})$ are unitary matrices, and $\Sigma \in \mathrm{Mat}_{m,n}(\mathbb{R})$ is a diagonal matrix which contains only non-negative real entries. We denote the entries on the diagonal of Σ by $s_1, ..., s_{\min(m,n)}$.

Theorem 2.5.32. *1. Every matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ has a SVD.*

*2. If we order the entries on the diagonal of Σ in a descending way, i.e. $s_1 \geq ... \geq s_{\min(m,n)}$, they are uniquely determined by A. We call $s_1, ..., s_{\min(m,n)}$ the **singular values** of A.*

3. The last $n - r$ rows of V^* form an orthonormal basis of the kernel of A, where r is the rank of A.

4. If the matrix A is real, then real orthogonal matrices $U \in \mathrm{Mat}_m(\mathbb{R})$ and $V \in \mathrm{Mat}_n(\mathbb{R})$ exist such that $U\Sigma V^*$ is a SVD of A.

Proof. For a proof of the claimed properties compare, for example, [9, Theorem 7.3.5]. A direct proof of claim 4 can also be found in [6, Theorem 2.5.2]. □

Whenever we talk about a singular value decomposition of a matrix in this thesis we assume w.l.o.g. that the singular values are ordered in a descending way.

In practice it is often sufficient and more economic to compute only the so-called reduced singular value decomposition of a matrix.

Definition 2.5.33. [Reduced SVD]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with rank r. A **reduced singular value decomposition** (reduced SVD) of A is a factorisation such that

$$A = U\Sigma V^*$$

holds, where the involved matrices have the following properties: $U \in \mathrm{Mat}_{m,r}(\mathbb{C})$ contains r orthogonal vectors in $\mathbb{C}^m$ which form a basis of $\mathrm{im}(A)$, $V \in \mathrm{Mat}_{n,r}(\mathbb{C})$ contains r orthogonal vectors in $\mathbb{C}^n$, and $\Sigma \in \mathrm{Mat}_r(\mathbb{R})$ is diagonal with non-negative real entries.

Definition 2.5.34. [Singular vectors]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let $U\Sigma V^*$ be a singular value decomposition of A such that $s_1 \geq ... \geq s_{\min(m,n)}$. Let us denote by $(u_1, ..., u_m)$ the column vectors of U and by $(v_1, ..., v_n)$ the column vectors of V. This means that $U\Sigma = AV$ holds, and consequently $u_i s_i = Av_i$ for $1 \leq i \leq \min(m, n)$. We call each u_i a **left-singular vector** of A with respect to s_i and we call each v_i a **right-singular vector** of A with respect to s_i.

A detailed explanation on how to compute a SVD of a matrix in an efficient and stable way is contained in [6, Section 8.6].

The following proposition tells us what will happen to the largest and smallest singular value of a matrix A if a new column is inserted into A. This will play a significant role when analysing the properties of the ABM algorithm (22).

Proposition 2.5.35. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let $\tilde{A} \in \mathrm{Mat}_{m,n+1}(\mathbb{C})$ be a matrix which we obtain by inserting a new column into A at an arbitrary position. Then the following relations between the largest singular values of A and $\tilde{A}$, s_1 and $\tilde{s}_1$, and the smallest singular values of A and $\tilde{A}$, $s_{\min(m,n)}$ and $\tilde{s}_{\min(m,n+1)}$, hold:*

$$\begin{aligned} \tilde{s}_1 &\geq s_1 \\ \tilde{s}_{\min(m,n+1)} &\leq s_{\min(m,n)}. \end{aligned}$$

Proof. See for instance [6, Corollary 8.6.3]. □

Once we have understood the relationship with the homogeneous least squares problem (see Definition 2.10.2), it is easy to give an intuitive explanation of Proposition 2.5.35 with respect to the smallest singular value. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let $\tilde{A} \in \mathrm{Mat}_{m,n+1}(\mathbb{C})$ be like in Proposition 2.5.35 and let x and $\tilde{x}$ be the solutions of the homogeneous least squares problems $\min_x \|Ax\|$ and $\min_{\tilde{x}} \|\tilde{A}\tilde{x}\|$. By adding an additional column the remainder of the homogeneous least squares solution cannot become bigger as the fit either stays identical or can be improved with the help of additional data, so $\|Ax\| \geq \|\tilde{A}\tilde{x}\|$.

2.6 Moore-Penrose Pseudoinverse

It is possible to generalise the concept of the inverse of a matrix to general, non square and non invertible, matrices. The most commonly used extension is the Moore-Penrose pseudoinverse.

Definition 2.6.1. [Moore-Penrose pseudoinverse]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be an arbitrary complex matrix. We call a matrix $B \in \mathrm{Mat}_{n,m}(\mathbb{C})$ the **(Moore-Penrose) pseudoinverse** of A if it satisfies the following conditions:

1. $ABA = A$
2. $BAB = B$
3. $(AB)^* = AB$
4. $(BA)^* = BA$

For convenience we will denote the Moore-Penrose pseudoinverse of A by A^+ and refer to it as the pseudoinverse of A.

Proposition 2.6.2 (Existence and Uniqueness). *For a given matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ the pseudoinverse A^+ exists and is unique. Let $A = U\Sigma V^*$ be a singular value decomposition of A and let $s_1, ..., s_k$ be the positive singular values of A. Let furthermore $\Sigma^+ = \mathrm{diag}\left(\frac{1}{s_1}, ..., \frac{1}{s_k}, 0, ...\right) \in \mathrm{Mat}_{n,m}(\mathbb{R})$ be the diagonal matrix which we obtain by taking the reciprocal of every positive singular value of A. Then the matrix $V\Sigma^+U^*$ is the pseudoinverse of A.*

Proof. In order to prove the existence of the pseudoinverse for every matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, we will show that the matrix $B = V\Sigma^+U^*$ satisfies all properties of the pseudoinverse. First we compute $\Sigma\Sigma^+ = \begin{pmatrix} I_{\min(m,n)} & 0 \\ 0 & 0 \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{C})$ and $\Sigma^+\Sigma = \begin{pmatrix} I_{\min(m,n)} & 0 \\ 0 & 0 \end{pmatrix} \in \mathrm{Mat}_n(\mathbb{C})$. Then we verify

$$\begin{aligned}
ABA &= U\Sigma V^*V\Sigma^+U^*U\Sigma V^* = U\Sigma\Sigma^+\Sigma V^* \\
&= U\Sigma V^* = A, \\
BAB &= V\Sigma^+U^*U\Sigma V^*V\Sigma^+U^* = V\Sigma^+\Sigma\Sigma^+U^* \\
&= V\Sigma^+U^* = B, \\
(AB)^* &= \left(U\Sigma V^*V\Sigma^+U^*\right)^* = U\left(\Sigma^+\right)^{\mathrm{tr}}V^*V\Sigma^{\mathrm{tr}}U^* \\
&= U\left(\Sigma^+\right)^{\mathrm{tr}}\Sigma^{\mathrm{tr}}U^* = U\left(\Sigma\Sigma^+\right)^{\mathrm{tr}}U^* \\
&= U\Sigma\Sigma^+U^* = U\Sigma V^*V\Sigma^+U^* \\
&= AB, \\
(BA)^* &= \left(V\Sigma^+U^*U\Sigma V^*\right)^* = V\Sigma^{\mathrm{tr}}U^*U\left(\Sigma^+\right)^{\mathrm{tr}}V^* \\
&= V\Sigma^{\mathrm{tr}}\left(\Sigma^+\right)^{\mathrm{tr}}V^* = V\left(\Sigma^+\Sigma\right)^{\mathrm{tr}}V^* \\
&= V\Sigma^+\Sigma V^* = V\Sigma^+U^*U\Sigma V^* \\
&= BA.
\end{aligned}$$

Next we will show uniqueness. For this purpose let us assume that matrices $B, C \in \mathrm{Mat}_{n,m}(\mathbb{C})$ exist that satisfy all properties of Definition 2.6.1. By calculation we conclude that

$$AB = (AB)^* = B^*A^* = B^*(ACA)^* = B^*A^*C^*A^* = (AB)^*(AC)^* = ABAC = AC,$$

and

$$BA = (BA)^* = ... = CA.$$

Finally, we observe that

$$B = BAB = CAB = CAC = C.$$

We have thus shown that the pseudoinverse is uniquely determined by A. □

Remark 2.6.3. Even though we know by Proposition 2.6.2 that we can compute the pseudoinverse of a matrix with the help of its singular value decomposition, this is often too costly as the computation of a SVD is computationally expensive. In practice other methods are used which are numerically less favourable but provide significantly better runtime.

Proposition 2.6.4 (Properties of the pseudoinverse). *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. The pseudoinverse A^+ of A has the following properties:*

1. *If* $A \in \mathrm{GL}_m(\mathbb{C})$, *then* $A^+ = A^{-1}$.
2. $(A^+)^+ = A$.
3. *If* $A \in \mathrm{Mat}_{m,n}(\mathbb{R})$ *then* $A^+ \in \mathrm{Mat}_{n,m}(\mathbb{R})$.

Proof. To prove the first claim we note that for $A \in \mathrm{GL}_m(\mathbb{C})$ the matrix A^{-1} satisfies all properties of Definition 2.6.1. We compute $AA^{-1}A = A$, $A^{-1}AA^{-1} = A^{-1}$, $\left(AA^{-1}\right)^* = I = I^*$, and $\left(A^{-1}A\right)^* = I = I^*$. By Proposition 2.6.2 we now that the pseudoinverse is unique which concludes the proof. To prove the second claim let us look at the pseudoinverse of A given by its SVD $A^+ = V\Sigma^+U^*$. We obtain

$$\left(A^+\right)^+ = \left(V\Sigma^+U^*\right)^+ = U\Sigma V^* = A.$$

The third claim follows from the fact that the matrices U and V can be chosen as purely real matrices if $A \in \mathrm{Mat}_{m,n}(\mathbb{R})$ (see Theorem 2.5.32 claim 4). □

Proposition 2.6.5. *Let* $m \geq n$. *If* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ *has full rank* n, *then* A^+ *is the left inverse of* A, *i.e.* $A^+A = I_n$.

Proof. Let $A = U\Sigma V^*$ be a singular value decomposition of A. The pseudoinverse of A is then given by $V\Sigma^+U^*$. We verify by computation, that

$$\begin{aligned} A^+A &= V\Sigma^+U^*U\Sigma V^* = V\Sigma^+\Sigma V^* \\ &= VI_nV^* = I_n. \end{aligned}$$

□

2.7 Numerical Stability

When we want to derive algorithms which are able to deal with measured noisy data in an appropriate way, we essentially face two challenges. Firstly, it is neither feasible nor desirable to work with exact representations of real (or complex) numbers inside the computer. That is why we use limited precision representations of the data in form of floating point numbers, and we have to address the numerical obstacles which are associated with it. Secondly, we face the challenge that we need to be able to assess the usefulness of a solution which we have computed in consideration of possible measurement errors contained in the input data. The solution to certain problems may change drastically if the input data vary only slightly, thus making the computed solution essentially meaningless in the context of measuring errors. This phenomenon is called the condition of a problem and is now briefly introduced. Additionally, we look at examples of ill-conditioned problems which will come up again in Chapter 5.

2.7.1 Arithmetic with Floating Point Numbers

Approximations of real or complex numbers are usually stored in computers in the floating point format which is natively supported by today's microprocessors. Guided by the IEEE 754 standard ([47]) we introduce the following definitions.

Definition 2.7.1. [Floating point numbers]

Let $t \in \mathbb{N}$, $b \in \mathbb{N} \setminus \{1\}$ (usually $b = 2$) and $r \in \mathbb{N}$. We call t the **precision**, b the **base**, and r the **range of the exponent**. Then the numbers

$$x = \pm \left(\frac{m}{b^t}\right) b^e$$

with $m \in \mathbb{N}$ in the range $2^{t-1} \leq m \leq 2^t - 1$ and $e \in \mathbb{Z}$ in the range $-2^{r-1} - 2 \leq e \leq 2^{r-1} - 1$ form together with 0 the set of **floating point numbers** F with precision t in base b with exponent range r. For a given floating point number x, the number $\pm \left(\frac{m}{b^t}\right)$ is called the **mantissa** and e is called the **exponent** of x.

Remark 2.7.2. The floating point number set F with which we are dealing is idealised in the sense that it ignores under- and overflow. These may occur e.g. when the absolute value of the result of an arithmetic operation is smaller or larger then the smallest or largest number that can be represented in F. In practice, normally an error is generated as no guarantees can be made any more with respect to the accuracy of the computed result.

In practice, most computer architectures natively support so-called IEEE single, double or quad(ruple) precision floating point numbers. In Table 2.1 we give the corresponding values for b, t, and r.

	base b	precision t	exponent range r	decimal digits
single precision	2	24	8	≈ 7.225
double precision	2	53	11	≈ 15.955
quad(ruple) precision	2	113	15	≈ 34.016

Table 2.1: IEEE single, double and quad(ruple) precision floating point numbers

The double precision format is used very commonly today, as it represents a good compromise between accuracy and speed. Unless stated otherwise all algorithms which are presented in this thesis were implemented in the ApCoCoA library using double precision arithmetic.

Example 2.7.3. [Floating point interval]

The interval $[1, 2]$ in IEEE double precision arithmetic is given by the following discrete subset of $[1, 2]$:

$$\left\{1, 1 + 2^{-52}, 1 + 2 \cdot 2^{-52}, 1 + 3 \cdot 2^{-52}, ..., 2\right\}.$$

So F contains $2^{52} + 1$ elements which lie in the interval $[1, 2]$. In general each interval $\left[2^j, 2^{j+1}\right]$ in F is given by the elements of $[1, 2]$ multiplied by 2^j (see also [5, pages 97-98]).

One remarkable feature of floating point numbers is that, in contrast to fixed point numbers, the absolute gap between two consecutive numbers becomes larger as the numbers themselves become bigger. The relative gap between two floating point numbers plays an important role in the stability analysis of algorithms. Ideally, the error introduced by a "stable" algorithm should be in the order of magnitude of this relative gap.

Definition 2.7.4. [Machine epsilon]

Let b be the base and let t be the precision of a set of floating point numbers F as defined above. Then we denote by

$$\varepsilon_{machine} = \frac{1}{2} b^{1-t}$$

the **machine epsilon** (or the **machine precision**) of F. If F is implemented by a certain computer architecture we also call $\varepsilon_{machine}$ the machine epsilon of this computer architecture.

In practice, the definition of $\varepsilon_{machine}$ may differ for a certain architecture because of specific implementation details but exact numbers are usually provided by the manufacturer. A device which implements the IEEE 754 standard ([47]) is guaranteed to provide $\varepsilon_{machine} = 2^{-24} \approx 5.96 \times 10^{-8}$ for single precision and $\varepsilon_{machine} = 2^{-53} \approx 1.11 \times 10^{-16}$ for double precision arithmetic.

Assumption 2.7.5 (Fundamental assumption of floating point arithmetic)**.** *Let $\star$ be an exact arithmetic operation on $\mathbb{C}$ (or $\mathbb{R}$) applied to the elements of F such as $+$, $-$, $\times$, or $\div$ and let $\circledast$ be its floating point counterpart. Then for all $x, y \in F$, there exists an $\varepsilon \in \mathbb{C}$ (or $\varepsilon \in \mathbb{R}$) with $|\varepsilon| \leq \varepsilon_{machine}$ such that*

$$x \circledast y = x \star y \, (1 + \varepsilon) \, .$$

Additionally, if $\hat{\sqrt{\cdot}}$ is the floating point counterpart of $\sqrt{\cdot}$, then for each $x \in F$ there exists an $\varepsilon \in \mathbb{C}$ (or $\varepsilon \in \mathbb{R}$) with $|\varepsilon| \leq \varepsilon_{machine}$ such that

$$\hat{\sqrt{x}} = \sqrt{x} \, (1 + \varepsilon)$$

is satisfied.

If Assumption 2.7.5 holds, it guarantees that all basic arithmetic operations have a relative error of at most $\varepsilon_{machine}$. For all our further considerations, we assume that the fundamental assumption of floating point arithmetic holds. This is at least true for all recent Intel and AMD computer architectures. Note that architectures exist for which Assumption 2.7.5 does not hold. However, theses systems only represent a shrinking minority. They require a different kind of stability analysis which is not covered in this thesis.

Remark 2.7.6. Let $x, y, z \in F$ and let $\circledast$ be a basic binary floating point operation. Please note that in general $(x \circledast y) \circledast z \neq x \circledast (y \circledast z)$.

Definition 2.7.7. Let $x, y \in F$. If we want to state explicitly that we are talking about basic floating point operations between x and y, e.g. when analysing the accuracy of an algorithm, we will use the notations $\hat{+}$, $\hat{-}$, $\hat{\times}$ and $\hat{\div}$. If no ambiguity can arise we will not make this distinction and use the common notation.

2.7.2 Condition of a Problem and Stability of Algorithms

When trying to give error estimates for numerical algorithms, there are basically two phenomena one has to pay attention to.
The first one is the **condition of a problem**. It measures by which order of magnitude small perturbations of the input data influence the solution of a problem. It is a property of the underlying mathematical problem and is independent of the algorithm used to attack it.
The second one is the **accuracy of an algorithm**. It describes how the rounding errors throughout an algorithm influence the computed solution of a problem.

Let us start with the discussion of the condition of a problem. We will only give a brief overview. Further details can, for example, be found in [5, Lecture 12]. For this purpose we introduce a few definitions which are commonly used in numerical linear algebra.

Definition 2.7.8. Let X and Y be normed finite dimensional vector spaces (most commonly $\mathbb{C}^m$ or $\mathrm{Mat}_{m,n}(\mathbb{C})$ together with a corresponding vector or matrix norm). A map $f : X \to Y$ is called a **problem**. The vector space X is also called **input space** and the vector space Y is called **solution space** of the problem. A problem f together with a data point $x \in X$ is called a **problem instance**. We call an element $\tilde{x} \in X$ an **ε-perturbation** of x if $\|x - \tilde{x}\| \leq \varepsilon \in \mathbb{R}_0^+$. If the actual value of ε does not play a role we say that $\tilde{x} \in X$ is a **perturbation** of x.

Definition 2.7.9. [Relative condition number]

Let $f : X \to Y$ be a map from a normed finite dimensional vector space X to a normed finite dimensional vector space Y, and let $x \in X$. Then the **(relative) condition number** $\kappa(x)$ of f at x is defined as

$$\kappa(x) = \lim_{\varepsilon \to 0^+} \sup_{\delta x \in X, \|\delta x\| \leq \varepsilon} \frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|}.$$

If it is clear from the context that we only consider **infinitesimal perturbations** of x, which means that $\|\delta x\| \leq \varepsilon$ for $\lim_{\varepsilon \to 0^+}$, we will abbreviate our notation by only writing:

$$\kappa(x) = \sup_{\delta x} \frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|}.$$

In case the norms on X and Y are (induced by) p-norms we write $\kappa_p(x)$ if we want to stress the underlying norm.

Remark 2.7.10. If f is differentiable we can write

$$\kappa(x) = \frac{\|J_f(x)\| \, \|x\|}{\|f(x)\|},$$

where $J_f(x)$ is the Jacobian matrix of $f(x)$. See [5, pages 90 and 91] for a more detailed explanation.

It is also possible to give a definition for an absolute condition number (see [5, Definition 12.1]). However, in numerical analysis the relative condition number plays a much more important role, as the error which is introduced by floating point arithmetic is a relative one. This is why we do not discuss the absolute condition number here.

Definition 2.7.11. We say that a problem $f : X \to Y$ is **well-conditioned** for an argument $x \in X$ if the associated relative condition number $\kappa(x)$ is small (e.g. smaller than 10^4) and **ill-conditioned** if the associated condition number is large (e.g. larger than 10^6). If $\kappa(x) = \infty$ we say that the problem f is **ill-posed** for x.

This is admittedly a rather fuzzy definition which leaves some room for interpretation. In practice, what can be viewed as a small and as a large condition number greatly depends on the context and the actual application. In accordance with [5, page 91] we have given the generally accepted bounds above.

Example 2.7.12. Let $f : \mathbb{C}^2 \to \mathbb{C}$ be the $\mathbb{C}$-linear map given by $f(x) = x_1 - x_2$ for $x = (x_1, x_2) \in \mathbb{C}^2$. Then the Jacobian matrix of f at a point x is given by

$$J_f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} & \frac{\partial f(x)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 & -1 \end{pmatrix}.$$

Using the ∞-norm on $\mathbb{C}^2$ and the corresponding induced matrix norm (the maximum of the absolute row sums) we obtain

$$\kappa(x) = \frac{\|J_f(x)\|_\infty \|x\|_\infty}{\|f(x)\|_\infty} = \frac{2\max\{|x_1|, |x_2|\}}{|x_1 - x_2|}.$$

The relative condition number can become very big if $|x_1 - x_2| \approx 0$. This means that if x_1 and x_2 have approximately the same value, the problem f is ill-conditioned.

Example 2.7.13. The problem of finding the eigenvalues of a non-normal matrix is in general also ill-conditioned as the following example will illustrate. The eigenvalues of the matrix

$$A = \begin{pmatrix} 1 & 1000 \\ 0.001 & 1 \end{pmatrix} \text{ are } (0, 2)$$

and the eigenvalues of the matrix

$$\tilde{A} = \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix} \text{ are } (1, 1).$$

As we can see, a slight perturbation of 0.001 in the lower left entry of the matrix has an unproportionally large influence on the result. Note that an actual bound on the stability of eigenvalues is presented later on in Theorem 2.9.1. See also Example 2.9.2 for further considerations.

We will now present a few well-known results which concern the stability of matrix-vector multiplication and the stability of solving a linear equation system.

Proposition 2.7.14 (Condition number of matrix-vector multiplication)**.**
Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, $x \in \mathbb{C}^n$, and let $\|\cdot\|$ be an arbitrary p-vector norm (with $1 \leq p \leq \infty$), which induces the corresponding matrix norm. The condition number κ of the linear map $f : \mathbb{C}^n \to \mathbb{C}^m$ given by $f(x) = Ax$ is

$$\kappa(x) = \frac{\|J_f(x)\| \|x\|}{\|f(x)\|} = \|A\| \frac{\|x\|}{\|Ax\|} \tag{2.1}$$

for $x \neq 0_n$ *and* $\kappa(x) = 1$ *for* $x = 0_n$. *Additionally, if* $m \geq n$ *and* A *has full rank* n, *the inequality*

$$\kappa(x) \leq \|A\| \, \|A^+\| \tag{2.2}$$

holds.

Proof. Equation 2.1 follows directly from the definition of the relative condition number if we observe that the Jacobian matrix of f is A. In the case $x = 0_n$, we obtain $\kappa(x) = 1$ as $\lim_{x \to 0_n} \left(\|A\| \frac{\|x\|}{\|Ax\|} \right) = \frac{\|A\|}{\|A\|} = 1$.
In order to prove inequality 2.2, let us assume that $m \geq n$ and A has full rank. First we will show that the inequality $\|x\| \leq \|A^+\| \, \|Ax\|$ holds, where A^+ is the pseudoinverse of A as defined in 2.6.1. For this purpose, let us recall Proposition 2.6.5, which states that $A^+A = I_n$ as the columns of A are linearly independent. Using the properties of the induced matrix norm (see Proposition 2.3.13) we conclude that $\|x\| = \|A^+Ax\| \leq \|A^+\| \, \|Ax\|$. Thus we have shown that

$$\kappa(x) \leq \|A\| \, \|A^+\|.$$

□

The advantage of the upper bound given by inequality 2.2 is its independence of the actual value of x. If $\|\cdot\| = \|\cdot\|_2$ equality is achieved if x is a scalar multiple of a right singular vector associated with a minimal singular value. Let $c \cdot x \in \mathbb{C}^n$ be a scalar multiple of a right singular vector (see Definition 2.5.34) x of A associated with a minimal singular value with $c \in \mathbb{C} \setminus \{0\}$. This means that $\kappa_2(x) = \|A\| \frac{\|cx\|}{\|Acx\|} = \|A\| \frac{|c|\|x\|}{|c|\|Ax\|} = \sigma_1(A) \frac{1}{\sigma_n(A)} = \|A\| \, \|A^+\|$.

As the product $\|A\| \, \|A^+\|$ shows up quite often in numerical linear algebra, it is customary to give it an own name. Thus we define the condition number of a matrix A as the upper bound 2.2 of the associated linear map $f : \mathbb{C}^n \to \mathbb{C}^m$ given by $f(x) = Ax$.

Definition 2.7.15. [Condition number of a matrix]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ be a matrix of full rank and let $\|\cdot\|$ be a matrix norm induced by a p-vector norm. Then we let

$$\kappa(A) = \|A\| \, \|A^+\|$$

and call it the **condition number** of A (with respect to $\|\cdot\|$). If A is rank deficient, we write $\kappa(A) = \infty$.

Example 2.7.16. If A is a unitary matrix, the associated condition number $\kappa(A)$ with respect to the Euclidean or Frobenius norm is 1 (compare Proposition 2.3.34). This is one of the reasons why most numerical algorithms are designed to use unitary transformations wherever possible.

Proposition 2.7.17. *Let* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ *be a matrix of full rank. In case* $\|\cdot\|$ *is the matrix norm induced by the Euclidean norm, then* $\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \geq 1$ *where* $\sigma_{\max}(A)$ *and* $\sigma_{\min}(A)$ *are the maximal and minimal singular values of* A.

Proof. Let $U\Sigma V^*$ be a SVD of A (see Definition 2.5.31). With the help of Proposition 2.3.34 and Proposition 2.6.2, we compute $\|A\| = \|U\Sigma V^*\| = \|\Sigma\| = \sigma_{\max}(A)$ and $\|A^+\| = \|V\Sigma^+U^*\| = \|\Sigma^+\| = \frac{1}{\sigma_{\min}(A)}$. Thus we can conclude that $\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$. As $\sigma_{\max}(A) \geq \sigma_{\min}(A)$ we observe that $\kappa(A) \geq 1$. □

Proposition 2.7.18 (Condition numbers of a system of linear equations)**.** *Let $A \in \mathrm{GL}_m(\mathbb{C})$, let $b \in \mathbb{C}^m$, and let $\|\cdot\|$ be a p-vector norm which induces a corresponding matrix norm with $1 \leq p \leq \infty$. Consider the system of linear equations $Ax = b$ with $x \in \mathbb{C}^m$.*

1. *For fixed A the map $f : \mathbb{C}^m \to \mathbb{C}^m$ given by $f(b) = A^{-1}b$ has condition number*

$$\kappa(b) = \left\|A^{-1}\right\| \frac{\|b\|}{\|A^{-1}b\|} = \left\|A^{-1}\right\| \frac{\|b\|}{\|x\|}$$

   if $b \neq 0_m$ and condition number $\kappa(b) = 1$ if $b = 0_m$. An upper bound for $\kappa(b)$ is given by $\kappa(b) \leq \|A\| \|A^+\| = \kappa(A)$.

2. *For fixed b the map $g : \mathrm{GL}_m(\mathbb{C}) \to \mathbb{C}^m$ given by $g(A) = A^{-1}b$ has a condition number of $\kappa(A)$.*

Proof. Let us start with the first claim. If we substitute A by A^{-1} in Proposition 2.7.14, the claim that $\kappa(b) = \left\|A^{-1}\right\| \frac{\|b\|}{\|A^{-1}b\|}$ and the upper bound given by $\kappa(A)$ immediately follow.

Next we look at the second claim and the map g. In order to obtain the condition number of g we need to deal with infinitesimal perturbations $\delta A \in \mathrm{Mat}_m(\mathbb{C})$ of A. Obviously if A is perturbed by δA also the solution x will be perturbed infinitesimally, which we will denote by $\delta x \in \mathbb{C}^m$. We obtain the equations

$$\begin{aligned}
(A + \delta A)(x + \delta x) &= b \\
Ax + (A + \delta A)\delta x + (\delta A)x &= b \\
(A + \delta A)\delta x + (\delta A)x &= 0 \\
(A + \delta A)\delta x &= -(\delta A)x.
\end{aligned}$$

As we are only considering infinitesimal perturbations of δA of A we may assume w.l.o.g. that $A + \delta A$ is invertible as well . This means that

$$\delta x = -(A + \delta A)^{-1}(\delta A)x$$

holds. We compute

$$\begin{aligned}
\|\delta x\| &= \|(A + \delta A)^{-1}(\delta A)x\| \\
\|\delta x\| &\leq \|(A + \delta A)^{-1}\| \|\delta A\| \|x\| \\
\frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A\|}{\|A\|} &\leq \|(A + \delta A)^{-1}\| \|A\| \\
\frac{\|x + \delta x - x\|}{\|x\|} \Big/ \frac{\|\delta A\|}{\|A\|} &\leq \|(A + \delta A)^{-1}\| \|A\| \\
\lim_{\varepsilon \to 0^+} \sup_{\|\delta A\| < \varepsilon} \frac{\|g(A + \delta A) - g(A)\|}{\|g(A)\|} \Big/ \frac{\|\delta A\|}{\|A\|} &\leq \lim_{\varepsilon \to 0^+} \sup_{\|\delta A\| < \varepsilon} \|(A + \delta A)^{-1}\| \|A\| \\
&= \|A^{-1}\| \|A\| = \kappa(A).
\end{aligned}$$

Equality is also attained for certain choices of δA, please compare [5, page 109]. As

$$\lim_{\varepsilon \to 0^+} \sup_{\|\delta A\| < \varepsilon} \frac{\|g(A + \delta A) - g(A)\|}{\|g(A)\|} \Big/ \frac{\|\delta A\|}{\|A\|}$$

matches the definition of the relative condition number this concludes the proof. □

Next we will deal with the error that is introduced by the use of floating point numbers in the algorithm itself. First of all we will explain what is generally understood by the term "stable" algorithm. Additionally, we will give a loose definition of what we mean by an actual numerical algorithm (compare also [5, page 102f.]).

Definition 2.7.19. [Algorithm]

Let X and Y be normed finite dimensional vector spaces (most commonly $\mathbb{C}^m$ or $\mathrm{Mat}_{m,n}(\mathbb{C})$ together with a corresponding vector or matrix norm) and let $f : X \to Y$ be a problem. Let $\tilde{f} : X \to Y$ be the map which sends an element $x \in X$ to the result $\tilde{f}(x)$ of applying an actual (floating point) implementation of the map f. We call $\tilde{f}$ an **algorithm** for the problem f. If $\tilde{f}(x) = f(x)$ for all $x \in X$ we say that $\tilde{f}$ is an **exact algorithm** for the problem f.

In the following let $f : X \to Y$ be a problem that maps elements of an (input) vector space X to a (solution) vector space Y. An algorithm for f given by an actual (floating point) computer implementation shall be denoted by $\tilde{f} : X \to Y$. Additionally, we will assume that the floating point implementation of the computer and therefore algorithm $\tilde{f}$ satisfies Assumption 2.7.5.

Remark 2.7.20. Following [5] we will use statements of the form

$$\|\text{computed quantity}\| \in O(\varepsilon_{machine}).$$

Intuitively this says that the norm of the "computed quantity", which can e.g. be a matrix or a vector computed by an algorithm $\tilde{f} : X \to Y$ for a problem $f : X \to Y$, is in the order of magnitude of the machine accuracy $\varepsilon_{machine}$. More precisely this means that if we consider the dimensions of X and Y as fixed the norm of the computed quantity can be uniformly bounded for all $x \in X$ by a constant expression $c \cdot \varepsilon_{machine}$ where $c \in \mathbb{R}^+$. Compare [5, pages 104-105] for further explanations.

Definition 2.7.21. [Stability]

An algorithm $\tilde{f} : X \to Y$ for a problem $f : X \to Y$ is called **stable** if for each $x \in X$ there exists an element $\tilde{x} \in X$ with $\frac{\|\tilde{x} - x\|}{\|x\|} \in O(\varepsilon_{machine})$ such that

$$\left\| \frac{\tilde{f}(x) - f(\tilde{x})}{f(\tilde{x})} \right\| \in O(\varepsilon_{machine}).$$

A slightly stronger version of stability is the so-called backward stability. Most algorithms which are used in numerical linear algebra satisfy this condition.

Definition 2.7.22. [Backward stability]

An algorithm $\tilde{f} : X \to Y$ for a problem $f : X \to Y$ is called **backward stable** if for each $x \in X$ there exists an element $\tilde{x} \in X$ with $\frac{\|\tilde{x}-x\|}{\|x\|} \in O(\varepsilon_{machine})$ such that the condition

$$\tilde{f}(x) = f(\tilde{x})$$

holds.

Informally, this means that the algorithm gives exactly the right answer to nearly the right question.

Remark 2.7.23. If Assumption 2.7.5 holds for a certain computer architecture all fundamental floating point operations are backward stable. See [5, pages 108 and 109] for a detailed explanation.

Proposition 2.7.24. *Let $\tilde{f}$ be a backward stable algorithm for a problem $f : X \to Y$ with associated condition number $\kappa(x)$ for $x \in X$. Furthermore, let us assume that Assumption 2.7.5 holds. Then the relation*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \in O(\kappa(x)\,\varepsilon_{machine})$$

is satisfied.

Proof. A proof is contained in [5, Theorem 15.1]. □

2.8 QR Decomposition via Householder Triangularisation

In this section we explain how a QR decomposition (2.5.12) of a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ can be computed in a numerically stable way via so-called Householder reflections.
The basic idea behind the Householder method for the calculation of the QR decomposition is to apply a series of unitary matrices $Q_i \in \mathrm{Mat}_m(\mathbb{C})$ to the matrix A in order to obtain an upper triangular matrix R. We want to form

$$\underbrace{Q_n \ldots . Q_2 Q_1}_{Q^*} A \;=\; R.$$

Each matrix Q_k is chosen in such a way that it introduces zeros below the diagonal of the k-th column and preserves the previously introduced zeros. So, to zero out the sub-diagonal entries of all n columns we have to apply n unitary matrices Q_i.

$$\underset{A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow[Q_1]{} \underset{Q_1 A}{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}} \xrightarrow[Q_2]{} \ldots . \xrightarrow[Q_n]{} \underset{Q_n \ldots . Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}$$

In order to leave the first $k-1$ columns unchanged, the matrix Q_k needs to have the following general structure

$$Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & F \end{pmatrix}$$

where $I_{k-1} \in \mathrm{Mat}_{k-1}(\mathbb{C})$ is the identity matrix and F is a special unitary matrix which we will now discuss in detail. If $x \in \mathbb{C}^{m-k+1}$ contains the lower $m-k+1$ entries of a column vector of A, namely $x = A_{k:m,k}$, and if we want to zero out all entries except the first one and preserve the norm of x, then F needs to perform the following operation:

$$x = \begin{bmatrix} \times \\ \times \\ \times \\ \times \\ \times \end{bmatrix} \xrightarrow{F} \begin{bmatrix} z\,\|x\| \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = z\,\|x\|\,e_1,$$

where $z \in \mathbb{C}$ satisfies $|z| = 1$ and e_1 is the first unit vector in $\mathbb{C}^{m-k+1}$. The map corresponding to F is supposed to reflect a point x along a hypersurface H such that $Fx = z\,\|x\|\,e_1$ (compare Figure 2.1). Theoretically every $z \in \mathbb{C}$ with $|z| = 1$ is suitable, however, in practice certain choices are numerically more stable than others. We will not discuss details here, but in general, one tries to determine z in such a way that $\|z \cdot \|x\| \cdot e_1 - x\|$ is maximised. This is achieved by letting $z = -\operatorname{sgn}(x_1)$. Please note that for non-zero complex numbers the function $\operatorname{sgn}(x)$ is defined as $\operatorname{sgn}(x) = \frac{x}{|x|}$. For $x = 0$ we let $\operatorname{sgn}(0) = 0$. Further information can be found in [6, Section 5.1.3].

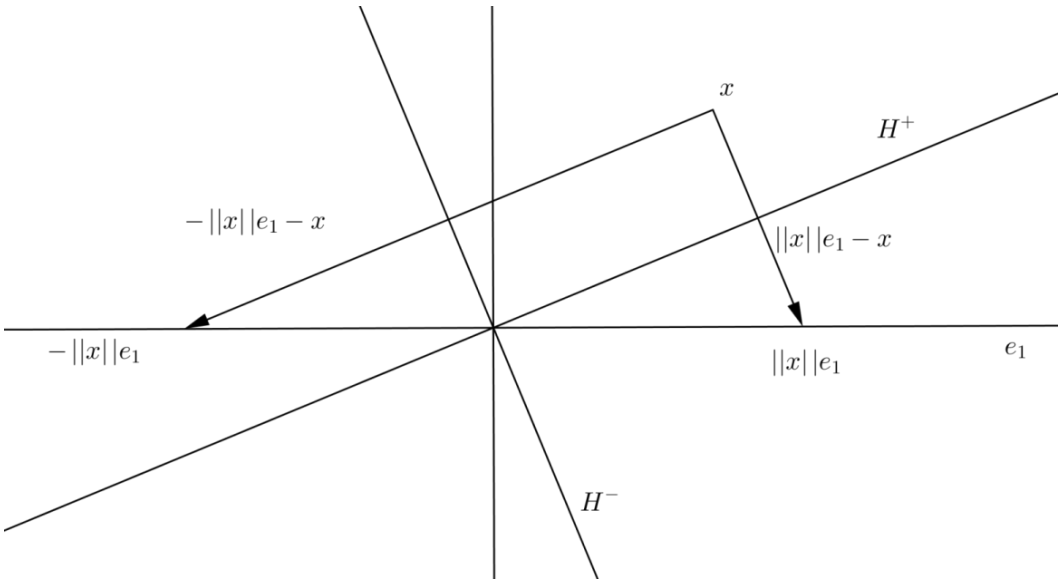

Figure 2.1: Two possible Householder reflectors in the real domain. Comp. [5, Fig. 10.2].

In accordance with [10], we introduce the following definitions:

Definition 2.8.1. [Householder reflection]

Let $x \in \mathbb{C}^m$. A **Householder reflection** (or **reflector**) w.r.t. to x is a linear transformation that represents a reflection along a hyperplane through the origin. It can be written in matrix form as

$$F = \left(I_m - 2\frac{vv^*}{v^*v} \right) \in \mathrm{Mat}_m(\mathbb{C})$$

with $v = \mathrm{sgn}(x_1) \|x\| e_1 + x$ if $v \neq 0_m$. In case $v = 0_m$ we let $F = I_m$. Please note, that $v \in \mathrm{Mat}_{m,1}(\mathbb{C})$ and therefore $vv^* \in \mathrm{Mat}_m(\mathbb{C})$ and $v^*v \in \mathbb{R}$.

Proposition 2.8.2 (Unitarity of reflectors)**.** *Householder reflectors are unitary and as a consequence also the matrices Q_k, which are of the form $Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & F \end{pmatrix}$ where $I_{k-1} \in \mathrm{Mat}_{k-1}(\mathbb{C})$ is the identity matrix and F represents a Householder reflection.*

Proof. We verify the claim by computing

$$\begin{aligned} F^*F &= \left(I_m - 2\frac{vv^*}{v^*v} \right)^* \left(I_m - 2\frac{vv^*}{v^*v} \right) = I_m - 2\frac{vv^*}{v^*v} - 2\frac{vv^*}{v^*v} + 4\frac{vv^*vv^*}{v^*vv^*v} \\ &= I_m - 4\frac{vv^*}{v^*v} + 4\frac{vv^*}{v^*v} = I_m. \end{aligned}$$

The matrices Q_k which contain the identity matrix together with F are unitary as well. □

Proposition 2.8.2 also becomes evident by a purely geometrical argument, as mirroring a point x twice on H will return the original point again, please compare Figure 2.1.

As a next step we encapsulate the procedure that computes the Householder reflector.

Algorithm 3: Householder

Input: A column vector $x \in \mathrm{Mat}_{m,1}(\mathbb{C})$
Output: An elementary reflector $v \in \mathrm{Mat}_{m,1}(\mathbb{C})$

```
1 v := sgn(x_1) ||x|| e_1 + x;
2 return v;
```

Now we can collect the results in the following algorithm.

Algorithm 4: QR decomposition

Input: A matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$
Output: The upper triangular matrix R of $A = QR$ and the elementary reflectors v_k which encode Q

```
1 for i := 1 to n do
2     v_i := householder(A_{i:m,i});
3     if v_i ≠ 0_{m-i+1} then
4         v_i := v_i / ||v_i||;
5         A_{i:m,i} := A_{i:m,i} − 2 v_i (v_i^* A_{i:m,i});
6     end
7 end
8 return (A, v_1, ..., v_n);
```

Theorem 2.8.3. *This is an algorithm which computes a QR decomposition of a matrix* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. *Upon termination of the algorithm the matrix* A *will be overwritten by an upper triangular matrix* R. *The corresponding unitary matrix* Q *is represented by its defining reflection vectors* $v_1, ..., v_n$ *(see Definition 2.8.1 and Remark 2.8.4) and is not formed explicitly. The algorithm has a runtime of* $O\left(2mn^2 - \frac{2}{3}n^3\right)$.

Proof. A proof and detailed discussion can be found in [5, Lecture 10]. □

Remark 2.8.4. The matrix Q is not computed directly as this is not necessary most of the time and also connected with additional cost. The effect of applying Q to a vector can also be achieved by using only the vectors v_k. We will now explain in detail how this can be achieved.

Recall that $Q^* = Q_n ... Q_1$ and $Q = Q_1 ... Q_n$ because all involved matrices are unitary.

Algorithm 5: Computation of Q^*x

Input: A vector $x \in \mathbb{C}^m$, elementary reflectors $v_1, ..., v_n$
Output: Q^*x

```
for i := 1 to n do
    x_{i:m} := x_{i:m} − 2v_i (v_i^* x_{i:m});
end
return x;
```

By reversing the order in which we apply the elementary reflectors we obtain an algorithm for computing Qx.

Algorithm 6: Computation of Qx

Input: A vector $x \in \mathbb{C}^m$, elementary reflectors $v_1, ..., v_n$
Output: Qx

```
for i := n downto 1 do
    x_{i:m} := x_{i:m} − 2v_i (v_i^* x_{i:m});
end
return x;
```

The advantage of not having calculated Q directly is that only $O(mn)$ operations are involved. If we would want to form the matrix Q explicitly we could, for example, compute QI_m via Algorithm 6. This means that we have to compute $Qe_1, ..., Qe_m$, so a total amount of $O\left(m^2 n\right)$ operations would be involved.

Another important result is that the computation of the QR decomposition via Householder reflectors is in fact backward stable.

Theorem 2.8.5. *Let* $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ *be given. If the QR decomposition of* $A = QR$ *is computed via Algorithm 4, then for the factors* $\tilde{Q}$ *and* $\tilde{R}$ *the relation*

$$\tilde{Q}\tilde{R} = A + \delta A$$

holds for some $\delta A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ *with* $\frac{\|\delta A\|}{\|A\|} \in O(\varepsilon_{machine})$. *This means that the algorithm is backward stable. Please note again that* $\tilde{Q}$ *is not formed explicitly in Algorithm 4, but it can be obtained with the help of Algorithm 6.*

Proof. Compare, for instance, the paper [11]. □

2.9 Computation of Eigenvalues and Eigenvectors

The heart of our algorithms will be the efficient and stable computation of eigenvalues and eigenvectors in floating point arithmetic. This topic has been extensively studied in the last 60 years and overviews are given, for example, in [5, Chapter 5], [6, Chapters 7/8], and [7, Chapters 4/5]. Basically, one can differentiate between two families of algorithms, namely those which can compute an eigendecomposition for arbitrary matrices and those who only work on Hermitian matrices. First we will give a general overview and explain which algorithms are particularly useful for us. Those will be discussed in more detail.

Name	Hermitian only	Full spectrum only	Eigenvalues	Eigenvectors
Inverse Iteration				x
QR-Algorithm		x	x	x
Divide & Conquer	x	x	x	x
Jacobi Method	x	x	x	x
Bisection method	x		x	

Table 2.2: Non-exhaustive overview of eigenvalue algorithms

2.9.1 The General Eigenvalue Problem

Almost all eigenvalue algorithms have in common that they consist in general of two phases. The first one is usually a preprocessing phase which can be carried out in a finite number of steps and which transforms the original matrix into a more structured form while preserving some or most of its properties. It is used primarily to speed up the following computation. The second phase, which would take theoretically an infinite number of steps but is stopped after the result has converged to near machine accuracy, reveals the eigenvalues.

Before we can start with presenting the actual algorithms we will begin with some standard results from eigenvalue perturbation theory.

Theorem 2.9.1 (Bauer-Fike). *Let* $A \in \mathrm{Mat}_m(\mathbb{C})$ *be diagonalisable, let* $\delta A \in \mathrm{Mat}_m(\mathbb{C})$ *be arbitrary, and let* $1 \leq p \leq \infty$. *Furthermore, let* VDV^{-1} *be an eigendecomposition of* A, *where* $V \in \mathrm{GL}_m(\mathbb{C})$ *and* $D \in \mathrm{Mat}_m(\mathbb{C})$ *is a diagonal matrix (compare Definition 2.5.18). If* μ *is an eigenvalue of* $A + \delta A$, *then there exists an eigenvalue* ν *of* A *such that the inequality*

$$|\nu - \mu| \leq \kappa_p(V) \|\delta A\|_p = \|V\|_p \|V^{-1}\|_p \|\delta A\|_p$$

holds.

Proof. Let $\mu \in \Lambda(A+\delta A)$. We will first consider the case where also $\mu \in \Lambda(A)$. Here we can choose $\nu = \mu$ so the theorem is trivially true. From now on let us assume that $\mu \notin \Lambda(A)$. This means that $\det(D-\mu I_m) \neq 0$ and $\det(A+\delta A-\mu I_m) = 0$. Then

$$\begin{aligned} 0 &= \det(A+\delta A-\mu I_m) = \det(V^{-1})\det(A+\delta A-\mu I_m)\det(V) \\ &= \det\left(D-V^{-1}\delta AV-\mu I_m\right) = \det(D-\mu I_m)\det\left((D-\mu I_m)^{-1}V^{-1}\delta AV+I_m\right). \end{aligned}$$

This means that $\det\left((D-\mu I_m)^{-1}V^{-1}\delta AV+I_m\right) = 0$ must hold. So

$$-1 \in \Lambda\left((D-\mu I_m)^{-1}V^{-1}\delta AV\right).$$

By Theorem 2.3.65, we know that

$$\begin{aligned} 1 &\leq \|(D-\mu I_m)^{-1}V^{-1}\delta AV\|_p \leq \|(D-\mu I_m)^{-1}\|_p\|V^{-1}\|_p\|\delta A\|_p\|V\|_p \\ &= \|(D-\mu I_m)^{-1}\|_p\|\delta A\|_p\kappa_p(V). \end{aligned}$$

Now, since $(D-\mu I_m)^{-1}$ is a diagonal matrix, it follows easily from the definition of the induced matrix norm $\|\cdot\|_p$ that

$$\begin{aligned} \|(D-\mu I_m)^{-1}\|_p &= \max\left\{\|(D-\mu I_m)^{-1}x\|_p \,\middle|\, x\in\mathbb{C}^n \text{ with } \|x\|_p = 1\right\} \\ &= \max_{v\in\Lambda(A)}\left(\frac{1}{|\nu-\mu|}\right) = \frac{1}{\min\limits_{v\in\Lambda(A)}|\nu-\mu|}. \end{aligned}$$

Therefore, we can conclude that $\min\limits_{v\in\Lambda(A)}|\nu-\mu| \leq \kappa_p(V)\|\delta A\|_p$, which proves the theorem. □

Example 2.9.2. Let us consider again Example 2.7.13 and let us briefly recall the setting. The eigenvalues of the matrix

$$A = \begin{pmatrix} 1 & 1000 \\ 0.001 & 1 \end{pmatrix} \text{ are } (0,2)$$

and the eigenvalues of the matrix

$$\tilde{A} = \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix} \text{ are } (1,1).$$

We note that matrix A is diagonalisable and we can thus apply Theorem 2.9.1. Furthermore $\tilde{A} = A+\delta A$ with $\delta A = \begin{pmatrix} 0 & 0 \\ -0.001 & 0 \end{pmatrix}$. We compute an eigendecomposition of $\tilde{A}$ such that $VDV^{-1} = \tilde{A}$. Let ν be an eigenvalue of $\tilde{A}$ then according to the theorem of Bauer-Fike there exists an eigenvalue μ of A, such that the bound $|\nu-\mu| \leq 1$ holds. This result is in line with the actual eigenvalues of A and $\tilde{A}$.

Remark 2.9.3. We know by Theorem 2.5.22 that a normal matrix A can be unitarily diagonalised. This means that in the theorem we can choose V satisfying $\|V\|_2 = \left\|V^{-1}\right\|_2 = \kappa_2(V) = 1$ and consequently we obtain

$$|\lambda-\mu| \leq \|\delta A\|_2.$$

Following [6, Subsection 7.2.4], we now present a theorem which concerns the sensitivity of eigenspaces and eigenvectors to perturbations of the input data. This result will play an important role in Chapter 4 when we analyse the stability of the computed solutions of the ABM and the extended ABM algorithm.

Definition 2.9.4. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and $B \in \mathrm{Mat}_n(\mathbb{C})$. Then we define the **separation** between both matrices as

$$\mathrm{sep}(A,B) = \min_X \frac{\|AX - XB\|_F}{\|X\|_F}$$

with $X \in \mathrm{Mat}_{m,n}(\mathbb{C}) \setminus \{0_{m,n}\}$.

Definition 2.9.5. Let S_1 and S_2 be linear subspaces of $\mathbb{C}^m$ such that $\dim(S_1) = \dim(S_2)$. Let furthermore P_1 be the orthogonal projector onto S_1 and let P_2 be the orthogonal projector onto S_2. Then we let

$$\mathrm{dist}(S_1,S_2) = \|P_1 - P_2\|_2$$

and call it the **distance** between S_1 and S_2.

Theorem 2.9.6. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let*

$$Q^*AQ = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}$$

be a Schur decomposition of A with $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$ (compare Definition 2.5.9). The involved matrices have the following dimensions: $Q_1 \in \mathrm{Mat}_{m,r}(\mathbb{C})$, $Q_2 \in \mathrm{Mat}_{m,m-r}(\mathbb{C})$, $T_{11} \in \mathrm{Mat}_{r,r}(\mathbb{C})$, $T_{12} \in \mathrm{Mat}_{r,m-r}(\mathbb{C})$, $T_{22} \in \mathrm{Mat}_{m-r,m-r}(\mathbb{C})$. Now let $\delta A \in \mathrm{Mat}_m(\mathbb{C})$ be an arbitrary matrix, which we partition via Q in the same way as A such that we obtain

$$Q^*\delta AQ = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}.$$

If $\mathrm{sep}(T_{11},T_{22}) > 0$ and

$$\|\delta A\|_2 \left(1 + \frac{5\|T_{12}\|_E}{\mathrm{sep}(T_{11},T_{22})}\right) \leq \frac{\mathrm{sep}(T_{11},T_{22})}{5}$$

then there exists $P \in \mathrm{Mat}_{m-r,r}(\mathbb{C})$ with

$$\|P\|_2 \leq \frac{4\|E_{21}\|_2}{\mathrm{sep}(T_{11},T_{22})}$$

such that the columns of $\hat{Q}_1 = (Q_1 + Q_2P)(I_r + P^*P)^{-\frac{1}{2}}$ form an orthonormal basis for an invariant subspace of $A + \delta A$. Additionally

$$\mathrm{dist}\left(\mathrm{im}(Q_1), \mathrm{im}(\hat{Q}_1)\right) \leq \frac{4\|E_{21}\|_2}{\mathrm{sep}(T_{11},T_{22})}$$

holds. Note, that the matrix $(I_r + P^*P)^{-\frac{1}{2}}$ is the inverse of the square root of the symmetric positive definite matrix $I_r + P^*P$ (see [9, Subsection 4.2.10]).

Proof. Compare [6, Theorem 7.2.4 and Corollary 7.2.5]. □

In case a subspace is one-dimensional, it is possible to give the following more specialised result.

Corollary 2.9.7. *If we let $r = 1$ and $T_{11} = \lambda$ in the setting of Theorem 2.9.6 we obtain the inequality*

$$\operatorname{dist}\left(\operatorname{im}\left(q_1\right), \operatorname{im}\left(\hat{q}_1\right)\right) \leq \frac{4\left\|E_{21}\right\|_2}{\sigma_{\min}\left(T_{22} - \lambda I_{m-1}\right)}.$$

Proof. The claim follows from Theorem 2.9.6 together with the observation that

$$\begin{aligned}\operatorname{sep}\left(T_{11}, T_{22}\right) &= \min_{x \neq 0_{m-1}} \frac{\left\|T_{11}x - xT_{22}\right\|_F}{\|x\|_F} = \min_{x \neq 0_{m-1}} \frac{\left\|x\left(\lambda I_{m-1} - T_{22}\right)\right\|_F}{\|x\|_F} \\ &= \min_{x \neq 0_{m-1}} \frac{\left\|x\left(T_{22} - \lambda I_{m-1}\right)\right\|_F}{\|x\|_F}.\end{aligned}$$

As x is a vector we can assume w.l.o.g. that $\|x\|_F = 1$, so we obtain

$$\operatorname{sep}\left(T_{11}, T_{22}\right) = \min_{x \neq 0_{m-1}} \left\|x\left(T_{22} - \lambda I_{m-1}\right)\right\|_F.$$

If x is a left-singular vector of $T_{22} - \lambda I_{m-1}$ associated with a minimal singular value the expression $\left\|x\left(T_{22} - \lambda I_{m-1}\right)\right\|_F$ becomes minimal and we arrive at

$$\operatorname{sep}\left(T_{11}, T_{22}\right) = \sigma_{\min}\left(T_{22} - \lambda I_{m-1}\right).$$

□

This result shows that the stability of computing eigenspaces and -vectors with respect to perturbations in the input data depends mostly on the initial separation of the subspaces together with the actual norm of the perturbation.

We will not discuss the algorithms which can be used to compute the eigenvalues and/or -vectors of a general matrix in detail. Of course they can also be applied to Hermitian matrices, however, more efficient algorithms exist for this special case. The most widely used algorithm today if **all** eigenvalues (and eigenvectors) of a dense unstructured matrix are desired is the QR algorithm and the Divide and Conquer algorithm if the input matrix is Hermitian.

The QR Algorithm

The basic idea behind the QR algorithm is to use the QR decomposition of a matrix $A = QR$ and to multiply the factors in reverse order RQ. This has the consequence that the entries below the diagonal decrease normwise while the matrix product RQ remains similar to A. The procedure is repeated until the entries below the diagonal are below a specified tolerance. Thus a Schur decomposition of the matrix is computed. To accelerate the process the input matrix is first transformed via similarity transformations to so-called upper Hessenberg form which is as close to upper triangular form as can be achieved with a finite number of computation steps.

Definition 2.9.8. [Hessenberg form]

A square matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is said to be in **upper Hessenberg form** if it has only zero entries below the first subdiagonal. Consequently, we say a square matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is in **lower Hessenberg form** if it has only zero entries above the first superdiagonal.

Algorithm 7: Householder Hessenberg reduction

Input: A square matrix $A \in \mathrm{Mat}_m(\mathbb{C})$

Output: A matrix in upper Hessenberg form which is unitarily similar to A and the basic reflectors

1 **for** *i := 1* **to** *m-2* **do**
  `// Householder reflectors are computed via Algorithm 3`
2 $v_i := \mathrm{householder}(A_{i+1:m,i})$;
3 **if** $v_i \neq 0_{m-i}$ **then**
4 $A_{i+1:m,i:m} := \left(I_m - 2\frac{v_i v_i^*}{v_i^* v_i}\right) A_{i+1:m,i:m}$;
5 $A_{1:m,i+1:m} := A_{1:m,i+1:m}\left(I_m - 2\frac{v_i v_i^*}{v_i^* v_i}\right)$;
6 **end**
7 **end**
8 **return** $(A, v_1, ..., v_{m-2})$;

Theorem 2.9.9. *Given a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$, the algorithm returns a matrix in upper Hessenberg form which is unitarily similar to A. Additionally, the algorithm is backward stable.*

Proof. This follows directly from the properties of the Householder reflectors which are constructed in such a way that they introduce zeros below the first subdiagonal when applied to A in line 5. The already introduced zeros remain untouched by line 6, which makes sure that the transformation is in fact a similarity transformation. The algorithm is backward stable because all applied similarity transformations are unitary. □

Now that we know how to transform a given matrix to upper Hessenberg form it is possible to state a basic version of the QR algorithm.

Algorithm 8: Basic QR algorithm

Input: A square matrix $A \in \mathrm{Mat}_m(\mathbb{C})$, error tolerance $\varepsilon \in \mathbb{R}^+$

Output: The m eigenvalues of A

1 $[H, U] := \mathrm{HessenbergReduction}(A)$ (e.g. via Algorithm 7);
2 **while** $\sum_{i=1}^{m}\sum_{j=i+1}^{m} |H_{j,i}| > \varepsilon$ **do**
3 $[Q, R] := \mathrm{QRDecomposition}(H)$ (e.g. via Algorithm 4);
4 $H := RQ$;
5 **end**
6 **return** $(H_{1,1}, ..., H_{m,m})$;

Theorem 2.9.10. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be such that A has no two distinct eigenvalues with equal absolute value. The Basic QR algorithm computes a Schur decomposition of A such that $A = (UQ)\,H\,(UQ)^*$, where $Q \in \mathrm{Mat}_m(\mathbb{C})$ and $U \in \mathrm{Mat}_m(\mathbb{C})$ are unitary matrices and $H \in \mathrm{Mat}_m(\mathbb{C})$ is an upper triangular matrix. Thus H is (unitarily) similar to A and reveals the eigenvalues of A on its diagonal.*

Proof. We will only sketch why this algorithm produces a series of matrices H_i which "essentially" converges (i.e. the elements on the diagonal converge while the super-diagonal elements may differ by units in each iteration) to an upper triangular matrix which is similar to A. A full proof can be found in [13, Section 11]. First the matrix A is transformed into Hessenberg form; the resulting matrix H is similar to A. The central steps of the algorithm are computing the QR decomposition of H and then multiplying both factors in reverse order. Each computed matrix H is unitarily similar to A as $H_i = R_{i-1}Q_{i-1} = Q_{i-1}^* H_{i-1} Q_{i-1}$ with H_i, Q_i, and R_i denoting the values of H, Q, and R during the i-th iteration of the while loop. □

Theorem 2.9.11. *The Basic QR algorithm is backward stable.*

Proof. The claim follows essentially from the fact that only unitary transformations are used to compute the Hessenberg form of the input matrix, followed by a sequence of unitary similarity transformations to compute the solution. A more detailed analysis can, for instance, be found in [6, Subsection 7.5.6]. □

In practice more advanced versions of the algorithm are used which achieve faster convergence by applying shifts during each step of the computation and by only implicitly computing the QR decomposition in each iteration step. One such algorithm, the Francis QR algorithm, is discussed and analysed in [6, Subsection 7.5.6]. The algorithm requires about $10m^3$ flops if only the eigenvalues are desired and the unitary transformations are not accumulated, otherwise it requires about $25m^3$ flops. Additionally, state of the art implementations of the QR algorithm (e.g. the one in LAPACK [17]) guarantee convergence to a Schur decomposition for essentially all input matrices $A \in \mathrm{Mat}_m(\mathbb{C})$.

Power Iteration

The technique of power iteration is in itself not often applied directly, but the ideas underlying it form the basis for more advanced techniques like inverse iteration. It is capable to compute the eigenvector associated with the eigenvalue of a matrix A which has the largest absolute value. Let us denote the eigenvalues of a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ by $\lambda_1, ..., \lambda_m$ and let us further assume without loss of generality that $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_m|$. The algorithm only works properly if $|\lambda_1|$ is reasonably larger than $|\lambda_2|$.

Algorithm 9: Power Iteration

Input: A matrix $A \in \mathrm{Mat}_m(\mathbb{C}), n \in \mathbb{N}$

Output: An eigenvector estimate corresponding to the eigenvalue λ_1 of A

```
v^(0) := random vector in C^m \ {0_m};
v^(0) := v^(0) / ||v^(0)||;
for i := 1 to n do
    v^(i) := A v^(i-1);
    v^(i) := v^(i) / ||v^(i)||;
end
return v^(n);
```

Theorem 2.9.12. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$. If $|\lambda_1| > |\lambda_2|$ and if $q_1^* v^{(0)} \neq 0$, meaning that the initial guess has components in the direction of q_1, where q_1 is the eigenvector associated with λ_1, then this algorithm produces a sequence $v^{(i)}$ of eigenvector estimates for q_1. The following bound for the iterates $v^{(i)}$ can be established:*

$$\left\| v^{(i)} - \left(e^{i\theta_i} q_1 \right) \right\| \in O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^i \right)$$

for some $\theta_i \in]-\pi, \pi]$ and $i \in \mathbb{N}$.

Proof. We only present the proof for the case that A is diagonalisable as this is the situation most relevant to us. A proof for general matrices can be found in [16, Theorem 4.1]. Let $q_1, ..., q_m$ be an orthonormal basis of eigenvectors with corresponding eigenvalues $\lambda_1, ..., \lambda_m$ for A. Let us additionally assume that $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_m|$. Then we can express $v^{(0)}$ as a linear combination of those basis vectors such that

$$v^{(0)} = \sum_{k=1}^{m} c_k q_k$$

with $c_k \in \mathbb{C}$. Now for some constant $n_i \in \mathbb{R}$ which arises because of the normalisation in each iteration we obtain

$$\begin{aligned} v^{(i)} &= n_i A^i v^{(0)} \\ &= n_i \left(\sum_{k=1}^{m} c_k \lambda_k^i q_k \right) \\ &= n_i \lambda_1^i \left(c_1 q_1 + c_2 \left(\frac{\lambda_2}{\lambda_1} \right)^i q_2 + \sum_{k=3}^{m} c_k \left(\frac{\lambda_k}{\lambda_1} \right)^i q_k \right). \end{aligned}$$

A direct consequence of this equation is that our eigenvector estimate will converge linearly to a multiple of q_1 depending on the ratio $\left| \frac{\lambda_2}{\lambda_1} \right|$, meaning $\left\| v^{(i)} - \left(e^{i\theta_i} q_1 \right) \right\| \in O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^i \right)$ for some $\theta_i \in]-\pi, \pi]$. □

Remark 2.9.13. For sufficiently large values of i we obtain

$$\left\| v^{(i)} - \left(e^{i\theta_i} q_1 \right) \right\| \in O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^i \right)$$

where $e^{i\theta_i} \approx \left(\frac{\lambda_1}{|\lambda_1|} \right)^i$. For instance, if λ_1 is real and positive this means that $v^{(i)}$ converges to q_1.

Remark 2.9.14. If $A \in \mathrm{Mat}_m(\mathbb{R})$ the condition that $|\lambda_1| > |\lambda_2|$ implies that $\lambda_1 \in \mathbb{R}$.

Proof. Let $A \in \mathrm{Mat}_m(\mathbb{R})$ and let us assume that $\lambda_1 \in \mathbb{C} \setminus \mathbb{R}$ is a complex eigenvalue of A. Then we know that $\lambda_2 = \bar{\lambda}_1$ has to be another complex eigenvalue of A. This would imply that $|\lambda_1| = |\lambda_2|$ which contradicts our assumptions. □

More details about the algorithm and applications can, for example, be found in [5, Lecture 27] or in [6, Section 8.2.1].

Inverse Iteration

Power iteration has two significant shortcomings. One disadvantage is that it is only capable of finding the eigenvector corresponding to the largest eigenvalue and additionally its convergence rate largely depends on the ratio of $|\lambda_1/\lambda_2|$. We will now discuss how Algorithm 9 can be modified to provide an effective method to determine the eigenvectors of a matrix if good estimates of the eigenvalues are known in advance. This method is particularly useful if only a subset of the eigenvectors is needed. This could, for instance, be the eigenvector corresponding to the smallest eigenvalue.

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be nonsingular and let $\lambda \in \mathbb{C}$ be an eigenvalue estimate for $\tilde{\lambda} \in \Lambda(A)$ such that $\lambda \notin \Lambda(A)$. First we observe that $\det(A - \lambda I_m) \neq 0$, which implies that the matrix $A - \lambda I_m$ is invertible. Then the eigenvectors of A associated with $\tilde{\lambda}$ are the same as the eigenvectors of $(A - \lambda I_m)^{-1}$ which correspond to the eigenvalue $(\tilde{\lambda} - \lambda)^{-1}$ of $(A - \lambda I_m)^{-1}$. In order to prove this claim let $x \in \mathbb{C}^m$ be an eigenvector of A which is associated with the eigenvalue $\tilde{\lambda}$. Then the equation $Ax = \tilde{\lambda} x$ holds and additionally

$$\begin{aligned} (A - \lambda I_m)^{-1} (A - \lambda I_m)\, x = x \quad &\Longleftrightarrow \\ (A - \lambda I_m)^{-1} (\tilde{\lambda} - \lambda) x = x \quad &\Longleftrightarrow \\ (A - \lambda I_m)^{-1} x = (\tilde{\lambda} - \lambda)^{-1} x. \end{aligned}$$

As we know, the rate of convergence for the power iteration algorithm is about $\left| \frac{\lambda_2}{\lambda_1} \right|$. If we choose λ to be an eigenvalue estimate the gap between the largest and second largest eigenvalue of $(A - \lambda I_m)^{-1}$ will broaden tremendously and in this way accelerate convergence. So the essential idea behind inverse iteration is to apply power iteration to the inverse of $A - \lambda I_m$. Note that the matrix $(A - \lambda I_m)^{-1}$ is not computed explicitly. We rather solve a system of linear equations in each round which is more economic. Obviously, the matrix $(A - \lambda I_m)^{-1}$ becomes more ill-conditioned the closer λ gets to an exact eigenvalue of A. Fortunately though, the

occurring error has a dominant component in the direction of the true eigenvector. A more detailed explanation of this behaviour can be found in [5, Lecture 27 and Algorithm 27.2].

Algorithm 10: Inverse Iteration

Input: A matrix $A \in \mathrm{Mat}_m(\mathbb{C})$, an eigenvalue estimate $\lambda \in \mathbb{C}$ of A, $n \in \mathbb{N}$
Output: An eigenvector estimate corresponding to the eigenvalue λ of A

1 $v^{(0)} :=$ random vector in $\mathbb{C}^m \backslash \{0_m\}$;
2 $v^{(0)} := \frac{v^{(0)}}{\|v^{(0)}\|}$;
3 **for** *i := 1* **to** n **do**
4 | Solve $(A - \lambda I)\, v^{(i)} = v^{(i-1)}$ for $v^{(i)}$;
5 | $v^{(i)} := \frac{v^{(i)}}{\|v^{(i)}\|}$;
6 **end**
7 **return** $v^{(n)}$;

In practice the input matrix is transformed to Hessenberg form first, for example, via Algorithm 7. As this is a common preprocessing technique used also in the QR algorithm, most of the time the Hessenberg form is readily available without additional cost.

If the input matrix is in Hessenberg form the cost of inverse iteration is in $O\left(m^2\right)$ per eigenvector, which is essentially the cost for solving the system of linear equations in line 4 (compare [6, Section 7.6.1]).

Reduction to (upper) bidiagonal form

Before we can start, we will first state what we mean by a bidiagonal matrix.

Definition 2.9.15. [Bidiagonal matrix]

We say a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is **upper bidiagonal** if all its entries below the diagonal and above the first superdiagonal are zero. Consequently, a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is called **lower bidiagonal** if all its entries above the diagonal and below the first subdiagonal are zero.

The following algorithm reduces a matrix to upper bidiagonal form and is used, for instance, as a first preprocessing step inside the Golub-Kahan algorithm for the computation of a SVD of a matrix. Given a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, the algorithm computes a series of $2n-2$ Householder transformations $U_1, V_1, U_2, V_2, \ldots, V_{n-2}, U_{n-1}, U_n$ and applies them alternately to the left and right-hand side of A such that $U_B^* A V_B = (U_1 \ldots U_n)^* A (V_1 \ldots V_{n-2})$ is upper bidiagonal, with $U_i \in \mathrm{Mat}_m(\mathbb{C})$ and $V_i \in \mathrm{Mat}_n(\mathbb{C})$.

Algorithm 11: Golub-Kahan Bidiagonalisation

Input: A matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$

Output: The matrix A in upper bidiagonal form and the generating Householder reflectors

for $i := 1$ **to** n **do**
  $u_i :=$ householder$(A_{i:m,i})$ (e.g. via Algorithm 3);
  if $u_i \neq 0_{m-i+1}$ **then** $A_{i:m,i:n} := \left(I_{m-i+1} - 2\frac{u_i u_i^*}{u_i^* u_i}\right) A_{i:m,i:n}$;
  if $i < n-1$ **then**
    $v_i :=$ householder$\left(A^*_{i,i+1:n}\right)$;
    if $v_i \neq 0_{n-1}$ **then** $A_{i:m,i+1:n} := A_{i:m,i+1:n}\left(I_{n-i} - 2\frac{v_i v_i^*}{v_i^* v_i}\right)$;
  end
end
return $(A, u_1, ..., u_n, v_1, ..., v_{n-2})$;

2.9.2 The Hermitian Eigenvalue Problem

The Hermitian (or symmetric) eigenvalue problem deserves special treatment, as a number of algorithms exist which exploit its structure and therefore achieve greater numerical stability and/or speed. We first collect some theoretical results with respect to the properties and stability of eigenvalues and eigenvectors for Hermitian matrices.

Proposition 2.9.16. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix. Then all eigenvalues $\lambda \in \Lambda(A)$ are real.*

Proof. We know that Hermitian matrices are normal. Thus we can apply the spectral theorem (2.5.22) which states that a unitary matrix $U \in \mathrm{Mat}_m(\mathbb{C})$ and a diagonal matrix $D \in \mathrm{Mat}_m(\mathbb{C})$ must exist such that $A = UDU^*$. As $A = A^*$ we compute

$$\begin{aligned} UDU^* &= UD^*U^* \iff \\ D &= D^*. \end{aligned}$$

Because D is diagonal and $D = D^*$ must hold, we can conclude that D must be a real matrix. The fact that D contains the eigenvalues of A on its diagonal concludes the proof. □

As Hermitian matrices only possess real eigenvalues, we adapt the convention to order them in a descending way in this section. So let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix, by $\lambda_k(A)$ we denote the k-th eigenvalue of A assuming that the m eigenvalues of A are ordered descending with respect to their values, meaning that $\lambda_1(A) \geq ... \geq \lambda_m(A)$.

Theorem 2.9.17 (Weyl's theorem). *If $A \in \mathrm{Mat}_m(\mathbb{C})$ and $\delta A \in \mathrm{Mat}_m(C)$ are Hermitian matrices, then the inequalities*

$$\lambda_k(A) + \lambda_m(\delta A) \leq \lambda_k(A + \delta A) \leq \lambda_k(A) + \lambda_1(\delta A)$$

hold for $1 \leq k \leq m$.

Proof. Compare [9, Theorem 4.3.1]. □

Corollary 2.9.18. *If $A \in \mathrm{Mat}_m(\mathbb{C})$ and $\delta A \in \mathrm{Mat}_m(\mathbb{C})$ are Hermitian matrices, then*

$$|\lambda_k(A+\delta A) - \lambda_k(A)| \leq \|\delta A\|_2 .$$

Proof. First we observe that $\max\{|\lambda_1(\delta A)|, |\lambda_m(\delta A)|\} = \|\delta A\|_2$. If we use this together with Theorem 2.9.17, we obtain

$$|\lambda_k(A+\delta A) - \lambda_k(A)| \leq |\lambda_1(\delta A)| \leq \max\{|\lambda_1(\delta A)|, |\lambda_m(\delta A)|\} = \|\delta A\|_2 .$$

□

Now, we present a result which concerns the stability of the singular values of a matrix with respect to perturbations.

Corollary 2.9.19. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $\delta A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. By $\sigma_k(M)$ we denote the k-th singular value of a matrix $M \in \mathrm{Mat}_{m,n}(\mathbb{C})$, assuming that we have ordered the singular values of M in a descending way such that $\sigma_1(M) \geq ... \geq \sigma_{\min(m,n)}(M)$. For every $1 \leq k \leq \min(m,n)$, the inequality*

$$|\sigma_k(A+\delta A) - \sigma_k(A)| \leq \|\delta A\|_2$$

holds.

Proof. A proof can be found in [6, Corollary 8.6.2]. □

The following result sheds some light on the stability of the eigenvectors of Hermitian matrices. We will use it later on to prove some bounds about the stability of the computed result of the ABM and the extended ABM algorithms, which is closely connected with the homogeneous least squares problem (2.10.2).

Definition 2.9.20. Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix, and let $\lambda_1 \geq ... \geq \lambda_m$ be its eigenvalues. By

$$\mathrm{gap}_i(A) = \min_{j \neq i} |\lambda_i - \lambda_j|$$

we denote the **gap** between the i-th eigenvalue and the other eigenvalues of the matrix A.

Theorem 2.9.21. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$, $\delta A \in \mathrm{Mat}_m(\mathbb{C})$, and $\tilde{A} = A + \delta A$ be Hermitian matrices. Now let $A = Q\Lambda Q^*$ and $\tilde{A} = \tilde{Q}\tilde{\Lambda}\tilde{Q}^*$ be eigendecompositions of A and $\tilde{A}$, respectively. Furthermore, let us denote the column vectors of Q and $\tilde{Q}$ by $q_1, ..., q_m$ and $\tilde{q}_1, ..., \tilde{q}_m$ such that $Q = (q_1, ..., q_m)$ and $\tilde{Q} = (\tilde{q}_1, ..., \tilde{q}_m)$. If θ_i denotes the angle between q_i and $\tilde{q}_i$, i.e. $\cos(\theta_i)\|q_i\|\|\tilde{q}_i\| = |\langle q_i, \tilde{q}_i\rangle|$, and $\mathrm{gap}_i\left(\tilde{A}\right) \neq 0$, then*

$$\frac{1}{2}\sin(\theta_i) \leq \frac{\|\delta A\|_2}{\mathrm{gap}_i\left(\tilde{A}\right)}$$

for all $1 \leq i \leq m$.

Proof. Compare [7, Theorem 5.4]. □

Remark 2.9.22. If $\frac{\|\delta A\|_2}{\|A\|_2} \in O(\varepsilon_{machine})$ (see Remark 2.7.20) in the setting of Theorem 2.9.21, then

$$\frac{1}{2}\sin(\theta_i) \in O\left(\frac{\varepsilon_{machine}\|A\|_2}{\mathrm{gap}_i\left(\tilde{A}\right)}\right) = O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2}{\mathrm{gap}_i\left(\tilde{A}\right)}\right).$$

This means that if we fix the dimension of the matrix m a constant $c \in \mathbb{R}^+$ exists such that

$$\frac{1}{2}\sin(\theta_i) \leq c\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2}{\mathrm{gap}_i\left(\tilde{A}\right)}.$$

For larger values of m the constant c will also increase. Therefore, if m becomes too big we will no longer be able to extract any useful information from the inequality.

An important property of Hermitian matrices is that the eigenvalues of the submatrices "interlace".

Proposition 2.9.23. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a Hermitian matrix, let $v \in \mathrm{Mat}_{m,1}(\mathbb{C})$ be a row vector, and let $a \in \mathbb{R}$. Then the matrix*

$$\tilde{A} = \begin{pmatrix} A & v \\ v^* & a \end{pmatrix} \in \mathrm{Mat}_{m+1}(\mathbb{C})$$

is Hermitian and the eigenvalues of A and $\tilde{A}$ interlace, which means that

$$\lambda_1(\tilde{A}) \geq \lambda_1(A) \geq \lambda_2(\tilde{A}) \geq \ldots \geq \lambda_m(\tilde{A}) \geq \lambda_m(A) \geq \lambda_{m+1}(\tilde{A}).$$

Proof. See [9, Theorem 4.3.8.]. □

Now we are ready to start with the actual algorithms. Similarly like in the case of general matrices, it is common to divide the process of computing the eigenvalues of a Hermitian matrix into two phases. First an algorithm is executed which significantly reduces the cost of the following phase in which the actual eigenvalues are computed. Following [6, Section 8.3] we first present a phase one algorithm which reduces a Hermitian matrix to tridiagonal form via a sequence of similarity transformations.

Reduction to tridiagonal form

First, we will introduce the necessary definitions.

Definition 2.9.24. [Tridiagonal matrix]

We say a matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ is **tridiagonal** if all its entries below the first subdiagonal and above the first superdiagonal are zero.

One possibility to transform a given Hermitian matrix A via similarity transformations into real symmetric tridiagonal form is to apply a sequence of Householder reflections. This is an economic way if the matrix is dense and has no special structure besides being Hermitian.

Algorithm 12: Householder tridiagonalisation

Input: A Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$

Output: The subdiagonal, diagonal, and superdiagonal entries of A are overwritten with its real symmetric tridiagonal form, the $m-2$ Householder reflectors which transform A into tridiagonal form

1 **for** *i := 1* **to** *m-2* **do**
  `// Householder reflectors are computed via Algorithm 3`
2 $v_i := \mathrm{householder}(A_{i+1:m,i})$;
3 **if** $v_i \neq 0_{m-i}$ **then**
4 $\beta := \frac{2}{v_i^* v_i}$;
5 $p := \beta A_{i+1:m,i+1:m} v_i$;
6 $w := p - (\beta p^* v_i/2)\, v_i$;
7 $A_{i+1,i} := \|A_{i+1:m,i}\|$;
8 $A_{i,i+1} := A_{i+1,i}$;
9 $A_{i+1:m,i+1:m} := A_{i+1:m,i+1:m} - v_i w^* - w v_i^*$;
10 **end**
11 **end**
12 $A_{m,m-1} := \|A_{m,m-1}\|$; $A_{m-1,m} := A_{m,m-1}$;
13 **return** $(A, v_1, ..., v_{m-2})$;

Theorem 2.9.25. *This algorithm transforms a given Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ into real symmetric tridiagonal form (up to floating point accuracy). Upon termination the subdiagonal, diagonal, and superdiagonal entries of A are overwritten with its real symmetric tridiagonal form. Additionally, the Householder reflectors which transform A into tridiagonal form are returned. The cost is in $O\left(\frac{4}{3}m^3\right)$.*

Proof. Please compare [6, Section 8.3.1] for a proof and a more detailed description. □

However, if the matrix we are dealing with is sparse there are more efficient ways to compute a suitable similarity transformation. One way to do this is to use the Lanczos algorithm, which we now briefly explain. Further details can, for example, be found in [5, Section 36].

Algorithm 13: Lanczos Algorithm

Input: A Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$

Output: A basis transformation matrix $Q = (q_1, ..., q_m) \in \mathrm{Mat}_m$ that transforms A into tridiagonal form, the entries of the tridiagonal matrix

$\beta_0 := 0$, $q_0 := 0_m \in \mathbb{C}^m$, $b :=$ random vector in $\mathbb{C}^m \setminus \{0_m\}$, $q_1 := b/\|b\|$;
for *i := 1* **to** m **do**
  $v := Aq_i - \beta_{i-1}q_{i-1}$;
  $\alpha_i := q_i^* v$;
  $v := v - \beta_{i-1}q_{i-1} - \alpha_i q_i$;
  $\beta_i := \|v\|$;
  $q_{i+1} := v/\beta_i$;
end
return $(q_1, ..., q_m, \alpha_1, ..., \alpha_m, \beta_1, ..., \beta_{m-1})$;

Theorem 2.9.26. *This is an algorithm which computes a basis transformation*

$$(q_1, ..., q_m) = Q \in \mathrm{Mat}_m(\mathbb{C})$$

such that

$$Q^*AQ = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \alpha_{m-1} & \beta_{m-1} \\ 0 & \cdots & 0 & \beta_{m-1} & \alpha_m \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{R})$$

is a real symmetric tridiagonal matrix.

Proof. A proof can be found in [6, Section 9.2]. □

This algorithm exploits the Hermitian structure of the input matrix A to construct a basis transformation $Q = (q_1, ..., q_m)$ such that Q^*AQ is real symmetric tridiagonal. One of the major advantages of the algorithm is that it can make good use of sparsity in A unlike an algorithm based on Householder reflections. The main loop is executed m times. Inside the loop the most expensive operation is a matrix-vector multiplication, namely Aq_i, followed by an inner product computation and a few vector operations, which do not contribute significantly to the runtime of the Lanczos algorithm. So, if A is sparse and we exploit the sparseness in the matrix-vector multiplication, the Lanczos algorithm is about one order of magnitude faster than algorithms built around Householder reflections. However, the Lanczos algorithm as stated above is numerically less stable. Special care has to be taken to make sure that the q_i remain reasonably orthogonal. For further details see [12, Section 9.2ff].

The Divide & Conquer Algorithm

Another algorithm for the symmetric eigenvalue problem, which has become popular in recent years, is the so-called divide & conquer algorithm. According to [7, Section 5.3.6], it is the fastest

currently available algorithm if all eigenvalues **and** eigenvectors are desired and if the dimension of the matrix exceeds about 25. Please note that especially for this method details which we will not present here, because they would exceed the scope of this thesis, are essential for the numerical stability. Thus the description given here is merely meant to present the essential ideas behind the algorithm. A detailed discussion can, for instance, be found in [7, Section 5.3.3].

Proposition 2.9.27 (Decoupling). *Suppose that* $T \in \mathrm{Mat}_m(\mathbb{C})$ *is of the form*

$$T = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}$$

where T_{11}, T_{12}, *and* T_{22} *are block matrices. Then* $\Lambda(T) = \Lambda(T_{11}) \cup \Lambda(T_{22})$ *and we say that the problem of computing the eigenvalues of* T *can be* ***decoupled****.*

Proof. A proof is contained in [6, Lemma 7.1.1]. □

Given a Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ with $m \geq 2$, it is first transformed into real symmetric tridiagonal form, e.g. using Algorithm 12. Let us denote this matrix by

$$T = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{m-1} \\ & & b_{m-1} & a_m \end{pmatrix}.$$

Let us additionally assume w.l.o.g. that T has no zero entries in its off-diagonal, i.e. $b_i \neq 0$ for all $1 \leq i \leq m-1$. Otherwise we could just decouple the problem (see Proposition 2.9.27 and also [7, page 221]) until we arrive at submatrices with the required property. Now let us denote by $1 \leq n < m$ a natural number which we use to split T such that we obtain the following decomposition:

$$T_1 = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n - b_n \end{pmatrix}, \qquad T_2 = \begin{pmatrix} a_{n+1} - b_n & b_{n+1} & & \\ b_{n+1} & a_{n+2} & \ddots & \\ & \ddots & \ddots & b_{m-1} \\ & & b_{m-1} & a_m \end{pmatrix}$$

and

$$T = \begin{pmatrix} T_1 & \\ & T_2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & & & \\ 0 & \ddots & \ddots & & \\ & \ddots & b_n & b_n & \\ & & b_n & b_n & \ddots \\ & & & \ddots & 0 \\ & & & & 0 \end{pmatrix} = \begin{pmatrix} T_1 & \\ & T_2 \end{pmatrix} + b_n v v^{\mathrm{tr}}$$

with $v = \begin{pmatrix} 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 \end{pmatrix}^{\mathrm{tr}}$. We have divided the problem into two sub problems T_1 and T_2 which can now be treated recursively with the same strategy. However, we still need to explain how the eigenvalues of T are related to the eigenvalues of T_1 and T_2. Now let us assume that we have already computed the eigenvalue decomposition of T_1 and T_2 such that $T_i = Q_i \Lambda_i Q_i^{\mathrm{tr}}$. Then we have

$$\begin{aligned} T &= \begin{pmatrix} T_1 & \\ & T_2 \end{pmatrix} + b_n v v^{\mathrm{tr}} = \begin{pmatrix} Q_1 \Lambda_1 Q_1^{\mathrm{tr}} & \\ & Q_2 \Lambda_2 Q_2^{\mathrm{tr}} \end{pmatrix} + b_n v v^{\mathrm{tr}} \\ &= \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \left(\begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix} + b_n u u^{\mathrm{tr}} \right) \begin{pmatrix} Q_1^{\mathrm{tr}} & \\ & Q_2^{\mathrm{tr}} \end{pmatrix} \end{aligned}$$

with

$$u = \begin{pmatrix} Q_1^{\mathrm{tr}} & \\ & Q_2^{\mathrm{tr}} \end{pmatrix} v = \begin{pmatrix} \text{last column of } Q_1^{\mathrm{tr}} \\ \text{first column of } Q_2^{\mathrm{tr}} \end{pmatrix}.$$

Now $\begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{R})$ is a diagonal matrix which we abbreviate by D. We name the entries on the diagonal of D with $d_1, ..., d_m$ in such a way that $d_1 \geq ... \geq d_m$. Next, let us investigate the shape of the characteristic polynomial of $D + b_n u u^{\mathrm{tr}}$. We let $\lambda \in \mathbb{C}$ such that $D - \lambda I$ is nonsingular. Then

$$\det\left(D + b_n u u^{\mathrm{tr}} - \lambda I_m\right) = \det\left((D - \lambda I_m)\left(I_m + b_n (D - \lambda I_m)^{-1} u u^{\mathrm{tr}}\right)\right).$$

As we have assumed that $D - \lambda I_m$ is nonsingular, the number λ is an eigenvalue of $D + b_n u u^{\mathrm{tr}}$ if

$$\det\left(I_m + b_n (D - \lambda I_m)^{-1} u u^{\mathrm{tr}}\right) = 0.$$

Let $x, y \in \mathbb{C}^m$. From Sylvester's determinant theorem (see [14, Corollary 18.1.2]) it follows that $\det\left(I_m + x y^{\mathrm{tr}}\right) = \det\left(1 + y^{\mathrm{tr}} x\right) = 1 + y^{\mathrm{tr}} x$. Therefore, we have

$$\begin{aligned} \det\left(I_m + b_n (D - \lambda I_m)^{-1} u u^{\mathrm{tr}}\right) &= 1 + b_n u^{\mathrm{tr}} (D - \lambda I_m)^{-1} u \\ &= 1 + b_n \sum_{i=1}^{m} \frac{u_i^{\mathrm{tr}} u_i}{d_i - \lambda} =: f(\lambda). \end{aligned}$$

Now the eigenvalues of T are exactly the roots of the so-called **secular equation** $f(\lambda) = 0$. As with all general eigenvalue algorithms it is of course not possible to give a closed form solution to this equation as this may involve determining the roots of a polynomial with degree greater than 4. If we interpret $f(\lambda) : \mathbb{R} \to \mathbb{R}$ as a real function we observe that the $\lambda = d_i$ are vertical asymptotes and $y = 1$ is a horizontal asymptote. By investigating the derivative of $f(\lambda)$ which is given by $f'(\lambda) = b_n \sum_{i=1}^{m} \frac{u_i^{\mathrm{tr}} u_i}{(d_i - \lambda)^2}$ it becomes evident that $f(\lambda)$ is monotonic and smooth on the intervals $]d_i, d_{i+1}[$. Because of this there has to be exactly one zero in each interval, which can relatively easily be found with the help of iterative techniques. Commonly a special version of the Newton-Raphson algorithm (see [7, pages 221 and 222]) is used which guarantees convergence in $O(m)$ flops per eigenvalue which is essentially the cost for evaluating $f(\lambda)$ and $f'(\lambda)$. Thus it costs $O\left(m^2\right)$ to find all m eigenvalues of T.

We now explain how to compute the eigenvectors of $D + b_n u u^{\mathrm{tr}}$. Let α be an eigenvalue of $D + b_n u u^{\mathrm{tr}}$. As

$$\begin{aligned}\left(D + b_n u u^{\mathrm{tr}}\right)(D - \alpha I_m)^{-1} u &= \left(D - \alpha I_m + \alpha I_m + b_n u u^{\mathrm{tr}}\right)(D - \alpha I_m)^{-1} u \\ &= u + \alpha (D - \alpha I_m)^{-1} u + u\left(1 - 1 + b_n u^{\mathrm{tr}} (D - \alpha I_m)^{-1} u\right) \\ &= u + \alpha (D - \alpha I_m)^{-1} u - u + f(\alpha) \\ &= \alpha (D - \alpha I_m)^{-1} u\end{aligned}$$

holds, we observe that $(D - \alpha I_m)^{-1} u$ is in fact the eigenvector corresponding to α. Because the matrix $D - \alpha I_m$ is diagonal, its inverse can be efficiently computed and the total cost per eigenvector is in $O(m)$. So it costs $O\left(m^2\right)$ to compute all m eigenvectors of T. What remains to be done is to have a closer look at the runtime of the whole algorithm. Let us assume that the matrix is always split in half (as far as possible) in each step and both eigenvalues and eigenvectors are computed. Then we obtain the following crude estimate for the runtime $t(m)$ for a matrix of dimension m. Namely $t(m) = 2t(m/2) + O\left(m^2\right) + O\left(m^2\right) + m^3$. Using the so-called master theorem (see [8, Theorem 4.1]) one can show that this accumulates to a runtime of $t(m) \approx \frac{4}{3} m^3$. Reduction to tridiagonal form takes another $\frac{8}{3} m^3$ flops. Thus in total the runtime of the algorithm is $4m^3$ which is a distinct improvement over the symmetric QR algorithm which has a runtime of $9m^3$ flops.

Algorithm 14: Divide&ConquerEV

Input: A Hermitian tridiagonal matrix $T \in \mathrm{Mat}_m(\mathbb{C})$

Output: The m eigenvalues of T stored in Λ, and the corresponding eigenvectors Q

1 **if** $T \in \mathrm{Mat}_1(\mathbb{C})$ **then return** $(\Lambda := T, Q = 1)$;

2 $[T_1, T_2, b_n] := \mathrm{Decompose}(T)$;

3 $[\Lambda_1, Q_1] := \mathrm{Divide\&ConquerEV}(T_1)$;

4 $[\Lambda_2, Q_2] := \mathrm{Divide\&ConquerEV}(T_2)$;

5 $M := \begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix} + b_n u u^{\mathrm{tr}}$;

6 Compute eigenvalues Λ and eigenvectors Q' of M using e.g. the Newton-Raphson algorithm;

7 $Q := \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} Q'$;

8 **return** (Λ, Q);

Bisection

The method which we now briefly discuss is very useful if only a small subset of eigenvalues and/or vectors are required. As we will see later, this case is especially relevant to us in Chapter 4, where we explain how the ABM and extended ABM algorithms can be implemented efficiently. Once again we do not describe all details. They can be found, for example, in [6, Section 8.5].

A given Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ with $m \geq 2$, is first transformed into real symmetric tridiagonal form, for instance, using Algorithm 12. As a next step we can use decoupling to obtain submatrices which only have non-zero entries on its subdiagonals and superdiagonals. Later on we can just unify the spectra of these submatrices. Let us denote one of theses decoupled matrices by

$$T = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix},$$

with $1 \leq n \leq m$. By $T^{(1)} \in \mathrm{Mat}_1(\mathbb{R}), ..., T^{(n)} \in \mathrm{Mat}_n(\mathbb{R})$ we denote the upper-left submatrices of size $1, ..., n$. First we observe that the eigenvalues of all submatrices are real, because they are all real and symmetric (compare Proposition 2.9.16). Via essentially the same arguments we used to investigate the secular equation in the Divide & Conquer eigenvalue algorithm, one can show that the eigenvalues of two subsequent matrices $T^{(k)}$ and $T^{(k+1)}$ strictly interlace (compare Proposition 2.9.23 together with [5, Exercise 25.1]), meaning that

$$\lambda_j^{(k+1)} < \lambda_j^{(k)} < \lambda_{j+1}^{(k+1)}$$

for $1 \leq k \leq n-1$ and $1 \leq j \leq k-1$. As $\det(T)$ is the product of all eigenvalues of T, it is now possible to count the number of negative eigenvalues of T, i.e. the number of eigenvalues in the interval $]-\infty; 0]$, by counting the number of sign changes of the Sturm sequence $\det\left(T^{(0)}\right), \det\left(T^{(1)}\right), ..., \det\left(T^{(m)}\right)$, if we define $\det\left(T^{(0)}\right) = 1$. By subtracting aI_n from the matrix T, we can thus determine the number of eigenvalues in the interval $]-\infty; a]$. If we are now interested in the number of eigenvalues in the interval $[a, b[$, with $a < b$, we can subtract the number of eigenvalues in $]-\infty; a]$ from $]-\infty; b]$.

What remains to be shown is that $\det\left(T^{(k)} - aI_n\right)$ can be computed efficiently. Of course $\det\left(T^{(1)} - aI_n\right) = a_1 - a$ but we note additionally that

$$\det\left(T^{(k)} - aI_n\right) = (a_k - a)\det\left(T^{(k-1)}\right) - b_{k-1}^2 \det\left(T^{(k-2)}\right)$$

for $2 \leq k \leq n$, which follows directly from Laplace's formula for determinants. Thus the cost of the evaluation is in $O(n)$ and it takes $O(n \log(\varepsilon_{machine}))$ to locate one eigenvalue with relative accuracy $\varepsilon_{machine}$. Consequently, if only a small subset of eigenvalues of a matrix is required this is better than the $O\left(n^2\right)$ operations which are needed by other algorithms like the QR or the Divide & Conquer algorithm.

Example 2.9.28. Let $T = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$. We are interested in the eigenvalues in the interval $[2.5, 3.5[$ up to a relative accuracy of 0.5. First we observe that T is already in tridiagonal form. We shift T by $3.5I_2$ and obtain $T_a = \begin{pmatrix} -2.5 & 2 \\ 2 & -2.5 \end{pmatrix}$. Now $\det\left(T_a^{(1)}\right) = -2.5$ and $\det\left(T_a^{(2)}\right) = 2.25$, so the Sturm sequence $(1, -2.5, 2.25)$ has two sign changes and we know that two eigenvalues are smaller than 3.5. We repeat this process for $T_b = \begin{pmatrix} -1.5 & 2 \\ 2 & -1.5 \end{pmatrix}$ and

obtain the sequence $(1, -1.5, -1.75)$ which means that one eigenvalue is smaller than 2.5. Thus there is one eigenvalue of T in the interval $[2.5, 3.5[$. We repeat this procedure by investigating the intervals $[2.5, 3[$ and $[3, 3.5[$. T_3 yields the sequence $(1, -2, 0)$ which means that there is one eigenvalue in $[3, 3.5[$. Because the last entry is zero we can even conclude that 3 must have been an eigenvalue and it is the only eigenvalue in the interval $[3, 3.5[$.

Algorithm 15: Eigenvalues via Bisection

Input: A symmetric tridiagonal matrix $T \in \mathrm{Mat}_m(\mathbb{R})$, an interval $[a, b[$, and an error tolerance ε

Output: The eigenvalues in $[a, b[$ of T stored in Λ

```
1  n_a := numSmallerEigenvalues(T, a);
2  n_b := numSmallerEigenvalues(T, b);
3  Λ := [ ];
4  if n_a = n_b then return Λ ;
5  IntervalList := [a, n_a, b, n_b];
6  while IntervalList ≠ ∅ do
7  |  [low, n_low, up, n_up] = first(IntervalList);
8  |  remove first element form IntervalList;
9  |  mid := (low+up)/2;
10 |  if up - low < ε then
11 |  |  Add(Λ, mid);
12 |  else
13 |  |  n_mid := numSmallerEigenvalues(T, mid);
14 |  |  if n_mid > n_low then Add(IntervalList, [low, n_low, mid, n_mid]) ;
15 |  |  if n_up > n_mid then Add(IntervalList, [mid, n_mid, up, n_up]) ;
16 |  end
17 end
18 return Λ;
```

2.10 The (Linear) Least Squares Problem

One of the problems that we are dealing with in this thesis and specifically in Chapter 4 is that we want to find polynomials which need not interpolate a given set of points (measurements), but are allowed to pass "close by" to them. In this way we can get rid of noise which is contained in the points (measurements), which we will use as input for our algorithms. The output of the algorithms will contain polynomials that have the above mentioned property.
The starting point is commonly an overdetermined linear system of equations which has no exact solution. One natural approach to constructing an approximate solution for such a system of equations was already discovered by C. F. Gauss in 1794 (see [2]).

Definition 2.10.1. [The linear least squares problem]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $b \in \mathbb{C}^m \setminus \{0_m\}$ with $m \geq n$. The task of finding vectors $x \in \mathbb{C}^n$ such that $\|Ax - b\|_2$ is minimised is called a **linear least squares problem**. For a specific solution x we call the vector $r = Ax - b$ the **residual (with respect to x)**. As we demand that $b \neq 0_m$ the problem is sometimes also referred to as an **inhomogeneous least squares problem**.

If we are dealing with $b = 0_m$ a solution to the problem is always provided by $x = 0_m$. In this setting it makes sense that we additionally require that $\|x\|_2 = c \in \mathbb{R}^+$ to rule out the trivial solution. So if we fix $c \in \mathbb{R}^+$, we can then answer the question which norm c vector x minimises $\|Ax\|_2$. This will play an important role in the ABM algorithm, which we will present in Section 4.3.

Definition 2.10.2. [The homogeneous least squares problem]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ and $c \in \mathbb{R}^+$. The task of finding vectors $x \in \mathbb{C}^n$ with $\|x\| = c$ such that $\|Ax\|_2$ is minimised is called a **homogeneous least squares problem**.

The homogeneous least squares problem will be discussed in Section 2.13.

The following example demonstrates the least squares fitting approach.

Example 2.10.3. [Polynomial least squares fitting]

A standard example where least squares is commonly used is (univariate) polynomial curve fitting. Given s distinct points $p_1, ..., p_s$ in $\mathbb{C}$ and additionally s numbers $b_1, ..., b_s$ in $\mathbb{C}$ it is always possible to find an interpolating polynomial $g \in \mathbb{C}[x]$ of degree at most $s-1$ such that $g(p_i) = b_i$ holds for all $1 \leq i \leq s$. However, sometimes it may be desirable or needed to utilise polynomials only up to a certain degree $d < s-1$. Of course, in general no polynomial of degree at most d will exist such that $g(p_i) = b_i$ is satisfied for all $1 \leq i \leq s$. In this case a least squares approach can be used to determine the coefficients $c_i \in \mathbb{C}$ of $g(x) = \sum_{i=0}^{d} c_i x^i$ such that $\sqrt{\sum_{j=1}^{s} |g(p_j) - b_j|^2}$ is minimal. This is equivalent to computing $c = (c_0, ..., c_d) \in \mathbb{C}^{d+1}$ such that

$$\|V_d c - b\|_2 = \left\| \begin{pmatrix} 1 & p_1 & p_1^2 & \cdots & p_1^d \\ 1 & p_2 & p_2^2 & \cdots & p_2^d \\ 1 & p_3 & p_3^2 & \cdots & p_3^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_s & p_s^2 & \cdots & p_s^d \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_s \end{pmatrix} \right\|_2$$

is minimised. The matrix V_d is called the **Vandermonde matrix** for degree d with respect to the points $(p_1, ..., p_s)$. As we can see, the least squares solution delivers the desired coefficients $c_0, ..., c_d$ of the polynomial g. Note that the solution is unique because the points p_1 to p_s are distinct and therefore the matrix V_d has full rank $d+1$, which according to Theorem 2.11.1 guarantees uniqueness.

There are different methods for the solution of the linear least squares problem. Below we present two commonly used methods, of which one can be used for the inhomogeneous and the other one for the homogeneous problem.

2.11 Solutions of the Inhomogeneous Least Squares Problem

In the setting of Definition 2.10.1, it is intuitively clear that we need to compute a vector x such that the residual vector r is orthogonal on the image of A. We will now substantiate this intuition and provide a few equivalent characterisations that can be used to compute the actual solution(s).

Theorem 2.11.1. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $b \in \mathbb{C}^m \setminus \{0_m\}$ with $m \geq n$ be given. A vector $x \in \mathbb{C}^n$ solves the linear least squares problem of minimizing $\|b - Ax\|_2$ if and only if $(b - Ax) \perp \mathrm{im}(A)$. By $r \in \mathbb{C}^m$ we will denote the residual, such that $r = b - Ax$ and by $P_A \in \mathrm{Mat}_m(\mathbb{C})$ we will denote the orthogonal projector onto $\mathrm{im}(A)$ (see Theorem 2.3.41). The following statements are equivalent:*

$$A^*r = 0_n \tag{2.3}$$
$$A^*Ax = A^*b \tag{2.4}$$
$$Ax = P_Ab. \tag{2.5}$$

Furthermore, the solution x is unique if and only if A has full rank n.

Proof. Equation 2.3 encodes that r is orthogonal on the image of A (see Definition 2.3.29 and Remark 2.3.2). The equivalence of Equation 2.3 and 2.4 follows from

$$\begin{aligned} A^*r = 0_n &\iff \\ A^*(b - Ax) = 0_n &\iff \\ A^*b - A^*Ax = 0_n &\iff \\ A^*Ax = A^*b. & \end{aligned}$$

In order to prove the equivalence of Equation 2.4 and 2.5 we use that $r \perp \mathrm{im}(A)$ and therefore the equation $P_Ar = 0_m$ needs to hold. We further compute

$$\begin{aligned} P_Ar = 0_m &\iff \\ P_A(b - Ax) = 0_m &\iff \\ P_Ab - P_AAx = 0_m &\iff \\ P_Ab - Ax = 0_m &\iff \\ Ax = P_Ab. & \end{aligned}$$

Next, we show that there is no vector $z \in \mathrm{im}(A)$ with $z \neq P_Ab$, which minimises $\|b - z\|_2$. For this purpose, let us denote P_Ab by y. If we assume that such a z exists, then $z - y \in \mathrm{im}(A)$ and consequently $(z - y) \perp r$ as well as $(z - y) \perp (b - y)$ need to be satisfied. Using the Pythagorean theorem we observe that $\|b - z\|_2^2 = \|y - z\|_2^2 + \|b - y\|_2^2 > \|b - y\|_2^2$, which is a contradiction to the assumption that $\|b - z\|_2$ is minimal for z. What remains to be shown is that x is unique if and only if A has full rank. This follows from Equation 2.5 in combination with Proposition 2.3.42. We know that the matrix P_A is uniquely determined by A and therefore P_Ab has to be unique as well. As $P_Ab \in \mathrm{im}(A)$ the system of linear equations $Ax = P_Ab$ has a unique solution for x if and only if A has full rank. □

If A has full rank, then A^*A has full rank n and it follows from Equation 2.4 that A^*A can be inverted and (the unique) solution x can be computed as

$$x = (A^*A)^{-1} A^*b.$$

Proposition 2.11.2 (Computation pseudoinverse). *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ and $\mathrm{rank}(A) = n$. Then the pseudoinverse A^+ of A is given by*

$$A^+ = (A^*A)^{-1} A^*.$$

Proof. Let $U\Sigma V^*$ be a singular value decomposition of A. We will show that the matrix $(A^*A)^{-1} A^* \in \mathrm{Mat}_{n,m}(\mathbb{C})$ is equal to $V\Sigma^+U^*$ (compare Proposition 2.6.2). We compute

$$\begin{aligned}
(A^*A)^{-1} A^* &= \left(V\Sigma^{\mathrm{tr}}U^*U\Sigma V^*\right)^{-1} V\Sigma^{\mathrm{tr}}U^* \\
&= \left(V\Sigma^{\mathrm{tr}}\Sigma V^*\right)^{-1} V\Sigma^{\mathrm{tr}}U^* \\
&= V\left(\Sigma^{\mathrm{tr}}\Sigma\right)^{-1} V^*V\Sigma^{\mathrm{tr}}U^* \\
&= V\left(\Sigma^{\mathrm{tr}}\Sigma\right)^{-1} \Sigma^{\mathrm{tr}}U^* \\
&= V\Sigma^+U^*.
\end{aligned}$$

□

Now we know that the full rank linear least squares problem can be solved via the computation of the pseudoinverse as $x = A^+b$.

Proposition 2.11.3. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ and $b \in \mathbb{C}^m \setminus \{0_m\}$ be given and let $A = QR$ be a reduced QR factorisation of A (see Definition 2.5.13). Then $x \in \mathbb{C}^n$ is a solution of the inhomogeneous least squares problem if*

$$Rx = Q^*b$$

holds. Furthermore, if A has full rank, then x can be computed as

$$x = R^{-1}Q^*b.$$

Thus the pseudoinverse A^+ of A is given by $R^{-1}Q^*$.

Proof. We can write $Ax = P_Ab$ where P_A is the orthogonal projector onto $\mathrm{im}(A)$. From Theorem 2.3.41 we know that $P_A = QQ^*$ so we conclude that

$$\begin{aligned}
Ax = QQ^*b \quad &\Longleftrightarrow \\
QRx = QQ^*b.&
\end{aligned}$$

As Q contains an orthonormal basis of $\mathrm{im}(A)$ as its columns (compare Definition 2.5.13 and Remark 2.5.17) we can left multiply with Q^* and obtain

$$Rx = Q^*b.$$

If we know additionally that A has full rank, then R can be inverted, which concludes the proof. □

Remark 2.11.4. If Algorithm 4 is used to solve the inhomogeneous least squares problem $\min_x \|Ax - b\|_2$, the total cost amounts to $O\left(2mn^2 - \frac{2}{3}n^3\right)$. First the reduced QR factorisation $A = QR$ is computed in $O\left(2mn^2 - \frac{2}{3}n^3\right)$. In the next step we compute Q^*b with Algorithm 5 in $O(mn)$. And finally, we solve the upper triangular system $Rx = Q^*b$ for x in $O\left(\frac{1}{2}n(n+1)\right)$.

2.12 Conditioning of the Least Squares Problem

In this section we briefly discuss the conditioning of the least squares problem. This will be important later on when we try to devise algorithms that make sure that the underlying least squares problems we are trying to solve, e.g. in the extended ABM algorithm (24), are well posed. More details on this topic can, for example, be found in [5, Lecture 18] or in [6, Section 5.3.7].

Let us start out with the standard setting of the linear least squares problem. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ be of full rank (otherwise there is no unique solution and we define the condition number as ∞) and let $b \in \mathbb{C}^m \setminus \{0_m\}$ be given. We want to compute $x \in \mathbb{C}^n$ such that $\|Ax - b\| = \|y\|$ is minimised. As we have seen in the previous section $x = A^+b$ and $y = P_Ab$ where $A^+ \in \mathrm{Mat}_{n,m}(\mathbb{C})$ is the pseudoinverse (see Definition 2.6.1) of A and where $P_A = QQ^* = AA^+$ is the orthogonal projector onto $\mathrm{im}(A)$ (compare Theorem 2.3.41). The whole situation in $\mathbb{R}^2$ is depicted in Figure 2.2. We now state the effects of perturbations in A and b on x and y as given and also proved in [5, Theorem 18.1].

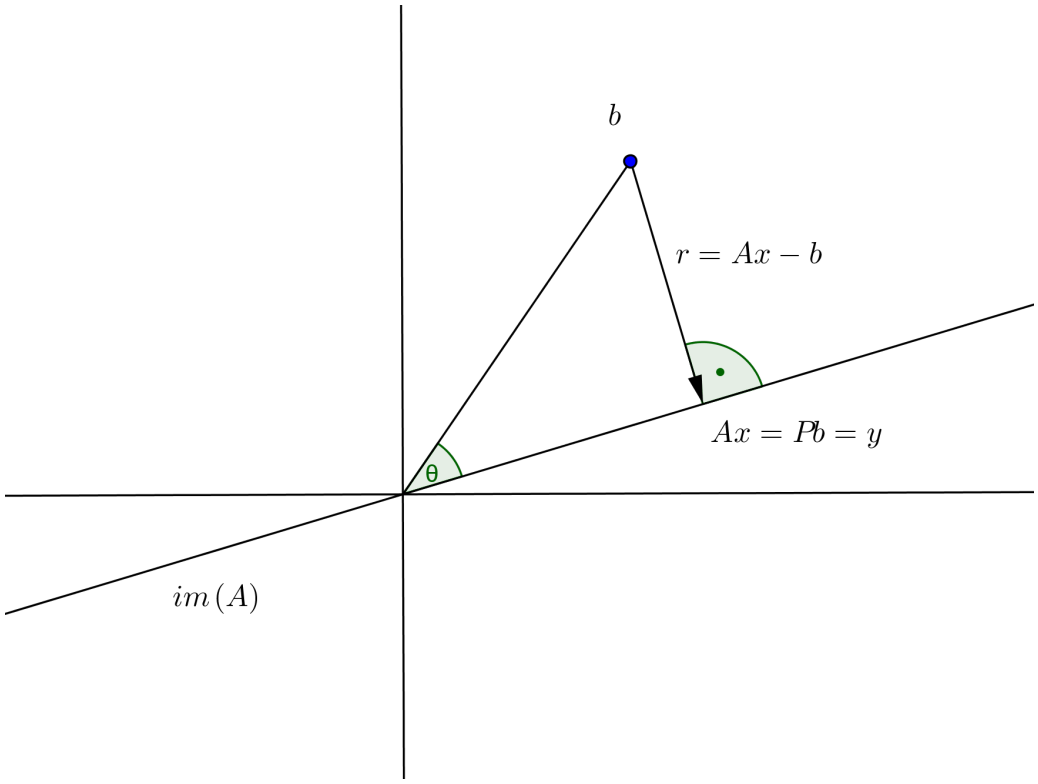

Figure 2.2: Geometry of the least squares problem

Theorem 2.12.1 (Conditioning of the linear least squares problem). *Given A and b as stated above, the linear least squares problem has the following $\|\cdot\|_2$ condition numbers:*

Input \ *Output*	x	y
A	$\kappa(A) + \frac{1}{\eta}\kappa(A)^2 \tan(\theta)$	$\frac{\kappa(A)}{\cos(\theta)}$
b	$\frac{\kappa(A)}{\eta\cos(\theta)}$	$\frac{1}{\cos(\theta)}$

where $\kappa(A)$ is the 2-norm condition number of A, $\theta = \cos^{-1}\left(\frac{\|y\|}{\|b\|}\right)$ and $\eta = \frac{\|A\|\|x\|}{\|Ax\|}$. The table has to be read in the following way: e.g. if we keep b fixed and vary A the condition number of computing y is given by $\frac{\kappa(A)}{\cos(\theta)}$.

We only investigate the sensitivity of x under (infinitesimal) perturbations of A as this is the case most relevant to us. This will be crucial when investigating the stability of the result computed by the extended ABM algorithm (24). All other proofs can be found in [5, Theorem 18.1].

Proof. Let $U\Sigma V^*$ be a SVD of A (compare Definition 2.5.31). As U and V are unitary transformations they do not influence the matrix norm induced by the 2-norm (see Propositions 2.3.33 and 2.3.34), which means that $\|A\|_2 = \|U\Sigma V^*\|_2 = \|\Sigma\|_2$. So for our analysis w.l.o.g. we may assume that

$$A = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_n \\ 0 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} A_1 \\ 0 \end{bmatrix}.$$

Please note that this means that we can restrict ourselves to studying a real diagonal matrix A with non-zero diagonal entries. When perturbing A by $\delta A \in \mathrm{Mat}_{m,n}(\mathbb{R})$ both the image of A onto which we project changes and additionally the projector is influenced. We will have to take care of both effects. First we have to answer the question which perturbation δA causes $\mathrm{im}(A)$ to tilt by a maximum angle $\delta\alpha$. Let $v \in \mathbb{C}^n$ with $\|v\| = 1$. The maximal distance by which we can move the image of v under A, denoted by $p = Av \in \mathrm{im}(A)$, is attained by creating a perturbation δp which is orthogonal on $\mathrm{im}(A)$. Thus we let $\delta A = \delta p v^*$, such that $\|\delta A\| = \|\delta p\|$ is satisfied. Now we observe that the closer $\|p\|$ is to zero, the larger the angle of tilt, we can achieve by a given $\|\delta p\|$, will be. This means that we have to choose v to be the left singular vector u_n corresponding to the smallest singular value σ_n. As we have assumed that A is diagonal u_n corresponds to the n-th unit vector in $\mathbb{R}^n$ and $p = \sigma_n u_n$. As $\delta\alpha \in \left[0; \frac{\pi}{2}\right[$ we know that $\delta\alpha \leq \tan(\delta\alpha)$, as $\frac{d}{dx}(\tan(x) - x) = \tan(x)^2 \geq 0$, $\tan(0) - 0 = 0$ and $\tan(x) - x$ is continuous for $x \in \left[0; \frac{\pi}{2}\right[$. So we obtain $\delta\alpha \leq \tan(\delta a) = \frac{\|\delta p\|}{\sigma_n} = \frac{\|\delta A\|}{\sigma_n} = \frac{\|\delta A\|}{\|A\|}\kappa(A)$.

Let us recall that A is diagonal so the orthogonal projection $P = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$ of b on $\mathrm{im}(A)$ is given by

$$Pb = P\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ 0 \end{bmatrix} = y,$$

with $b_1 = y_{1:n} \in \mathbb{R}^n$. Consequently $x = A_1^{-1} b_1$.

Let us now split δA into two parts, namely $\delta A_1 = \delta A_{1:n,n} \in \mathrm{Mat}_n(\mathbb{R})$, which leaves $\mathrm{im}(A)$ unchanged, and $\delta A_2 = \delta A_{n+1:m,n} \in \mathrm{Mat}_{m-n,n}(\mathbb{R})$, which will also tilt $\mathrm{im}(A)$. Then we are in the situation that

$$\delta A = \begin{bmatrix} \delta A_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \delta A_2 \end{bmatrix}$$

in which we can investigate the effects independently of each other. The action of δA_1 is exactly the one described in Proposition 2.7.18, where we discussed the sensitivity of the solution of a linear system x with respect to perturbations in A. The condition number is therefore $\kappa(A)$. Now we focus on the effect of δA_2. Essentially this means that we are perturbing b_1 while leaving A_1 unchanged. So the condition number for this problem is given by

$$\frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta b_1\|}{\|b_1\|} \leq \frac{\left\|A_1^{-1}\right\|}{\|x\| / \|b_1\|} = \frac{\left\|A_1^{-1}\right\| \|A_1\| \|A_1 x\|}{\|A_1\| \|x\|} = \frac{\kappa(A)}{\eta}.$$

What remains to be done is to derive a relation between δb_1 and δA_2. Only the components of δy which are parallel to $\mathrm{im}(A)$, i.e. the part of δy which is contained in $\mathrm{im}(A)$, will translate to changes in δb_1. See Figure 2.3 for a graphical illustration in $\mathbb{R}^2$. By geometric arguments we know that

$$\sin(\theta) = \frac{\|\delta b_1\|}{\|\delta y\|}.$$

Now we need to relate δy to $\delta\alpha$. Using the law of sines (compare [5, page 134]) we obtain

$$\frac{\sin(\delta\alpha)}{\|\delta y\|} = \frac{\sin\left(\frac{\pi}{2} + \theta\right)}{\|y\|} = \frac{\cos(\theta)}{\|y\|} = \frac{1}{\|b\|}.$$

After observing that $\sin(x) \leq x$ for $x \geq 0$ we can write

$$\|\delta y\| = \|b\| \sin(\delta\alpha) \leq \|b\| \, \delta\alpha.$$

Now we if we put everything together we obtain

$$\|\delta b_1\| = \sin(\theta) \, \|\delta y\| \leq \sin(\theta) \, \|b\| \, \delta\alpha.$$

Using $\|b_1\| = \cos(\theta) \, \|b\|$ (compare Figure 2.3), we can write

$$\frac{\|\delta b_1\|}{\|b_1\|} \leq \tan(\theta) \, \delta\alpha \leq \tan(\theta) \, \frac{\|\delta A_2\|}{\|A\|} \kappa(A).$$

Finally, we obtain

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} \Big/ \tan(\theta) \, \frac{\|\delta A_2\|}{\|A\|} \kappa(A) &\leq \frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta b_1\|}{\|b_1\|} \leq \frac{\kappa(A)}{\eta} \\ \frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A_2\|}{\|A\|} &\leq \frac{\tan(\theta) \, \kappa(A)^2}{\eta}. \end{aligned}$$

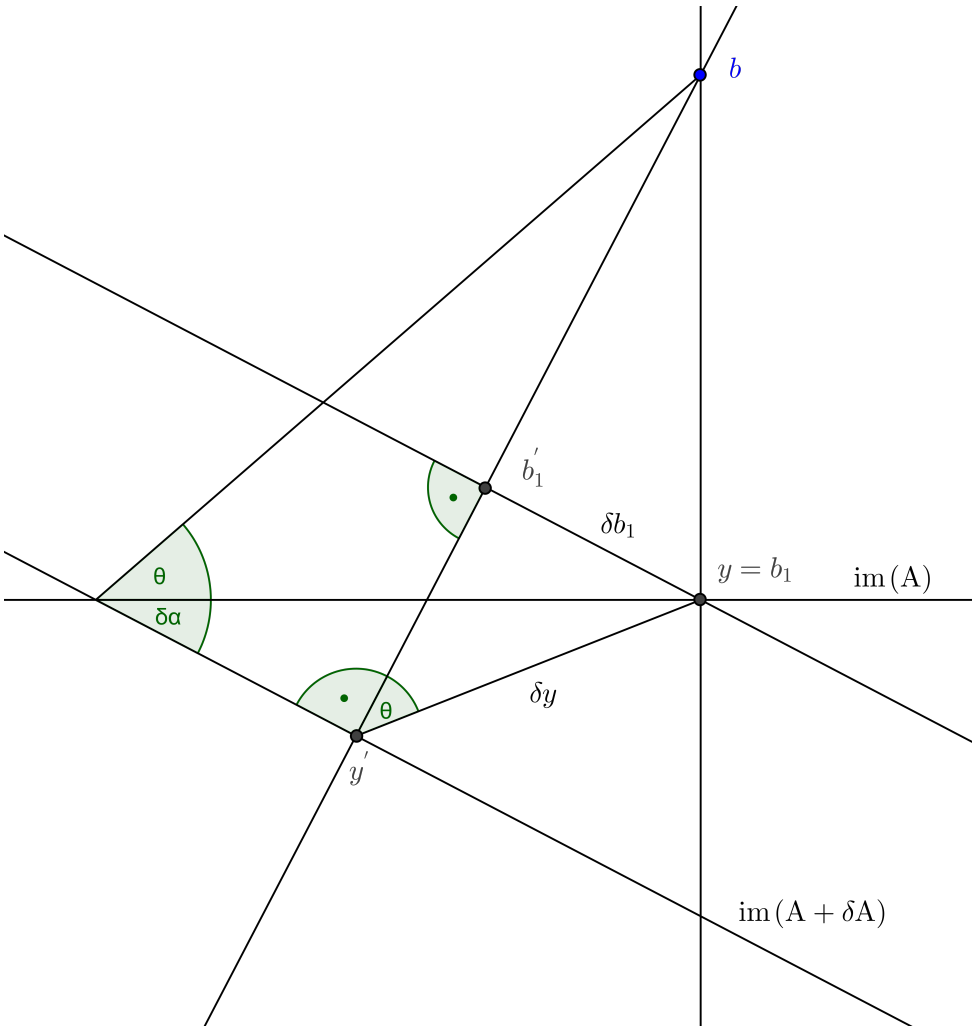

Figure 2.3: Relation between δb_1 and δy

To conclude our proof, we use the fact that $\|\delta A\| \leq \|\delta A_1\| + \|\delta A_2\|$ and compute

$$\begin{aligned}
\frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A\|}{\|A\|} &= \frac{\|\delta x\| \, \|A\|}{\|x\| \, \|\delta A\|} = \frac{\|\delta x\| \, \|A\|}{\|x\|} \cdot \frac{\|\delta A\|}{\|\delta A\|^2} \leq \frac{\|\delta x\| \, \|A\|}{\|x\|} \cdot \frac{\|\delta A_1\| + \|\delta A_2\|}{\|\delta A\|^2} \\
&\leq \frac{\|\delta x\| \, \|A\|}{\|x\|} \cdot \frac{\|\delta A_1\| + \|\delta A_2\|}{\|\delta A_1\| \, \|\delta A_2\|} \leq \frac{\|\delta x\| \, \|A\|}{\|x\|} \cdot \left(\frac{1}{\|\delta A_2\|} + \frac{1}{\|\delta A_1\|} \right) \\
&= \frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A_1\|}{\|A\|} + \frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A_2\|}{\|A\|} \\
&\leqq \kappa(A) + \frac{\tan(\theta) \, \kappa(A)^2}{\eta}.
\end{aligned}$$

□

2.13 Solutions of the Homogeneous Least Squares Problem

Before we start, let us briefly recall the definition of the homogeneous least squares problem (see also Definition 2.10.2). Given $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $c \in \mathbb{R}^+$, solve $\min_x \|Ax\|_2$ subject to $\|x\|_2 = c$. So far we have treated the solution of the inhomogeneous least squares problem in Section 2.11. Unfortunately the methods developed up till now cannot be directly applied to the homogeneous least squares problem, as the additional constraint $\|x\|_2 = c$ cannot be easily incorporated into the previous approach.

We will now further analyse how such a constrained solution can be constructed. In order to address our problem, we are in need of a few more definitions. As we are trying to solve a constrained optimisation problem the method of Lagrange Multipliers, which was brought up by Joseph Louis Lagrange, will become handy.

Definition 2.13.1. Let $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}$ be real valued continuously differentiable functions, and let $c \in \mathbb{R}^+$. The task of minimizing

$$f(x_1, ..., x_n)$$

under the constraint

$$g(x_1, ..., x_n) = c$$

is called a **constrained minimisation (or optimisation) problem**. We call the function $\Phi : \mathbb{R}^{n+1} \to \mathbb{R}$

$$\Phi(x_1, ..., x_n, \psi) = f(x_1, ..., x_n) - \psi(g(x_1, ..., x_n) - c)$$

the **Lagrange function** of f, g and c. The newly introduced real variable ψ is called **Lagrange multiplier**.

Theorem 2.13.2. *If $s = (s_1, ..., s_n) \in \mathbb{R}^n$ is a solution of a constrained minimisation problem, as defined above, there exists $\lambda \in \mathbb{R}$ such that $(s_1, ..., s_n, \lambda)$ is a stationary point (see [67, page 16]) of the Lagrange function Φ.*

Proof. This theorem is a special case of [67, Theorem 12.1] with only one equality constraint and no inequality constraints. A proof of the more general theorem is also contained in [67]. □

Remark 2.13.3. Stationarity of the Lagrange function for (s, λ) is a necessary but not sufficient first order condition which has to be satisfied. The necessary first order conditions for a more general optimisation problem with several equality and inequality constraints are known as the Karush-Kuhn-Tucker conditions. After we have found the stationary points we can decide with the help of standard techniques which of them are in fact local minima. For a detailed discussion of necessary higher order conditions consider [67, Section 12.4].

Remark 2.13.4. Consider the real valued complex functions $f : \mathbb{C}^n \to \mathbb{R}$ and $g : \mathbb{C}^n \to \mathbb{R}$. We can view them as two real valued functions $\tilde{f} : \mathbb{R}^{2n} \to \mathbb{R}$ and $\tilde{g} : \mathbb{R}^{2n} \to \mathbb{R}$ by splitting each complex input variable into two real variables where the first one contains its real and the second one its complex part. In case $\tilde{f}$ and $\tilde{g}$ are continuously differentiable we can use Theorem 2.13.2 to minimise f under the constraint $g(x_1, ..., x_n) = c$.

Proposition 2.13.5. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. The matrix A^*A is diagonalisable and has only real non-negative eigenvalues.*

Proof. Let $A = U\Sigma V^*$ be a SVD of A. We obtain

$$\begin{aligned} A^*A &= (U\Sigma V^*)^* U\Sigma V^* = V\Sigma^{\mathrm{tr}}U^*U\Sigma V^* \\ &= V\Sigma^{\mathrm{tr}}\Sigma V^*. \end{aligned}$$

The matrix $\Lambda = \Sigma^{\mathrm{tr}}\Sigma \in \mathrm{Mat}_n(\mathbb{R})$ is diagonal and contains the squares of the singular values of A. Therefore, $V\Lambda V^*$ is an eigendecomposition of A^*A, and we have shown that A^*A is diagonalisable and has only non-negative eigenvalues. □

The following theorem characterises the solutions of the homogeneous least squares problem. It is well-known, and can for example be proven via the properties of the SVD or via Lagrange multipliers. We choose the latter approach and also we give a proof for complex matrices, which is commonly not contained in the literature and could therefore be cited.

Theorem 2.13.6. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ with $m \geq n$ and let $c \in \mathbb{R}^+$. A basis for the solutions $x \in \mathbb{C}^n$ of the homogeneous least squares problem subject to the constraint $\|x\|_2 = c$ is given by the norm c eigenvectors corresponding to the smallest eigenvalue $\lambda \in \mathbb{R}_0^+$ of A^*A. The norm of the residual $\|Ax\|_2$ is given by $\sqrt{\lambda}c$.*

Proof. We want to minimise $f' : \mathbb{C}^n \to \mathbb{R}$ given by $f'(x_1, ..., x_n) = f'(x) = \|Ax\|$ subject to the constraint on $g' : \mathbb{C}^n \to \mathbb{R}$ with $g'(x_1, ..., x_n) = g'(x) = \|x\|$ given by $g'(x) = \|x\| = c$. As minimizing $\|Ax\| = \sqrt{(Ax)^*(Ax)}$ is equivalent to minimizing $f(x) = (Ax)^*(Ax)$ and the constraint $\|x\| = c$ is equivalent to $g(x) = x^*x = c$ we will go for those simpler expressions. First of all we decompose x in its real and imaginary part by letting $x = a + bi = (a_1, ..., a_n) + (b_1, ..., b_n)\, i$ where $a_1, ..., a_n, b_1, ..., b_n$ are real variables. We thus obtain $\tilde{f} : \mathbb{R}^{2n} \to \mathbb{R}$ with

$$\begin{aligned} \tilde{f}(a,b) &= (A(a+bi))^*(A(a+bi)) = (a-bi)^{\mathrm{tr}} A^*A(a+bi) \\ &= a^{\mathrm{tr}}(A^*A)a + a^{\mathrm{tr}}(A^*A)bi - b^{\mathrm{tr}}i(A^*A)a + b^{\mathrm{tr}}(A^*A)b \\ &= a^{\mathrm{tr}}(A^*A)a + a^{\mathrm{tr}}(A^*A)bi - b^{\mathrm{tr}}i(A^*A)a + b^{\mathrm{tr}}(A^*A)b \\ &= a^{\mathrm{tr}}(A^*A)a + 2\Im\left(a^{\mathrm{tr}}(A^*A)b\right) + b^{\mathrm{tr}}(A^*A)b \end{aligned}$$

and $\tilde{g} : \mathbb{R}^{2n} \to \mathbb{R}$ with

$$\begin{aligned} \tilde{g}(a,b) &= (a+bi)^*(a+bi) = a^2 + abi - abi + b^2 \\ &= a^2 + b^2. \end{aligned}$$

Both $\tilde{f}$ and $\tilde{g}$ are continuously differentiable. In order to find the minimum we must first find the stationary points of $\Phi(x,\lambda) = \Phi(a,b,\lambda) = a^{\mathrm{tr}}(A^*A)a + a^{\mathrm{tr}}(A^*A)bi - b^{\mathrm{tr}}i(A^*A)a + b^{\mathrm{tr}}(A^*A)b - \lambda\left(a^2+b^2-1\right)$. We differentiate with respect to a and obtain

$$\begin{aligned} \frac{\partial}{\partial a}(\Phi(a,b,\lambda)) &= \frac{\partial}{\partial a}\left(a^{\mathrm{tr}}(A^*A)a + a^{\mathrm{tr}}(A^*A)bi - b^{\mathrm{tr}}i(A^*A)a - \lambda a^2\right) \\ &= \left(A^*A + (A^*A)^{\mathrm{tr}}\right)a + (A^*A)bi - (A^*A)^{\mathrm{tr}}bi - \lambda 2a \\ &= \left(A^*A + A^{\mathrm{tr}}\bar{A}\right)a + 2\Re\left((A^*A)\,i\right)b + 2\lambda a \\ &= 2\Re(A^*Aa + (A^*A)\,ib) - 2\lambda a = 2\Re(A^*A(a+bi)) - 2\lambda a. \end{aligned}$$

Next we differentiate Φ with respect to b. This yields

$$\begin{aligned}\frac{\partial}{\partial b}\left(\Phi\left(a,b,\lambda\right)\right) &= \frac{\partial}{\partial b}\left(b^{\mathrm{tr}}\left(A^*A\right)b + a^{\mathrm{tr}}\left(A^*A\right)bi - b^{\mathrm{tr}}i\left(A^*A\right)a + b^{\mathrm{tr}}\left(A^*A\right)b - \lambda b^2\right)\\ &= \left(A^*A + A^{\mathrm{tr}}\bar{A}\right)b + \left(A^*A\right)^{\mathrm{tr}}ai - \left(A^*A\right)ai - 2\lambda b\\ &= 2i\frac{\left(A^*A + A^{\mathrm{tr}}\bar{A}\right)}{2i}b + 2\Im\left(\left(A^*A\right)a\right) - \left(A^*A\right)ai - 2\lambda b\\ &= 2\Im\left(\left(A^*A\right)a + \left(A^*A\right)ib\right) - 2\lambda b = 2\Im\left(A^*A\left(a+bi\right)\right) - 2\lambda b.\end{aligned}$$

By the definition of the Lagrange multiplier, the derivative with respect to λ is again our constraint $\tilde{g}\left(a,b\right) = c$. Now we need to find solutions of the equations system

$$\begin{aligned}2\Re\left(A^*A\left(a+bi\right)\right) - 2\lambda a &= 0\\ 2\Im\left(A^*A\left(a+bi\right)\right) - 2\lambda b &= 0\\ a^2 + b^2 &= c.\end{aligned}$$

If we note that A^*A is Hermitian and has only real eigenvalues it becomes apparent from these expressions that the (complex) norm c eigenvectors of A^*A are the stationary points of the Lagrange function and λ is the corresponding eigenvalue. As $\|Ax\| \geq 0$ the minimum has to be assumed on one of the stationary points. Now if we let x be an eigenvector and λ the corresponding eigenvalue we obtain

$$\|Ax\| = \sqrt{x^*A^*Ax} = \sqrt{x^*\lambda x} = \sqrt{\lambda x^*x} = \sqrt{\lambda c}.$$

This means that $\|Ax\|$ is minimal for all eigenvectors corresponding to the smallest eigenvalue. Note that A^*A is diagonalisable which means that it possesses a complete basis of eigenvectors. □

Remark 2.13.7. As we can deduce from the proof of Theorem 2.13.6, the choice of c does not influence the structure of the solution. The only property that is changed by varying c is the norm of the eigenvectors.

For reasons of simplicity and because it is customary to work with norm one eigenvectors from now on we will let $c = 1$.

Theorem 2.13.6 suggests a direct method to obtain the solutions of the homogeneous least squares problem: First, we form A^*A and then we calculate a basis of the eigenspace associated with the smallest eigenvalue of this matrix.

Algorithm 16: Homogeneous least squares

Input: A matrix $A \in \mathrm{Mat}_{m,n}\left(\mathbb{C}\right)$

Output: A basis $v_1, ..., v_k$ for the solutions of the homogeneous least squares problem, the norm of the residual

1 $M := A^*A$;
2 $(v_1, ..., v_k) :=$ (norm one) eigenvectors of M with respect to the smallest eigenvalue λ of M;
3 $r := \sqrt{\lambda}$;
4 **return** $(v_1, ..., v_k, r)$;

As a next step, we will analyse the accuracy of Algorithm 16. For this purpose let us denote by $\lambda_k(\cdot)$ the k-th eigenvalue of a Hermitian matrix $A \in \mathrm{Mat}_m(\mathbb{C})$, assuming that the eigenvalues are ordered descendingly, and by $\sigma_k(\cdot)$ the k-th singular value of a matrix, assuming that the singular values are ordered descendingly (compare Section 2.1).

Theorem 2.13.8 (Accuracy of Algorithm 16). *Let $m \geq n$, let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be a matrix, let $E \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be such that $\frac{\|E\|_2}{\|A\|_2} \in O(\varepsilon_{machine})$, and let $\tilde{A} = A + E$. Let us further assume that the homogeneous least squares problem, i.e. find all $x \in \mathbb{C}^n$ subject to $\|x\|_2 = 1$ such that $\|Ax\|_2$ is minimised, is addressed via Algorithm 16 and a backward stable algorithm such as the QR algorithm (2.9.1) is used to solve the underlying eigenvalue problem in step 2 on a computer that satisfies Assumption 2.7.5. Let us denote by x the exact solution, by $r = \|Ax\|_2$ the exact residual, by $\tilde{x}$ the computed solution and by $\tilde{r} = \|\tilde{A}\tilde{x}\|_2$ the computed residual of the homogeneous least squares problem. Then for r and $\tilde{r}$ the following error estimate holds:*

$$|r - \tilde{r}| \in O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{r}\right) = O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{\tilde{r}}\right).$$

Let θ be the angle between x and $\tilde{x}$, i.e. $\cos(\theta)\|x\|\|\tilde{x}\| = \cos(\theta) = |\langle x, \tilde{x}\rangle|$. If $\mathrm{gap}_n\left(\tilde{A}^*\tilde{A}\right) \neq 0$ (see Definition 2.9.20), then the following error estimate holds:

$$\frac{1}{2}\sin(\theta) \in O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{\mathrm{gap}_n\left(\tilde{A}^*\tilde{A}\right)}\right).$$

Proof. If x and r are computed via Algorithm 16, then x is a norm one eigenvector of $A^*A \in \mathrm{Mat}_n(\mathbb{C})$ associated with the smallest eigenvalue r^2 of A^*A. Please recall from Proposition 2.13.5 that A^*A has only real non-negative eigenvalues. In the following let $B = A^*A$ and let

$$\begin{aligned}\tilde{B} &= \tilde{A}^* \hat{\times} \tilde{A} = \tilde{A}^*\tilde{A} + G = (A+E)^*(A+E) + G \\ &= A^*A + AE + E^*A + E^*E + G \\ &= B + F\end{aligned}$$

be the equivalent of matrix B that was computed with floating point arithmetic (compare Definition 2.7.7) with associated (error) matrices $E, G, F \in \mathrm{Mat}_n(\mathbb{C})$. As we assume that the machine with which we are working has backward stable implementations of the basic arithmetic operations and because $\frac{\|E\|_2}{\|A\|_2} \in O(\varepsilon_{machine})$ we know that $\frac{\|G\|_2}{\|A\|_2} \in O(\varepsilon_{machine})$ and also $\frac{\|F\|_2}{\|B\|_2} \in O(\varepsilon_{machine})$ must hold.

First of all we recall that B is a Hermitian matrix, for this purpose see Proposition 2.3.24. Additionally, we may assume that the matrix $\tilde{B}$ is also Hermitian as any sensible floating point implementation would only compute and store the upper or lower triangular part of $\tilde{B}$. The other elements are obtained as the complex conjugates of the elements that are actually stored. Note, that if we would compute each entry of $\tilde{B}$ explicitly in floating point arithmetic, the

matrix $\tilde{B}$ would in general not be Hermitian! By Corollary 2.9.18, we know that the smallest eigenvalue λ_i of B satisfies the inequality

$$\left|\lambda_i(B) - \lambda_i(\tilde{B})\right| \leq \|F\|_2 .$$

This means that $\lambda_i(\tilde{B}) = \lambda_i(B) + k\|F\|_2$ for some $k \in [-1,1]$. This implies that

$$\left|\lambda_i(B) - \lambda_i(\tilde{B})\right| \in O\left(\varepsilon_{machine}\|A^*A\|_2\right) = O\left(\varepsilon_{machine}\|A\|_2^2\right) = O\left(\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2\right).$$

Now we need to distinguish a few different cases. In case $r = \tilde{r} = 0$ the claim is trivially true. Let us next consider the case that $r = 0$ and $\tilde{r} \neq 0$. Then

$$|r - \tilde{r}| = \left|\frac{\lambda_i(\tilde{B})}{\tilde{r}}\right| \in O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{\tilde{r}}\right).$$

As a next step we consider the case that $r \neq 0$ and $\tilde{r} = 0$. Here we observe that

$$|r - \tilde{r}| = \left|\frac{\lambda_i(B)}{r}\right| \in O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{r}\right).$$

So we can finally consider the case where $r \neq 0$ and $\tilde{r} \neq 0$. We verify that

$$\begin{aligned}
\tilde{r} &= \sqrt{\lambda_i(\tilde{B})} = \frac{r\sqrt{\lambda_i(\tilde{B})}}{r} \\
&= \frac{\sqrt{\lambda_i(B)}\sqrt{\lambda_i(\tilde{B})}}{r} = \frac{\sqrt{\lambda_i(\tilde{B}) - k\|F\|_2}\sqrt{\lambda_i(\tilde{B})}}{r} \\
&= \frac{\lambda_i(\tilde{B})}{r}\sqrt{1 - \frac{k\|F\|_2}{\lambda_i(\tilde{B})}} \\
&= \frac{\lambda_i(\tilde{B})}{r}\sqrt{1 - \frac{k\|F\|_2}{\lambda_i(B) + k\|F\|_2}}.
\end{aligned}$$

So we obtain

$$|r - \tilde{r}| = \left|\frac{\lambda_i(B)}{r} - \frac{\lambda_i(\tilde{B})}{r}\sqrt{1 - \frac{k\|F\|_2}{\lambda_i(B) + k\|F\|_2}}\right| \in O\left(\frac{\lambda_i(B) - \lambda_i(\tilde{B})}{r}\right).$$

Finally, we obtain

$$|r - \tilde{r}| \in O\left(\frac{\lambda_i(B) - \lambda_i(\tilde{B})}{r}\right) = O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}\right\|_2^2}{r}\right).$$

The claim about θ follows directly from Remark 2.9.22 if we use A^*A as the input matrix. □

Remark 2.13.9. Consider the same hypotheses as in Theorem 2.13.8. By Corollary 2.9.19, we know that for an arbitrary singular value σ_k the following bound can be established:

$$\left|\sigma_k(\tilde{A}) - \sigma_k(A)\right| \leq \|\delta A\|_2 .$$

This means that a backward stable algorithm for the problem would in fact compute $\tilde{r}$ such that

$$|r - \tilde{r}| \in O\left(\varepsilon_{machine} \|A\|_2\right) .$$

The algorithm which we have presented here is worse by a factor of $\frac{\|A\|_2}{r} = \frac{\sigma_1(A)}{\sigma_n(A)} = \kappa(A)$. This may become a problem if A is very ill-conditioned. However, if necessary the numerical stability can be improved and brought to the level that $|r - \tilde{r}| \in O\left(\varepsilon_{machine} \|A\|_2\right)$ with the help of the techniques which will be described in Subsection 4.3.2.

3 Border Bases

Contents

Border bases, a generalisation of Gröbner bases for zero-dimensional ideals, have attracted the interest of many researchers recently. One of the reasons is the enhanced numerical stability of border bases compared to Gröbner bases which makes them more suitable for applications. This was, for instance, demonstrated in "Approximate computation of zero-dimensional polynomial ideals" ([28]). Kreuzer, Poulisse, et al. extended exact border bases to approximate border bases and gave an efficient algorithm, the so-called AVI algorithm (21), for their computation. Border basis were also used by Stetter in his book "Numerical Polynomial Algebra" ([49]) as a comparatively stable tool for polynomial system solving. Further notable applications of border basis were given by Mourrain in [24] and [26].
As exact and approximate border bases are fundamental to the understanding of the ABM family of algorithms we start out with a brief introduction to the topic of border bases and then move on to approximate border bases.

3.1 Exact Border Bases

In this chapter we deal with exact border bases and motivate those special properties which make them suitable for practical applications. Let K be a field, $n \geq 1$ and $P = K[x_1, ..., x_n]$ the polynomial ring over K in n indeterminates. Recall that by $\mathbb{T}^n$ we denote the set of all terms in P.

Just like [22] we introduce the following definitions:

Definition 3.1.1. [Order Ideal]

A finite set of terms $\emptyset \subset \mathcal{O} \subseteq \mathbb{T}^n$ is called an **order ideal** if $t \in \mathcal{O}$ implies that $t' \in \mathcal{O}$ for every t' dividing t.

Definition 3.1.2. [Border]

The **border** of an order ideal will be denoted by $\partial\mathcal{O}$ and is defined as

$$\partial\mathcal{O} = \mathbb{T}_1^n \cdot \mathcal{O} \setminus \mathcal{O} = (x_1\mathcal{O} \cup ... \cup x_n\mathcal{O}) \setminus \mathcal{O}\,.$$

Example 3.1.3. Let $P = K[x,y]$ and let $\mathcal{O} = \{1, x, y, xy, x^2, x^3\}$ be an order ideal. The border of $\mathcal{O}$ is given by $\partial\mathcal{O} = \{y^2, x^2y, xy^2, x^4, x^3y\}$. In Figure 3.1 we can see a visualisation of $\mathcal{O}$ and of its border.

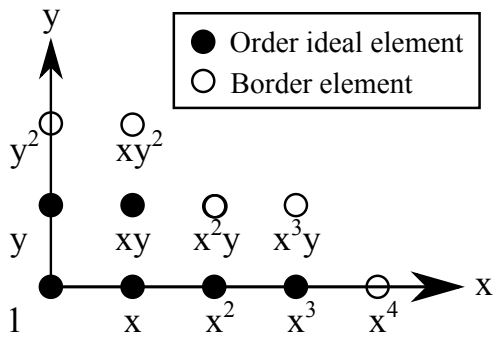

Figure 3.1: Depiction of an order ideal and of its border.

Definition 3.1.4. [Border closure]

The **first border closure** of $\mathcal{O}$ is denoted by $\overline{\partial\mathcal{O}}$ and is defined as

$$\overline{\partial\mathcal{O}} = \partial\mathcal{O} \cup \mathcal{O}.$$

Definition 3.1.5. We let $\partial^0\mathcal{O} = \overline{\partial^0\mathcal{O}} = \mathcal{O}$. For every $k \geq 1$ we inductively define the **k-th border** of $\mathcal{O}$ by $\partial^k\mathcal{O} = \partial\left(\overline{\partial^{k-1}\mathcal{O}}\right)$ and the **k-th border closure** of $\mathcal{O}$ by $\overline{\partial^k\mathcal{O}} = \overline{\partial^{k-1}\mathcal{O}} \cup \partial^k\mathcal{O}$.

Definition 3.1.6. [Index]

We define the **index** of $t \in \mathbb{T}^n$ with respect to $\mathcal{O}$ as

$$\mathrm{ind}_{\mathcal{O}}(t) = \min\left\{k \geq 0 \,\middle|\, t \in \overline{\partial^k\mathcal{O}}\right\}.$$

Given a polynomial $f = c_1t_1 + ... + c_st_s \in P \setminus \{0\}$, with $c_1, ..., c_s \in K \setminus \{0\}$ and $t_1, ..., t_s \in \mathbb{T}^n$, we assume that the t_i are ordered such that $\mathrm{ind}_{\mathcal{O}}(t_1) \geq \mathrm{ind}_{\mathcal{O}}(t_2) \geq ... \geq \mathrm{ind}_{\mathcal{O}}(t_s)$. Then we define the **index** of f as $\mathrm{ind}_{\mathcal{O}}(f) = \mathrm{ind}_{\mathcal{O}}(t_1)$.

Definition 3.1.7. [$\mathcal{O}$-border prebasis]

Consider an order ideal $\mathcal{O}$ with corresponding border $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$. A set of polynomials $G = \{g_1, ..., g_\nu\} \subset P$ is called an **$\mathcal{O}$-border prebasis** if every polynomial $g_i \in G$ is of the form $g_i = b_i + h_i$ such that $h_i \in P$ satisfies $\mathrm{supp}(h_i) \subseteq \mathcal{O}$.

Algorithm 17: Border Division

Input: An order ideal $\mathcal{O} = \{t_1, ..., t_\mu\}$, an $\mathcal{O}$-border prebasis $G = \{g_1, ..., g_\nu\}$, and a polynomial $f \in P$

Output: A tuple $(f_1, ..., f_\nu, c_1, ..., c_\mu) \in P^\nu \times K^\mu$

$f_1 := ... := f_\nu = 0$, $c_1 := ... := c_\mu := 0$, $h := f$;
$[b_1, ..., b_\nu] = \partial\mathcal{O}$;
while $h \neq 0$ **do**
  if $\text{ind}_\mathcal{O}(h) = 0$ **then**
    for *i := 1* **to** μ **do**
      $c_i :=$ coefficient of t_i in h;
    end
    return $(f_1, ..., f_\nu, c_1, ..., c_\mu)$;
  else
    /* We assume that $h = a_1h_1 + ... + a_sh_s$ with $a_1, ..., a_s \in K \setminus \{0\}$ and $h_1, ..., h_s \in \mathbb{T}^n$ such that $\text{ind}_\mathcal{O}(h_1) = \text{ind}_\mathcal{O}(h)$ */
    for *i := 1* **to** ν **do**
      if $\text{isDivisible}(h_1, b_i)$ **and** $\text{ind}_\mathcal{O}(h_1/b_i) = \text{ind}_\mathcal{O}(h) - 1$ **then**
        $t' := h_1/b_i$;
        $h := h - a_1t'g_i$;
        $f_i := f_i + a_1t'$;
        break;
      end
    end
  end
end
return $(f_1, ..., f_\nu, c_1, ..., c_\mu)$;

Theorem 3.1.8. *Given an order ideal $\mathcal{O}$, an $\mathcal{O}$-border prebasis $G = \{g_1, ..., g_\nu\}$, and a polynomial $f \in P$, this is an algorithm that returns a tuple $(f_1, ..., f_\nu, c_1, ..., c_\mu) \in P^\nu \times K^\mu$ such that*

$$f = f_1g_1 + ... + f_\nu g_\nu + c_1t_1 + ... + c_\mu t_\mu$$

with $\deg(f_i) \leq \text{ind}_\mathcal{O}(f) - 1$ for all $1 \leq i \leq \nu$ where $f_ig_i \neq 0$. For a fixed order of the elements in G the result is unique.

Proof. A proof can be found in [22, Proposition 6]. □

Definition 3.1.9. [Normal Remainder]

Let $\mathcal{O}$ be an order ideal, let G be an $\mathcal{O}$-border prebasis (in which the elements are ordered in a fixed way) and let $f \in P$. Now let

$$f = f_1g_1 + ... + f_\nu g_\nu + c_1t_1 + ... + c_\mu t_\mu$$

be the decomposition computed by the Border Division Algorithm (17). Then we call

$$\mathrm{NR}_{\mathcal{O},G}(f) = c_1 t_1 + ... + c_\mu t_\mu$$

the **normal $\mathcal{O}$-remainder** of f.

In the following let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, and let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be its border. Additionally, let $G = \{g_1, ..., g_\nu\}$ be an $\mathcal{O}$-border prebasis, and let I be a zero-dimensional ideal of P which contains G.

Definition 3.1.10. [$\mathcal{O}$-border basis]

An $\mathcal{O}$-border prebasis is called an **$\mathcal{O}$-border basis** of I if the residue classes of the elements of $\mathcal{O}$ form a vector space basis of P/I.

We now investigate characterisations of border bases which we will use later to prove, for example, the correctness of the Buchberger-Möller algorithm for border bases.

Definition 3.1.11. [Rewrite relations]

Let G be an $\mathcal{O}$-border basis as defined above, and let $f \in P$ be a polynomial. Furthermore, let $t \in \mathrm{supp}(f)$ be such that t is a multiple of a border term, meaning $t = t'b_i$ with $t' \in \mathbb{T}^n$ and $b_i \in \partial\mathcal{O}$. If $c \in K$ is the coefficient of t in f, then $h = f - ct'g_i$ does not contain t any more. We say that f **reduces to h in one step** using g_i and write $f \xrightarrow{g_i} h$. The reflexive, transitive closure of the relations $\xrightarrow{g_i}$, with $1 \leq i \leq \nu$, is called the **rewrite relation** associated to G and denoted by $\xrightarrow{G}$.

Definition 3.1.12. [Neighbours]

Let $b_i, b_j \in \partial\mathcal{O}$ and $b_i \neq b_j$. The border terms b_i and b_j are called **next-door neighbours** if $b_i = x_k b_j$ or if $b_j = x_k b_i$ for some $k \in \{1, ..., n\}$. The border terms b_i and b_j are called **across-the-street neighbours** if $x_k b_i = x_\ell b_j$ for some $k, \ell \in \{1, ..., n\}$. If b_i and b_j are either next-door neighbours or across-the-street neighbours, we can simply refer to them as **neighbours**.

Remark 3.1.13. [Least common multiple]

The **least common multiple** of two terms $t_1 = \prod_{i=1}^n x_i^{a_i}$ and $t_2 = \prod_{i=1}^n x_i^{b_i}$ is given by $\mathrm{lcm}(t_1, t_2) = \prod_{i=1}^n x_i^{\max(a_i, b_i)}$.

Definition 3.1.14. [S-polynomial]

Let $g_i = b_i + h_i$ and $g_j = b_j + h_j$ be two polynomials of an $\mathcal{O}$-border prebasis. The polynomial

$$S(g_i, g_j) = \frac{\mathrm{lcm}(b_i, b_j)}{b_i} g_i - \frac{\mathrm{lcm}(b_i, b_j)}{b_j} g_j$$

is called the **S-polynomial** of g_i and g_j.

Theorem 3.1.15 (Buchberger's criterion for border bases)**.** *An $\mathcal{O}$-border prebasis G is a border basis of $I = \langle G \rangle$ if and only if for all neighbours b_i and b_j in the border of $\mathcal{O}$ the S-polynomial $S(g_i, g_j)$ can be reduced to zero via $\xrightarrow{G}$.*

Proof. See [22, Proposition 18]. □

3.2 Numerical Stability of Border Bases

In this subsection we highlight which properties make border bases suitable for practical applications. For this purpose we have a look at an example first, which can also be found in [46]. A similar example that illustrates the stability of H-bases is presented by Möller and Sauer in [40]. To fully understand the example given here, some knowledge about Gröbner bases theory is necessary. However, as this is not the subject of this thesis, the interested reader is referred to [45] and [46] for full details about Gröbner bases.

Example 3.2.1. Let $P = \mathbb{C}[x,y]$, $f_1 = \frac{1}{4}x^2 + y^2 - 1$, and $f_2 = x^2 + \frac{1}{4}y^2 - 1$. As depicted in Figure 3.2, the zero sets of the polynomials intersect in the four points $\mathbb{X} = \left\{\left(\pm\frac{2}{\sqrt{5}}, \pm\frac{2}{\sqrt{5}}\right)\right\}$. Now let σ be the `DegRevLex` term ordering. Then the set $\left\{x^2 - \frac{4}{5}, y^2 - \frac{4}{5}\right\}$ is the reduced σ-Gröbner basis of the ideal $I = (f_1, f_2)$. Consequently, $\mathrm{LT}_\sigma(I) = \left(x^2, y^2\right)$ and $\mathbb{T}^2 \backslash \mathrm{LT}_\sigma(I) = \{1, x, y, xy\}$.

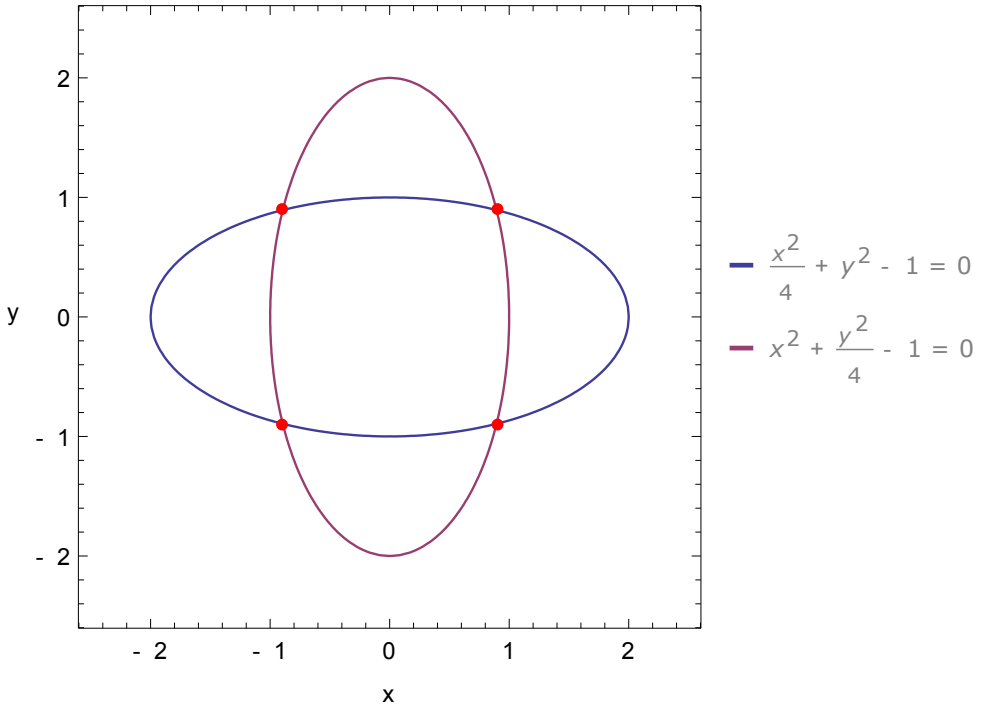

Figure 3.2: Intersection of the zero set of the polynomials f_1 and f_2.

Let us perturb f_1 and f_2 such that $\tilde{f}_1 = \frac{1}{4}x^2 + y^2 + \varepsilon xy - 1$ and $\tilde{f}_2 = x^2 + \frac{1}{4}y^2 + \varepsilon xy - 1$ where $\varepsilon \in \mathbb{R}$ is a small number. If we exclude $\varepsilon = \pm\frac{5}{4}$, then the intersection of the zero sets of the polynomials again consists of four points which are close to those in $\mathbb{X}$, namely $\tilde{\mathbb{X}} = \left\{\left(\pm\frac{2}{\sqrt{5-4\varepsilon}}, \pm\frac{2}{\sqrt{5-4\varepsilon}}\right)\right\}$. The reduced σ-Gröbner basis of $\tilde{I} = (\tilde{f}_1, \tilde{f}_2)$ is the set

$$\left\{x^2 - y^2, xy + \frac{5}{4\varepsilon}y^2 - \frac{1}{\varepsilon}, y^3 - \frac{16\varepsilon}{16\varepsilon^2 - 25}x + \frac{20}{16\varepsilon^2 - 25}y\right\}$$

with $\mathrm{LT}_\sigma(\tilde{I}) = \left(x^2, xy, y^3\right)$ and $\mathbb{T}^2 \setminus \mathrm{LT}_\sigma(\tilde{I}) = \left\{1, x, y, y^2\right\}$. What we can observe is that a possibly small perturbation of the coefficients of f_1 and f_2 can lead to a significant change in

the Gröbner basis of I. Note that we cannot let $\varepsilon = 0$ in $\tilde{I}$. This behaviour is known as a **representation singularity**.

Now let us examine what happens when I is perturbed when looking at border bases.
Both I and $\tilde{I}$ have a border basis with respect to $\mathcal{O} = \{1, x, y, xy\}$. As the border of $\mathcal{O}$ is $\{x^2, x^2y, xy^2, y^2\}$ the $\mathcal{O}$-border basis of I is

$$\left\{x^2 - \frac{4}{5}, x^2y - \frac{4}{5}y, xy^2 - \frac{4}{5}x, y^2 - \frac{4}{5}\right\}.$$

Furthermore, the $\mathcal{O}$-border basis of $\tilde{I}$ is given by

$$\left\{x^2 + \frac{4}{5}\varepsilon xy - \frac{4}{5}, x^2y - \frac{16\varepsilon}{16\varepsilon^2 - 25}x + \frac{20}{16\varepsilon^2 - 25}y,\right.$$
$$\left.xy^2 + \frac{20}{16\varepsilon^2 - 25}x - \frac{16\varepsilon}{16\varepsilon^2 - 25}y, y^2 + \frac{4}{5}\varepsilon xy - \frac{4}{5}\right\}.$$

In this case we can observe that the representation in terms of the border basis of I and $\tilde{I}$ is stable, meaning that if we let $\varepsilon = 0$ in the border basis of $\tilde{I}$ we obtain the border basis of I.

This example illustrates that if structural information shall be derived from the generating system of an ideal, Gröbner bases have some serious shortcomings which are not so prominent in border bases. This effect is mainly due the additional degrees of freedom provided by a border basis, as a generic border prebasis is parametrised by $n \times b$ parameters, with $n = |\mathcal{O}|$ and $b = |G| = |\partial\mathcal{O}|$. It is also possible to show that border bases are in general stable with respect to perturbations in their zero set $\mathbb{X} = \mathcal{Z}(I)$. In fact, this is the case we are mostly concerned with as we want to compute (approximate) border basis for measurements $\tilde{\mathbb{X}}$ which contain a certain amount of noise. First we note that if the set $\mathbb{X}$ contains s elements, a suitable order ideal $\mathcal{O}$ will contain s elements as well. Furthermore, the evaluation matrix (see Definition 3.3.4) $M \in \mathrm{Mat}_s(\mathbb{C})$ of $\mathcal{O}$ with respect to $\mathbb{X}$ will be non-degenerate. If we perturb the points $\mathbb{X}$ in a generic way and compute the corresponding evaluation matrix $\tilde{M}$ with respect to $\mathcal{O}$ it will almost surely still have full rank. Thus it is possible to construct a new border basis for the perturbed points by only changing the coefficients of the polynomials, which we had computed for $\mathbb{X}$. The monomial structure dictated by $\mathcal{O}$ remains unchanged. In other words, for every point $p \in \mathbb{X}$ exists a non-empty environment $U_\varepsilon = \{x \in \mathbb{C}^n \,|\, \|x - p\| < \varepsilon\}$ for $\varepsilon > 0$, in which the point p may be perturbed, while the associated border basis can still maintain its shape with respect to $\mathcal{O}$. More details about this result can be found in [34].

3.3 Affine Point Sets

First of all, we make ourselves familiar with affine point sets and ideals of points. These are essential ingredients when performing exact multivariate polynomial interpolation. As already mentioned in the introduction, in our setting we are not interested in exact polynomial interpolation as we are dealing with noisy input data. However, the exact methods will be a good starting point for what we call "approximate polynomial interpolation".

Just like [46] we introduce the following definitions.

Definition 3.3.1. Let K be a field and $P = K[x_1, ..., x_n]$ the polynomial ring over K in n indeterminates.

1. An element $p = (c_1, ..., c_n) \in K^n$ is called a **K-rational point**. We call $c_1, ..., c_n$ the **coordinates** of p.
2. A finite set of distinct K-rational points $\mathbb{X} = \{p_1, ..., p_s\}$ is called an **affine point set**.
3. Let $\mathbb{X}$ be an affine point set. Then the set of all polynomials $f \in P$ such that $f(p) = 0$ for all points $p \in \mathbb{X}$ forms an ideal of the polynomial ring P. It is called the **vanishing ideal** of $\mathbb{X}$ in P and we denote it by $\mathcal{I}(\mathbb{X})$. Sometimes, we may also refer to $\mathcal{I}(\mathbb{X})$ as an **ideal of points**.
4. The K-algebra $P/\mathcal{I}(\mathbb{X})$ is called the **(affine) coordinate ring** of $\mathbb{X}$.

Example 3.3.2. [The vanishing ideal of a single point]

If $\mathbb{X}$ contains only one point $p = (c_1, ..., c_n) \in K^n$, then the vanishing ideal of $\mathbb{X}$ is trivially given by

$$\mathcal{I}(\mathbb{X}) = (x_1 - c_1, ..., x_n - c_n).$$

Proposition 3.3.3 (Characterisation of Ideals of Points)**.** *Let $\mathbb{X} = \{p_1, ..., p_s\} \subseteq K^n$ be an affine point set. Then $\mathcal{I}(\mathbb{X})$ can be computed as the intersection of the vanishing ideals of all the individual points:*

$$\mathcal{I}(\mathbb{X}) = \mathcal{I}(p_1) \cap ... \cap \mathcal{I}(p_s).$$

Proof. Obviously a polynomial is in $\mathcal{I}(\mathbb{X})$ if and only if it vanishes at every point p_i in $\mathbb{X}$. □

Definition 3.3.4. [Evaluation map]

Let $\mathbb{X} = \{p_1, ..., p_s\} \subseteq K^n$ be an affine point set. By $\text{eval}_{\mathbb{X}} : P \to K^s$ defined by $\text{eval}_{\mathbb{X}}(f) = (f(p_1), ..., f(p_s))$ we denote the K-linear **evaluation map** associated with $\mathbb{X}$.

Calculating ideals of points via intersections of ideals is not very efficient if done via the computation of Gröbner bases (compare [45, Proposition 3.2.7]). In the following section we study a direct algorithm which uses linear algebra and has a guaranteed polynomial runtime.

3.4 The Buchberger-Möller Algorithm for Border Bases

The following algorithm was first introduced in [21] but specifically given for Gröbner bases. We present the border basis version given in [32] as Algorithm 4.1. Additionally, we give a detailed proof which shows that the algorithm computes a border basis of $\mathcal{I}(\mathbb{X})$, as the details of the proof were left to the reader in [32].

Algorithm 18: The Buchberger-Möller (BM) algorithm for border bases

Input: A degree compatible term ordering σ on $\mathbb{T}^n$, an affine point set $\mathbb{X} = \{p_1, ..., p_s\} \subseteq K^n$

Output: $(G, \mathcal{O})$ where $\mathcal{O}$ is an order ideal and G an $\mathcal{O}$-border basis of $\mathcal{I}(\mathbb{X})$

1 $d := 1$, $\mathcal{O} := [1]$, $G := [\,]$, $M = (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(K)$;
2 $L := [t_1, ..., t_\ell] :=$ all terms of degree 1 ordered decreasingly w.r.t. σ;
3 **repeat**
4 $\quad A := (\mathrm{eval}_{\mathbb{X}}(t_1), ..., \mathrm{eval}_{\mathbb{X}}(t_\ell), M)$;
5 $\quad B := \ker(A)$ such that the rows of B form a basis of the kernel of A;
6 $\quad B' := \mathrm{RREF}(B)$;
$\quad$ // We assume that $\mathcal{O} = [o_1, ..., o_{\mathrm{len}(\mathcal{O})}]$
7 $\quad \tilde{G} := [\tilde{g}_1, ..., \tilde{g}_k]^{\mathrm{tr}} = B'\left(t_1, ..., t_\ell, o_1, ..., o_{\mathrm{len}(\mathcal{O})}\right)^{\mathrm{tr}}$;
8 $\quad G := \mathrm{concat}(G, \tilde{G}^{\mathrm{tr}})$;
9 $\quad$ **for** $i := \ell$ **downto** *1* **do**
10 $\quad\quad$ **if** $t_i \neq \mathrm{LT}_\sigma(\tilde{g}_1)$ **and** ... **and** $t_i \neq \mathrm{LT}_\sigma(\tilde{g}_k)$ **then**
11 $\quad\quad\quad \mathcal{O} := \mathrm{concat}([t_i], \mathcal{O})$;
12 $\quad\quad\quad M := (\mathrm{eval}(t_i), M)$;
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 $\quad d := d + 1$;
16 $\quad L := [t_1, ..., t_\ell] :=$ all terms of degree d in $\partial\mathcal{O}$ ordered decreasingly w.r.t. σ;
17 **until** $L = [\,]$;
18 **return** $(G, \mathcal{O})$;

Theorem 3.4.1. *This is an algorithm which computes in a finite number of steps an $\mathcal{O}$-border basis G of $\mathcal{I}(\mathbb{X})$.*

Proof. We will start by proving the finiteness of the algorithm. In line 4 the matrix M has $|\mathcal{O}|$ columns and the new columns which are added in line 12 are linearly independent of the previous ones. Obviously it is not possible to add linearly independent columns infinitely many times as when the number of columns exceeds the number of rows their column space can no longer be linearly independent. So M can have at most s columns. Then in matrix A all columns associated with t_1 to t_ℓ are linearly dependent of the columns of M, thus no new terms will go into $\mathcal{O}$. As a consequence L will be empty in line 17 and the algorithm will terminate.

Now we prove that G is an $\mathcal{O}$-border basis. First we show that the set $\mathcal{O}$ is an order ideal of terms. For this purpose we only need to consider elements of $\mathcal{O}$ which are at least quadratic, as 1 is always included in $\mathcal{O}$ (see line 1) all divisors of the terms of degree 1 are trivially included in $\mathcal{O}$. So let $x_i t \in \mathcal{O} \setminus \{1, x_1, ..., x_n\}$. We have to prove that all divisors of $x_i t$ are in $\mathcal{O}$. Let us assume this is not the case so there exists an indeterminate x_j such that $t = x_j t'$ and $x_i t' = \mathrm{LT}_\sigma(g)$ for some polynomial g in G. So $x_i t'$ would be the divisor of $x_i t$ missing in the order ideal. We know that $\mathrm{eval}_{\mathbb{X}}(g) = 0_s$ and consequently if we multiply by x_j, then $\mathrm{eval}_{\mathbb{X}}(x_j g) = 0_s$ as well. Additionally, we know that $x_i t = \mathrm{LT}_\sigma(x_j g)$. However, the terms $\mathrm{supp}(x_j g) \setminus \{x_i t\}$ are

either in $\mathcal{O}$ or can be rewritten via relations in G in terms of $\mathcal{O}$, because we are using a degree compatible term ordering. This means we can find suitable coefficients $c_i \in K$ in lines 5 and 6 such that we can form the polynomial $x_j g = x_i t - \sum_{i=1}^{|\mathcal{O}|} c_i o_i$ from the elements already in $\mathcal{O}$ together with $x_i t$. So $x_i t$ would not be an element of the order ideal, which is a contradiction to our initial assumption. This proves that $\mathcal{O}$ is always an order ideal.

It is easy to see that by construction G is always an $\mathcal{O}$-border prebasis.

Via Buchberger's criterion for border bases (see Theorem 3.1.15) we can prove that G is an $\mathcal{O}$-border basis. Therefore, it remains to be shown that for all neighbour pairs (i, j) the S-polynomials reduce to zero via G. Let $g_i = b_i + h_i$ with $\operatorname{supp}(h_i) \subseteq \mathcal{O}$.

We will first look at the across-the-street neighbours. So $S_{ij} = x_k g_i - x_\ell g_j$ and $\operatorname{NR}_{\mathcal{O},G}(S_{ij}) = x_k g_i - x_\ell g_j - \sum_\nu c_\nu g_\nu$ with $c_\nu \in K$ and $1 \leq \nu \leq |G|$. We obtain the expression

$$\begin{aligned} \operatorname{NR}_{\mathcal{O},G}(S_{ij}) &= x_k(b_i + h_i) - x_\ell(b_j + h_j) - \sum_\nu c_\nu g_\nu \\ &= x_k h_i - x_\ell h_j - \sum_\nu c_\nu g_\nu. \end{aligned}$$

We choose the c_v in such a way that all border terms in $x_k h_i$ and $x_l h_j$ cancel out. Thus $\operatorname{supp}(\operatorname{NR}_{\mathcal{O},G}(S_{ij})) \subseteq \mathcal{O}$. We know that $\operatorname{eval}_{\mathbb{X}}(S_{ij}) = 0_s$ because $\operatorname{eval}_{\mathbb{X}}(g_i) = 0_s$ for all $1 \leq i \leq |G|$. If we assume that $\operatorname{NR}_{\mathcal{O},G}(S_{ij}) \neq 0$ for at least one pair (i, j) this means that we have found a new relation among the elements in $\mathcal{O}$ which vanishes on the set $\mathbb{X}$. However, this is not possible as the algorithm assures that only elements are added to $\mathcal{O}$ which preserve linear independence among the columns of the evaluation matrix of $\mathcal{O}$ with respect to $\mathbb{X}$.

Finally, we will analyse the next-door neighbours. So we may assume that $S_{ij} = g_i - x_k g_j$ and $\operatorname{NR}_{\mathcal{O},G}(S_{ij}) = g_i - x_k g_j - \sum_\nu c_\nu g_\nu$ with $c_\nu \in K$ and $1 \leq \nu \leq |G|$. Here we obtain the expression

$$\begin{aligned} \operatorname{NR}_{\mathcal{O},G}(S_{ij}) &= (b_i + h_i) - x_l(b_j + h_j) - \sum_\nu c_\nu g_\nu \\ &= h_i - x_l h_j - \sum_\nu c_\nu g_\nu. \end{aligned}$$

The conclusion follows from exactly the same arguments as in the case of the across-the-street neighbours. □

Remark 3.4.2. Even if we would not require $\mathbb{X}$ to be a set of points and would allow duplicates the output of the algorithm would remain unchanged. The reason is that adding a linearly dependent row to matrix A in line 4 will not effect the computation of the kernel of A in line 5. For reasons of efficiency it makes sense, though, to require that $\mathbb{X}$ is a set. Compare also Example 4.3.9, where we investigate the effects of multiple identical points on the output of the ABM algorithm.

Let us demonstrate the workings of the algorithm by means of the following example.

Example 3.4.3. Let $P = \mathbb{R}[x_1, x_2]$ and $\mathbb{X} = \{(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)\}$ be given and let σ be the `DegRevLex` term ordering.

- $d = 0$, $\mathcal{O} = [1]$, $G = [\,]$, and $M = (1, ..., 1)^{\mathrm{tr}}$
- $d = 1$, $\partial\mathcal{O} = \{x_1, x_2\}$, and $L = [x_1, x_2]$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 1 \end{pmatrix}$, $\ker(A) = \{0_3\}$, $\mathcal{O} = [x_1, x_2, 1]$
- $d = 2$, $\partial\mathcal{O} = \{x_1^2, x_1x_2, x_2^2\}$, and $L = [x_1^2, x_1x_2, x_2^2]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0.25 & 0.25 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $\ker(A) = \{(1, 0, -1, -1, 1, 0)\}$
- $(1, 0, -1, -1, 1, 0)\left(x_1^2, x_1x_2, x_2^2, x_1, x_2, 1\right)^{\mathrm{tr}} = x_1^2 - x_2^2 - x_1 + x_2 = g_1$, $G = [g_1]$, $\mathcal{O} = [x_1x_2, x_2^2, x_1, x_2, 1]$
- $d = 3$, $\partial\mathcal{O} = \{x_1^2x_2, x_1x_2^2, x_2^3\}$, and $L = [x_1^2x_2, x_1x_2^2, x_2^3]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0.125 & 0.125 & 0.125 & 0.25 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$,
  $\ker(A) = \{(1, -1, 0, 0, 0, 0, 0, 0), (0, -1, 1, 1, -1, 0, 0, 0), (0, 0, 2, 0, -3, 0, 1, 0)\}$
- $B = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & -3 & 0 & 1 & 0 \end{pmatrix}$,
  $\mathrm{RREF}(B) = \begin{pmatrix} 1 & 0 & 0 & -1 & -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & -1 & -0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & -1.5 & 0 & 0.5 & 0 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 & 0 & -1 & -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & -1 & -0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & -1.5 & 0 & 0.5 & 0 \end{pmatrix} \left(x_1^2x_2, x_1x_2^2, x_2^3, x_1x_2, x_2^2, x_1, x_2, 1\right)^{\mathrm{tr}}$

$$\begin{aligned} g_2 &= x_1^2x_2 - x_1x_2 - 0.5x_2^2 + 0.5x_2 \\ g_3 &= x_1x_2^2 - x_1x_2 - 0.5x_2^2 + 0.5x_2 \\ g_4 &= x_2^3 - 1.5x_2^2 + 0.5x_2 \end{aligned}$$

- $G = [g_1, g_2, g_3, g_4]$, $\mathcal{O} = [x_1x_2, x_2^2, x_1, x_2, 1]$

In Figure 3.3 we can see a visualisation of the zero set of g_1 to g_4.

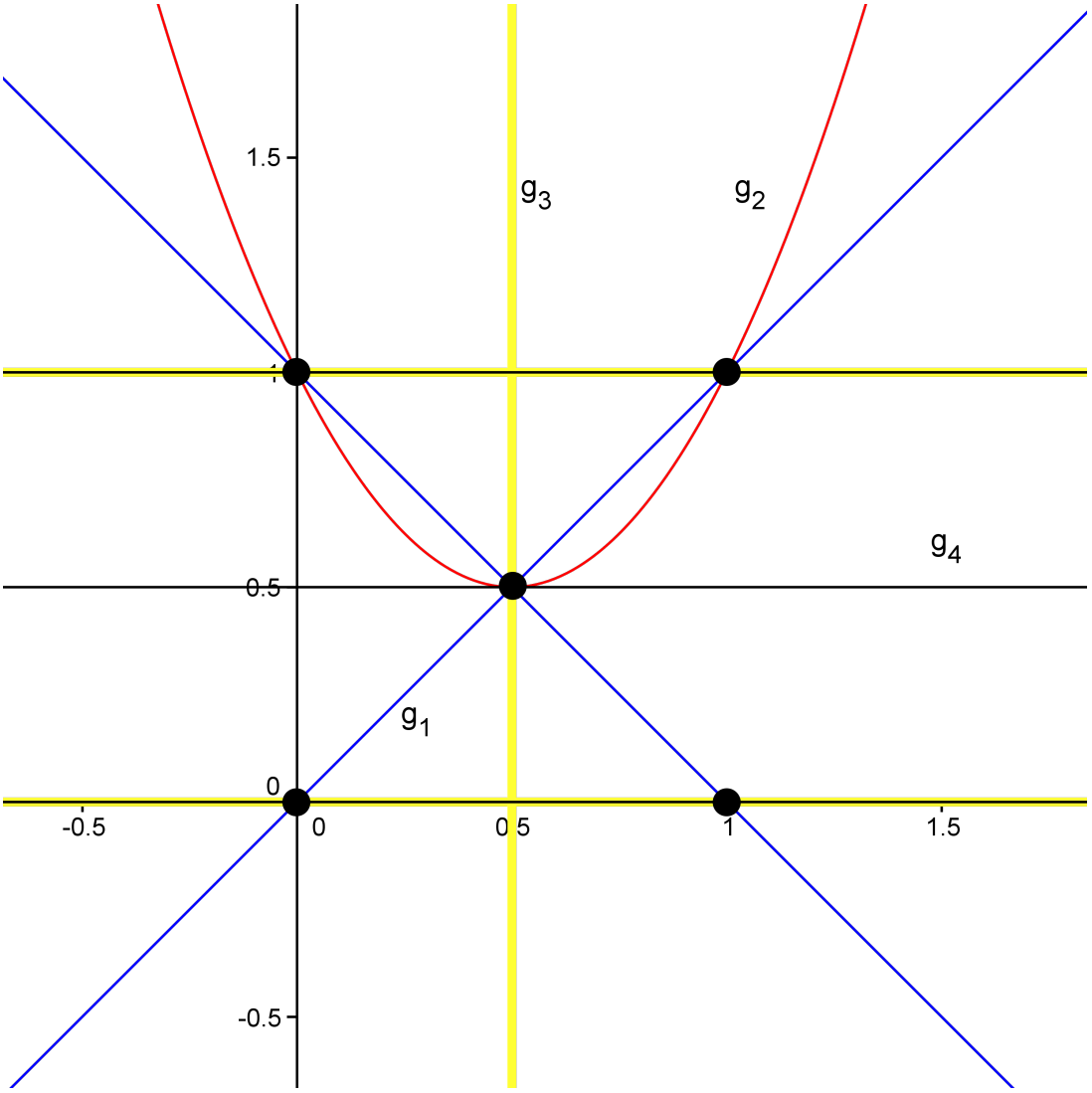

Figure 3.3: A visualisation of the polynomials $g_1, ..., g_4$ of Example 3.4.3

3.4.1 Runtime Analysis of the Buchberger-Möller Algorithm

In this section we analyse the runtime of the Buchberger-Möller algorithm for border bases in case we are dealing with a set of generically chosen points. This means that no special low degree relations among the points exist and the kernel computed in line 5 of Algorithm 18 will be trivial unless we arrive at the situation where the number of rows exceeds the number of columns.We will look at each step in detail this time, because the following algorithms which we present share a lot of common structure with the BM algorithm. Later we will only discuss the differing parts. We also explain in this subsection how it is possible to speed up the implementation by updating the kernel rather then completely recomputing it in every degree. This principle will also carry over to the later chapters where we will discuss how the approximate kernel of a matrix can be updated.

Remark 3.4.4. The following cost measures only represent the number of basic operations in terms of addition, subtraction, multiplication, and division in the field K. Depending on the base field the actual cost for the basic operations can be tremendous if implemented in a naive way. Therefore, it should be made sure that an implementation is used that guarantees at least polynomial runtime. For example, an efficient implementation for $K = \mathbb{Q}$ in the context of Gröbner Bases is presented in [27].

Fortunately, in our case we are mostly interested in floating point implementations, for which all basic operations are guaranteed to be in $O(1)$.

Remark 3.4.5. [Updating the kernel of a matrix]

Before we begin with the actual analysis of Algorithm 18, we will explain how the kernel in line 5 of the algorithm can be computed in a more economic way. Please recall from Proposition 2.4.4 that the kernel of a dense matrix $A \in \mathrm{Mat}_{m,n}(K)$ can be computed efficiently using the Gauss Jordan algorithm in $O(\frac{1}{2}\min(m,n)^2 \max(m,n))$. However, if additionally the kernel of a slightly modified matrix, just like in our case where we are pre-pending new columns to A in line 4, shall be computed, it becomes inefficient to recompute the whole kernel. In this case it makes sense to store the information about the individual reduction steps to reduced row echelon form, as these reoccur in the computation of the modified matrix. Bookkeeping during a Gauss-Jordan reduction essentially amounts to computing the PLURQ decomposition (see Definition 2.5.3) of A, where $P \in \mathrm{Mat}_m(K)$ and $Q \in \mathrm{Mat}_n(K)$ are permutation matrices, $L \in \mathrm{Mat}_{m,k}(K)$ is a lower triangular matrix which encodes the reduction part to row echelon form and $U \in \mathrm{Mat}_k(K)$ is an upper triangular matrix which encodes the reduction to reduced row echelon form, and $R \in \mathrm{Mat}_{k,n}(K)$ is A transformed to RREF. Once this decomposition is available it can be updated essentially in quadratic time in each subsequent computation. Please note that the matrices U and L never need to be formed explicitly. We will not spell out the algorithm in all details but we sketch the basic methodology. What we will not discuss is the proper handling of row and column permutations which can become necessary to guarantee lower and upper triangular shape of the matrices L and U.

Let $M \in \mathrm{Mat}_{m,n}(K)$ be an arbitrary matrix and let $\tilde{M} \in \mathrm{Mat}_{m,n+\ell}(K)$ be matrix M to which ℓ new columns $c_1, ..., c_\ell$ were added. The first thing that we will do is to determine a permutation matrix P_1 such that all new columns are moved to the back, meaning that they are now at positions $n+1$ to $n+\ell$ of $\tilde{M}P_1$. Let us now additionally assume that we have previously obtained a PLURQ decomposition of M, such that $U^{-1}L^{-1}P^{\mathrm{tr}}MQ^{\mathrm{tr}} = R$. If we compute $U^{-1}L^{-1}P^{\mathrm{tr}}(c_1...c_\ell)Q^{\mathrm{tr}}$ we reproduce the reductions which we have previously applied while computing the RREF of M. The cost is in $O(\ell m^2)$. What remains to be done is to restore the RREF of the whole matrix. Only the new columns have to be treated, this costs $O(\frac{1}{2}\min(m,\ell)^2 \max(m,\ell))$ operations. With the new transformations we must update our PLURQ decomposition, such that we obtain a full decomposition of $\tilde{M}$. Note that no full matrix-matrix multiplications are necessary. These would in fact cost $O(m^3)$ and would make the whole updating process pointless. In fact we need to do no further processing as it is sufficient to only store the individual transformations per row which we have computed while forming the RREF. This amounts to an additional cost of $O(\ell m)$. Once we have computed

the RREF of $\tilde{M}$ it is again easy to read off the kernel. When interpreting the entries of the kernel vectors as coefficients of our terms we only have to take care of the permutations which we have applied.

In the following example we will show how the kernel of a matrix can be updated. The matrices are taken from Example 3.4.3.

Example 3.4.6. Let

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 1 \end{pmatrix}.$$

Let us now assume that we have in a previous step computed a PLURQ decomposition of this matrix. We have thus obtained the RREF of A by forming

$$U^{-1}L^{-1}P^{\mathrm{tr}}AQ^{\mathrm{tr}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

where

$$U^{-1} = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; L^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & 1 & 0 \\ -0.5 & -0.5 & 0 & 0 & 1 \end{pmatrix},$$

$$P^{\mathrm{tr}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } Q^{\mathrm{tr}} = I_3.$$

The kernel just contains the zero vector as the matrix has full rank. In the next step we add three new leading columns to A. We have

$$\tilde{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0.25 & 0.25 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}.$$

We only need to compute $U^{-1}L^{-1}P^{\mathrm{tr}}\,(c_1,...,c_l)\,Q^{\mathrm{tr}}$. Additionally, we find a new suitable permutation matrix Q^{tr} and obtain

$$U^{-1}L^{-1}P^{\mathrm{tr}}\tilde{A}\begin{pmatrix}0&0&0&0&1&0\\0&0&0&1&0&0\\0&0&0&0&0&1\\0&0&1&0&0&0\\0&1&0&0&0&0\\1&0&0&0&0&0\end{pmatrix}=\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&1\\0&0&1&0&1&0\\0&0&0&1&0&0\\0&0&0&0.25&-0.25&-0.25\end{pmatrix}.$$

We restore row echelon form by applying the transformations

$$\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&-0.25&1\end{pmatrix}\quad\text{and}\quad\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&-4\end{pmatrix}$$

and update L^{-1} accordingly (no full matrix-matrix multiplications are required). In order to obtain reduced row echelon form we apply

$$\begin{pmatrix}1&0&0&0&-1\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$$

and update U^{-1}. Finally, we obtain

$$\begin{aligned}U^{-1}L^{-1}P^{\mathrm{tr}}\tilde{A}Q^{\mathrm{tr}} &= \begin{pmatrix}1&0&-1&0&-1\\0&1&-1&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\-1&-1&1&1&0\\1&1&1&1&-4\end{pmatrix}P^{\mathrm{tr}}\tilde{A}Q^{\mathrm{tr}}\\ &= \begin{pmatrix}1&0&0&0&0&-1\\0&1&0&0&0&1\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&1\end{pmatrix}.\end{aligned}$$

So $Q^{\mathrm{tr}}\,(-1,1,0,0,1,-1)^{\mathrm{tr}}=(1,0,-1,-1,1,0)^{\mathrm{tr}}$, and $\ker(\tilde{A})=\{(1,0,-1,-1,1,0)\}$ just as we had computed previously in Example 3.4.3.

Proposition 3.4.7. *The average runtime complexity of Algorithm 18 is cubic in the number of input points s, if the kernel is updated as described in Remark 3.4.5.*

Proof. For the sake of simplicity, but without sacrificing a lot of generality, let us assume that the number of input points is

$$s = \sum_{i=0}^{p} \binom{n+i-1}{i} = \binom{n+p}{p}$$

which is the number of terms up to and including degree p in the polynomial ring $P = K[x_1, ..., x_n]$. The "average case" in terms of runtime complexity, in which we are interested, will occur if the set of input points has random coordinates and is therefore generic. As such the order ideal $\mathcal{O}$ will contain s elements and the border of $\mathcal{O}$ will contain $\binom{n+p}{p+1}$ elements after the algorithm has terminated. We will now have a closer look at all individual steps:

Lines 1 and 2 are only executed once and have a cost of $O(s+n)$.
Lines 4-16 are inside a loop which will be executed exactly $p+1$ times. This holds because the points are generic and no special relation between them can be found until the number of columns exceeds the number of rows in the evaluation matrix.
The cost of line 4 is in $O\left(s\binom{n+d-1}{d}\right)$, as all terms in the current degree d need to be evaluated on all points. Compare [46, Remark 6.3.12].
In line 5 the kernel of a $\mathrm{Mat}_{s,\binom{n+d}{d}}(K)$ matrix is computed. If done via Gauss-Jordan elimination (2.4.4) the cost would be in $O\left(s\binom{n+d}{d}^2\right)$ as long as $d \leq p$ and in case $d = p+1$ the cost will be in $O\left(s^2\binom{n+d}{d}\right)$. However, if we use Remark 3.4.5 to update the kernel consecutively the cost will be in $O\left(s\binom{n+d-1}{d}\binom{n+d-1}{d-1}\right)$. The result of this computation, matrix B, is generally empty, unless $d = p+1$. In this case matrix B will have $\binom{n+p}{p+1}$ rows and line 6 will cost $O\left(\binom{n+p}{p+1}^2\binom{n+p+1}{p+1}\right)$, if the reduced row echelon form is computed via Gauss-Jordan elimination (2.4.5), and line 7 will be in $O\left(\binom{n+p}{p+1}\binom{n+p+1}{p+1}\right)$ which is the cost of a matrix-vector multiplication.
The cost of line 8 is in $O(1)$.
The loop which spans from line 9 to 14 has costs which are in $O\left(\binom{n+d-1}{d}\right)$, essentially because every term has to be dealt with. The check inside the loop can be handled in constant time because either G is empty or all terms in L are leading terms of the elements in G.
Line 15 can again be handled in constant time.
The cost of line 16 is in $O\left(\binom{n+d}{d+1}\right)$ as long as $d \leq p$, because all terms of degree $d+1$ in n indeterminates are computed in this step. So if we put all parts together we have total costs of

$$O\left(\underbrace{s}_{1} + \underbrace{n}_{2} + \sum_{d=1}^{p+1}\left(\underbrace{s\binom{n+d-1}{d}}_{4} + \underbrace{1}_{8} + \underbrace{\binom{n+d-1}{d}}_{9-14}\right)\right)$$
$$+ \quad O\left(\sum_{d=1}^{p}\left(\underbrace{s\binom{n+d}{d}^2}_{5} + \underbrace{\binom{n+d}{d+1}}_{16}\right)\right)$$

$$+ \quad O\left(\underbrace{s^2\binom{n+p+1}{p+1}}_{5} + \underbrace{\binom{n+p}{p+1}^2\binom{n+p+1}{p+1}}_{6} + \underbrace{\binom{n+p}{p+1}\binom{n+p+1}{p+1}}_{7} + \underbrace{1}_{15}\right),$$

if we compute the whole kernel in each step. This measure changes to

$$O\left(\sum_{d=1}^{p+1}\left(\underbrace{s\binom{n+d-1}{d}}_{4} + \underbrace{s\binom{n+d-1}{d}\binom{n+d-1}{d-1}}_{5} + \underbrace{1}_{8} + \underbrace{\binom{n+d-1}{d}}_{9-14}\right)\right)$$
$$+ \quad O\left(\underbrace{s}_{1} + \underbrace{n}_{2} + \sum_{d=1}^{p}\underbrace{\binom{n+d}{d+1}}_{16}\right)$$
$$+ \quad O\left(\underbrace{\binom{n+p}{p+1}^2\binom{n+p+1}{p+1}}_{6} + \underbrace{\binom{n+p}{p+1}\binom{n+p+1}{p+1}}_{7} + \underbrace{1}_{15}\right)$$

if we update the kernel. We can now observe that the dominating cost factor is always the computation of the kernel of A inside the loop. All other factors can be ignored in the big O notation. This means that the runtime of the algorithm is in

$$O\left(\sum_{d=1}^{p} s\binom{n+d}{d}^2 + s^2\binom{n+p+1}{p+1}\right)$$
$$= O\left(\sum_{d=1}^{p} s\binom{n+d}{d}^2 + s^2\left(\binom{n+p}{p}\left(1+\frac{n}{p}\right)\right)\right) \tag{3.1}$$

or in

$$O\left(\sum_{d=1}^{p+1} s\binom{n+d-1}{d}^2\right) \tag{3.2}$$

respectively.

We can further simplify expression 3.1 using the inequalities $s\binom{n+d}{d}^2 \leq s^2\binom{n+d}{d}$ for $d \leq p$ and obtain

$$\begin{aligned}
& O\left(\sum_{d=1}^{p} s^2\binom{n+d}{d} + s^3\left(1+\frac{n}{p}\right)\right) \\
= \quad & O\left(s^2\binom{n+p+1}{p} - s^2 + s^3\left(1+\frac{n}{p}\right)\right) \\
= \quad & O\left(s^3\left(1+\frac{p}{n+1}\right) - s^2 + s^3\left(1+\frac{n}{p}\right)\right) \\
= \quad & O\left(s^3\left(2+\frac{p}{n+1}+\frac{n}{p}\right)\right).
\end{aligned}$$

The only thing that remains to be done, is to derive a lower and upper bound for p. We know that $1 \leq p$. Additionally, we know that $\left(\frac{n+p}{n}\right)^n \leq s$ so $\sqrt[n]{s}n - n \geq p$ (which is only a very crude bound). Thus we finally obtain the runtime estimate

$$O\left(s^3\left(2 + \frac{p}{n+1} + \frac{n}{p}\right)\right) = O\left(s^3\left(2 + \frac{n\left(\sqrt[n]{s} - 1\right)}{n+1} + n\right)\right).$$

Thus the dominating cost factor is $s^3\left(2 + \frac{n\left(\sqrt[n]{s}-1\right)}{n+1} + n\right)$.

We will conclude our analysis by further investigating expression 3.2. We can use the inequality $s\binom{n+d-1}{d}\binom{n+d-1}{d-1} \leq s^2\binom{n+d-1}{d}$ for $d \leq p+1$ and obtain

$$\begin{aligned} & O\left(\sum_{d=1}^{p+1} s\binom{n+d-1}{d}\binom{n+d-1}{d-1}\right) \\ = \; & O\left(\sum_{d=1}^{p} s^2\binom{n+d-1}{d} + s\binom{n+p}{p}\binom{n+p}{p+1}\right) \\ = \; & O\left(s^3 + s^3\frac{n}{p}\right) = O\left(s^3\left(1+n\right)\right). \end{aligned}$$

So the dominating cost factor here is $O\left(s^3\right)$ which shows that the algorithm is cubic in the number of input points. □

3.4.2 Implementation in ApCoCoA

The Buchberger-Möller algorithm for border bases was implemented by the author in the ApCoCoA library ([20]). The updating techniques described in Remark 3.4.5 are, at the time of writing, not implemented. The algorithm is built around the data types of the CoCoA library ([19]) but uses a custom (but not optimised) implementation for the computation of the kernel. Detailed usage instructions can be found in Appendix 8.1.

3.4.3 Basis Transformation

Given a finite set of points $\mathbb{X} \subset \mathbb{C}^n$ and a degree compatible term ordering σ, the Buchberger-Möller algorithm returns the order ideal $\mathcal{O} = \mathcal{O}_\sigma(\mathcal{I})$ and the $\mathcal{O}$-border basis of $\mathcal{I}(\mathbb{X})$. This means that we can influence the computed $\mathcal{O}$ via the chosen term ordering σ. However, we cannot directly control which elements will be contained in $\mathcal{O}$. Because of special requirements it is sometimes necessary to transform a given $\mathcal{O}$-border basis G for an ideal of points $\mathcal{I}(\mathbb{X})$ into a representation with respect to a different order ideal $\tilde{\mathcal{O}} = \left\{\tilde{t}_1, ..., \tilde{t}_\mu\right\}$. We can use the fact that we know the set $\mathbb{X}$ to our advantage, such that this task can be performed using only techniques from linear algebra. First we need to make sure that $|\mathcal{O}| = |\tilde{\mathcal{O}}|$. The second requirement is that the evaluation matrix A of $\mathbb{X}$ with respect to $\tilde{\mathcal{O}}$ is of full rank. Now let $\{\tilde{b}_1, ..., \tilde{b}_\nu\} = \partial\tilde{\mathcal{O}}$. Once these conditions are satisfied A is always invertible and it is possible to solve the linear system $Ax = \text{eval}_{\mathbb{X}}(\tilde{b}_i)$ for all $1 \leq i \leq \nu$ and to obtain in this way the coefficients of the polynomials $\tilde{g}_i$.

Algorithm 19: Border Basis Transformation

Input: A set of points $\mathbb{X} = \{p_1, ..., p_\mu\}$, an $\mathcal{O}$-border basis G for $\mathcal{I}(\mathbb{X})$, and an order ideal $\tilde{\mathcal{O}}$ with $|\tilde{\mathcal{O}}| = |\mathcal{O}|$

Output: An $\tilde{\mathcal{O}}$-border basis of $\mathcal{I}(\mathbb{X})$, if it exists

1 $A := \mathrm{eval}_{\mathbb{X}}(\tilde{\mathcal{O}})$;
2 **if** $\ker(A) \neq \{0_\mu\}$ **then**
3 | **return** No $\tilde{\mathcal{O}}$-border basis exists for $\mathcal{I}(\mathbb{X})$;
4 **end**
5 $B := A^{-1}$;
6 $\{\tilde{b}_1, ..., \tilde{b}_\nu\} := \partial\tilde{\mathcal{O}}$;
7 **for** *i := 1* **to** ν **do**
8 | $\tilde{g}_i := \tilde{b}_i - B\,\mathrm{eval}_{\mathbb{X}}(\tilde{b}_i)\left(\tilde{t}_1, ..., \tilde{t}_\mu\right)^{\mathrm{tr}}$;
9 **end**
10 $\tilde{G} := \{\tilde{g}_1, ..., \tilde{g}_\nu\}$;
11 **return** $\tilde{G}$;

Proposition 3.4.8. *This is an algorithm which transforms an $\mathcal{O}$-border basis G for an ideal of points $\mathcal{I}(\mathbb{X})$ into an $\tilde{\mathcal{O}}$-border basis $\tilde{G}$, if it exists.*

Proof. First of all $\mathcal{O}$ and $\tilde{\mathcal{O}}$ need to contain the same number of elements. This is already made sure by the requirements we have imposed on the input of the algorithm. An $\tilde{\mathcal{O}}$-border basis for $\mathcal{I}(\mathbb{X})$ only exists, if the evaluation matrix $A \in \mathrm{Mat}_\mu(K)$ of the order ideal $\tilde{\mathcal{O}}$ with respect to $\mathbb{X}$ is of full rank which is checked in line 2 of the algorithm. Then it is possible to invert A and to compute the coefficients of the polynomials $\tilde{g}_i$ by letting $\tilde{g}_i = \tilde{b}_1 - A^{-1}\,\mathrm{eval}_{\mathbb{X}}\left(\tilde{b}_1\right)\left(\tilde{t}_1, ..., \tilde{t}_\mu\right)^{\mathrm{tr}}$. By construction the polynomials $\tilde{g}_i$ form a border prebasis. The polynomials also form a border basis as otherwise new vanishing polynomials could be constructed with the help of the S-polynomials of the neighbouring elements in $\partial\mathcal{O}$, which only contain elements in $\tilde{\mathcal{O}}$. However, this is not possible as the kernel of A was trivial. □

Remark 3.4.9. If $K = \mathbb{Q}[i]$, the algorithm can be implemented with exact arithmetic for instance in CoCoA. We have used such an implementation for the computations in the rational recovery chapter (5).

Remark 3.4.10. In Algorithm 19 we have used the fact that we know $\mathbb{X}$. Please note that it is also possible to transform a given $\mathcal{O}$-border basis G into a $\tilde{\mathcal{O}}$-border basis if $\tilde{\mathcal{O}}$ supports a border basis of $\langle G \rangle$ even if $\mathbb{X}$ is not known. An algorithm for this task is contained in [23, Proposition 5]. However, if $\mathbb{X}$ is known, the algorithm given here should be used as it is more efficient.

Example 3.4.11. Let us start with the border basis which was computed in Example 3.4.3. For the set of points $\mathbb{X} = \{(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)\}$ together with the `DegRevLex` term

ordering we had obtained $\mathcal{O} = \{1, x_2, x_1, x_2^2, x_1x_2\}$ and $G = \{g_1, ..., g_4\}$ with

$$\begin{aligned} g_1 &= x_1^2 - x_2^2 - x_1 + x_2, \\ g_2 &= x_1^2x_2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_3 &= x_1x_2^2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_4 &= x_2^3 - 1.5x_2^2 + 0.5x_2. \end{aligned}$$

If we let $\tilde{\mathcal{O}} = \{1, x_2, x_1, x_1x_2, x_1^2\}$ and apply Algorithm 19 we obtain $\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_4\}$ with

$$\begin{aligned} \tilde{g}_1 &= x_2^2 - x_1^2 + x_1 - x_2, \\ \tilde{g}_2 &= x_1^2x_2 - x_1x_2 - 0.5x_1^2 + 0.5x_1, \\ \tilde{g}_3 &= x_1x_2^2 - x_1x_2 - 0.5x_1^2 + 0.5x_1, \\ \tilde{g}_4 &= x_1^3 - 1.5x_1^2 + 0.5x_1. \end{aligned}$$

The result is not too surprising as the points in $\mathbb{X}$ have symmetric coordinates. This also explains that no result can be obtained if we let $\tilde{\mathcal{O}} = \{1, x_2, x_1, x_2^2, x_1^2\}$, as $\text{eval}_{\mathbb{X}}(\tilde{\mathcal{O}})$ does not have full rank.

4 The AVI/ABM Family of Algorithms

Contents

In the following sections let $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in K^n$ be a (finite) tuple of affine points and let $P = K[x_1, ..., x_n]$ be a polynomial ring in n indeterminates with $K = \mathbb{R}$ or $K = \mathbb{C}$. By $\|\cdot\|$ we denote the Euclidean norm, unless stated otherwise.

The aim of the Approximate Vanishing Ideal (AVI) algorithm and the Approximate Buchberger-Möller (ABM) algorithm, which we will present in this chapter, is to generalise the BM algorithm (18) for border bases to allow for approximate relations among the coordinates of the points in $\mathbb{X}$. This is a practical requirement as we are dealing with noisy measurements and in such an environment exact interpolation would not yield any physically meaningful results. So the algorithms that we give in this chapter are concerned with the efficient and numerically stable computation of "approximate" border bases with respect to $\mathbb{X}$.

We start with the definition of approximate border bases as a generalisation of exact border bases. Then we introduce the concept of the approximate kernel of a matrix via its SVD and explore some of its properties which are relevant for the AVI algorithm. After these preparations we detail the AVI algorithm and afterwards some of its shortcomings. These are addressed via different flavours of the ABM algorithm in Sections 4.3, 4.4, and 4.5, which represent our main contribution to the subject of approximate border bases. In order to be able to describe the algorithms, we explain how the approximate kernel of a matrix and the solution of the homogeneous least squares problem are related and how we can exploit this to speed up the computations. Finally, we discuss the properties and differences of the algorithms and provide a comprehensive runtime analysis.

In Section 4.7 we compare our approach to other state of the art algorithms. Via a couple of example computations we mange to show that we can significantly outperform the SOI and NBM algorithms of Fassino et al. that were proposed in [34] and [35], while we obtain similar or even better results for the same input data.

4.1 Approximate Border Bases

Similarly to [28] we introduce the following definitions:

Definition 4.1.1. [ε-approximate vanishing]

Given $\varepsilon \geq 0$, a polynomial $f \in P$ is said to be **ε-approximately vanishing** with respect to $\mathbb{X}$ if the Euclidean norm of the image of its evaluation map (see Definition 3.3.4) associated with $\mathbb{X}$ is less than or equal to ε. We write

$$\|\text{eval}_{\mathbb{X}}(f)\| \leq \varepsilon.$$

Please note that, in contrast to [28], we also allow $\varepsilon = 0$, which will make it possible to handle the exact and the approximate case in one algorithm. All other definitions have been adapted accordingly.

Definition 4.1.2. [Unitary polynomial]

A polynomial $f \in P$ is called **unitary** if the norm of its coefficient vector equals one.

Definition 4.1.3. [ε-approximate vanishing ideal]

Given $\varepsilon \geq 0$, an ideal $J \subseteq P$ is called an **ε-approximate vanishing ideal** with respect to $\mathbb{X}$ if there exists a system of unitary generators G of J such that each element of G vanishes ε-approximately with respect to $\mathbb{X}$.

In an algebraic sense, for $\varepsilon > 0$, an ε-approximate vanishing ideal is in general the unit ideal. This is one of the reasons why we try to construct proper ideals which are "close" to the approximate ones in Chapter 5.

Definition 4.1.4. [ε-approximate border basis]

Let $\mathcal{O} = \{t_1, ..., t_\mu\} \subseteq \mathbb{T}^n$ be an order ideal of terms, let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be its border, and let $G = \{g_1, ..., g_\nu\}$ be an $\mathcal{O}$-border prebasis of the ideal $I = \langle g_1, ..., g_\nu \rangle$ in P. This means that g_j is of the form $g_j = b_j - \sum_{i=1}^{\mu} c_{ij} t_i$ with $c_{ij} \in \mathbb{C}$. For every pair (i, j) such that b_i and b_j are neighbours in $\partial\mathcal{O}$ we compute the normal remainder $S'_{ij} = \text{NR}_{\mathcal{O},G}(S_{ij})$ of the S-polynomial of g_i and g_j with respect to G. We say G is an **ε-approximate $\mathcal{O}$-border basis** of the ideal I if $\|S'_{ij}\| \leq \varepsilon$ for all such pairs (i, j), where $\|S'_{ij}\|$ is the Euclidean norm of the coefficient vector of S'_{ij}. If the actual value of ε is of no particular interest we may omit it and refer to G as an **approximate $\mathcal{O}$-border basis**.

Definition 4.1.5. Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and $\varepsilon \geq 0$. Let $A = U\Sigma V^*$ be a SVD (compare Definition 2.5.31) of A and let $s_1, ..., s_{\min(m,n)}$ be the associated singular values. If $\varepsilon < s_1$ we let $1 \leq i \leq \min(m,n)$ be the index such that $s_{\max(1,i-1)} > \varepsilon \geq s_i$ otherwise we let $i = 1$. We form the matrix $\tilde{A} = U\tilde{\Sigma}V^*$ by setting $s_i = s_{i+1} = ... = s_{\min(m,n)} = 0$ in Σ. The vector subspace $\ker(\tilde{A})$ is called the **ε-approximate kernel** of A and denoted by $\mathrm{apker}(A, \varepsilon)$.

Proposition 4.1.6. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, let $A = U\Sigma V^*$ be a SVD of A, let $\varepsilon \geq 0$ and let i as in Definition 4.1.5. The last $n - i + 1$ columns $v_i, ..., v_n$ of the matrix V form an orthonormal basis of $\mathrm{apker}(A, \varepsilon)$. For all v_k, with $i \leq k \leq n$, the inequality $\|Av_k\| \leq \varepsilon$ holds.*

Proof. See [6, Section 2.5.5]. □

4.2 The AVI Algorithm

In [28] Heldt et al. have proposed an algorithm for the computation of an **A**pproximate **V**anishing **I**deal (AVI) for a finite set of real points $\mathbb{X}$. The algorithm differs from the Buchberger-Möller algorithm for border bases (3.4) primarily in the aspect that polynomial relations among the coordinates of the points are not found via the exact computation of the kernel of the evaluation matrices but via the computation of the ε-approximate kernel. For this purpose the singular value decomposition (compare Definitions 2.5.31 and 4.1.5) of the evaluation matrices is used. Another important ingredient is a numerically reasonably stable algorithm for the computation of the reduced row echelon form of a matrix. This is the approximate analogue to line 6 in Algorithm 18. It is necessary in both algorithms to ensure that each polynomial that is added to the (approximate) border basis has a unique border term. We will first of all recall the algorithm which computes the stabilised reduced row echelon form and then present the AVI algorithm. Afterwards we will highlight its properties. The AVI algorithm can be easily generalised to $K = \mathbb{C}$ but we will only give the version presented in [28] for $K = \mathbb{R}$. Hence, let us assume in this section without further notice that $K = \mathbb{R}$.

The following algorithm is essentially a modified version of QR decomposition via Gram-Schmidt orthogonalisation. As pointed out in [28], its numerical stability could be further improved if the modified Gram-Schmidt algorithm, or Givens or Householder transformations would be used instead of the numerically less favourable Gram-Schmidt orthogonalisation. The algorithm computes a matrix $R \in \mathrm{Mat}_{\min(m,n),n}(\mathbb{R})$ which is "almost" in reduced row echelon form. I.e. it differs from a standard reduced row echelon form in that the pivot elements of R are not one and zero rows may appear in between. Please note, that Lemma 3.2 in [28] is inaccurate in that respect as it states that R is in reduced row echelon form.

Additionally let us fix some notations. By $\mathrm{cols}(\cdot)$ and $\mathrm{rows}(\cdot)$ we denote the number of columns and rows of a matrix (compare also Section 8.2), by $\|\cdot\|$ we denote the Euclidean norm, and by $\langle \cdot, \cdot \rangle$ we denote the standard scalar product.

Algorithm 20: Stabilised reduced row echelon form

Input: A matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{R})$, and $\tau > 0$

Output: A matrix $R \in \mathrm{Mat}_{\min(m,n),n}(\mathbb{R})$ almost in reduced row echelon form

$\lambda_1 := \|A_{1:m,1}\|$;
if $\lambda_1 < \tau$ **then**
  $Q := (q_1) := (0, ..., 0)^{\mathrm{tr}} \in \mathrm{Mat}_{m,1}(\mathbb{R})$; $R := (0, ..., 0)^{\mathrm{tr}} \in \mathrm{Mat}_{\min(m,n),1}(\mathbb{R})$;
else
  $Q := (q_1) := \frac{1}{\lambda_1} A_{1:m,1} \in \mathrm{Mat}_{m,1}(\mathbb{R})$; $R := (\lambda_1, 0, ..., 0)^{\mathrm{tr}} \in \mathrm{Mat}_{\min(m,n),1}(\mathbb{R})$;
end
for *i := 1* **to** n **do**
  if $\mathrm{cols}(Q) \leq m$ **then**
    $q_i := A_{1:m,i} - \sum_{j=1}^{\mathrm{cols}(Q)} \langle A_{1:m,i}, q_j \rangle q_j$; $\lambda_i := \|q_i\|$;
    if $\lambda_i < \tau$ **then**
      $R := \left(R, \left(\langle A_{1:m,i}, q_1 \rangle, ..., \left\langle A_{1:m,i}, q_{\mathrm{cols}(Q)} \right\rangle, 0, ..., 0\right)^{\mathrm{tr}}\right)$;
      if $n - i < m - \mathrm{cols}(Q)$ **then** $Q := \left(Q, (0, ..., 0)^{\mathrm{tr}}\right)$;
    else
      $Q := \left(Q, \frac{1}{\lambda_i} q_i\right)$; $R := \left(R, \lambda_i \left(\langle A_{1:m,i}, q_1 \rangle, ..., \langle A_{1:m,i}, q_{i-1} \rangle, 1, 0, ..., 0\right)^{\mathrm{tr}}\right)$;
    end
  else $R := \left(R, \left(\langle A_{1:m,i}, q_1 \rangle, ..., \langle A_{1:m,i}, q_m \rangle\right)^{\mathrm{tr}}\right)$;
end
Starting with the last row and working upwards, use the first non-zero entry of each row of R to clean out the non-zero entries above it;
for *i:=1* **to** $\min(m, n)$ **do**
  $\varrho_i := \|R_{i,1:n}\|$;
  if $\varrho_i < \tau$ **then** $R_{i,1:n} := (0, ..., 0)$;
  else $R_{i,1:n} := \frac{1}{\varrho_i} R_{i,1:n}$;
end
return R;

For further details about the algorithm and some error bounds compare [28, Lemma 3.2].

Remark 4.2.1. The version given here differs in details from the one presented in [28]. It fixes a few border cases, mostly concerned with the dimensions of the matrices, which were not treated in the original version of the algorithm.

For the following algorithm let $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in [-1, 1]^n \subset \mathbb{R}^n$ be a (finite) tuple of real affine points, let $P = \mathbb{R}[x_1, ..., x_n]$ be the polynomial ring in n indeterminates over the real numbers, and let $\mathrm{eval}_{\mathbb{X}} : P \to \mathbb{R}^s$ be the evaluation map associated with $\mathbb{X}$.

Algorithm 21: AVI Algorithm

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in [-1, 1]^n \subset \mathbb{R}^n$, small numbers $\varepsilon > \tau > 0$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}$-border basis G (see Theorem 4.2.2 for details)

$d := 1$, $\mathcal{O} := [1]$, $G := [\,]$, $M := (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{R})$;
$L := [t_1, ..., t_\ell] :=$ all terms of degree 1 ordered decreasingly w.r.t. σ;
repeat
  $linDepInKer := false$;
  $A := (\mathrm{eval}_{\mathbb{X}}(t_1), ..., \mathrm{eval}_{\mathbb{X}}(t_\ell), M)$;
  $B :=$ appKer(A, ε) such that the rows of B form an orthonormal basis of the approximate kernel of A;
  $B' :=$ stableRREF(B, τ) via Algorithm 20, with all zero rows removed;
  // We assume that $\mathcal{O} = [o_1, ..., o_{\mathrm{len}(\mathcal{O})}]$
  while rows$(B') > 0$ **do**
    if *linDepInKer* **then**
      $\tilde{G} := [\tilde{g}_1, ..., \tilde{g}_k]^{\mathrm{tr}} = B' \left(o_1, ..., o_{\mathrm{len}(\mathcal{O})}\right)^{\mathrm{tr}}$;
      foreach $g_i \in G$ **do**
        foreach $\tilde{g}_j \in \tilde{G}$ **do**
          Replace $\mathrm{LT}_\sigma(\tilde{g}_j)$ in g_i by $\mathrm{LT}_\sigma(\tilde{g}_j) - \tilde{g}_j$;
          remove$(\mathcal{O}, t_j)$; remove$(M, \mathrm{eval}(t_j))$;
        end
      end
    else
      $\tilde{G} := [\tilde{g}_1, ..., \tilde{g}_k]^{\mathrm{tr}} = B' \left(t_1, ..., t_\ell, o_1, ..., o_{\mathrm{len}(\mathcal{O})}\right)^{\mathrm{tr}}$;
    end
    $G :=$ concat$(G, \tilde{G}^{\mathrm{tr}})$;
    for $i := \ell$ **downto** *1* **do**
      if $t_i \neq \mathrm{LT}_\sigma(\tilde{g}_1)$ **and** ... **and** $t_i \neq \mathrm{LT}_\sigma(\tilde{g}_k)$ **then**
        $\mathcal{O} :=$ concat$([t_i], \mathcal{O})$; $M :=$ (eval$(t_i), M$);
      end
    end
    $B :=$ appKer(M, ε) such that the rows of B form an orthonormal basis of the approximate kernel of M;
    $B' :=$ stableRREF(B, τ) via Algorithm 20, with all zero rows removed;
    if rows$(B') > 0$ **then** $linDepInKer := true$;
  end
  $d := d + 1$;
  $L := [t_1, ..., t_\ell] :=$ all terms of degree d in $\partial\mathcal{O}$ ordered decreasingly w.r.t. σ;
until $L = [\,]$;
return $(G, \mathcal{O})$;

Theorem 4.2.2. *This is an algorithm which computes a pair $(G, \mathcal{O})$ of sets, $G = \{g_1, ..., g_\nu\}$ and $\mathcal{O} = \{t_1, ..., t_\mu\}$. These have the following properties:*

1. *The set G consists of unitary polynomials which generate a δ-approximate vanishing ideal of $\mathbb{X}$, where $\delta = \varepsilon\sqrt{\nu} + \tau\nu(\mu + \nu)\sqrt{s}$.*
2. *There is no unitary polynomial in $\langle\mathcal{O}\rangle_{\mathbb{R}}$ which vanishes ε-approximately on $\mathbb{X}$.*
3. *If $\mathcal{O}$ is an order ideal of terms, then the set $\tilde{G} = \{(1/\mathrm{LC}_\sigma(g))\, g \mid g \in G\}$ is an $\mathcal{O}$-border prebasis.*
4. *If $\mathcal{O}$ is an order ideal of terms, then the set $\tilde{G}$ is a η-approximate border basis for $\eta = 2\delta + 2\nu\delta^2/\gamma\varepsilon + 2\nu\delta\sqrt{s}/\varepsilon$. Here γ denotes the smallest absolute value of one of the border term coefficients of g_i, which means that $\gamma = \min_{1\leq i\leq\nu} |\mathrm{LC}_\sigma(g_i)|$.*

Proof. Proofs for all claims can be found in [28, Theorem 3.3]. □

Remark 4.2.3. Note that the version of the AVI algorithm given here differs slightly from the one presented in [28]. The additional steps from line 9 to 16 are necessary to make sure that for all polynomials g_i in G it is guaranteed that $|\mathrm{supp}(g_i) \cap \partial\mathcal{O}| = 1$, which means that each g_i must only contain one border term of $\mathcal{O}$. The condition could be violated in the original version, because whenever in line 28 the condition is detected that the ε-approximate kernel of M is not empty, we will get some new polynomials which have border terms which were already in the set $\mathcal{O}$. That is the reason why G, $\mathcal{O}$, and M have to be updated.

Remark 4.2.4. A subtle difference between the Buchberger-Möller algorithm for border bases and the AVI algorithm and all following versions is that in the exact case we require the input to be a set of input points where as in the approximate case we do not require the input to be duplicate free. The reasoning is simple: In case we compute the exact kernel of a matrix adding an exactly identical row will not alter the kernel. Thus it makes sense to assume that we are dealing with a set of points to avoid unnecessary computational steps which have no influence on the result. The situation is, however, different when we are computing the approximate kernel of a matrix. The repeated occurrence of a point does influence the result as we will see in Example 4.3.9, where we will also elaborate on the practical consequences.

The order of the points inside $\mathbb{X}$ does not play a role, so any permutation will produce the same result.

4.2.1 Runtime analysis of the AVI algorithm

As the BM algorithm for border bases (18) and the AVI algorithm share the same basic structure, similar costs for most steps arise, compare Subsection 3.4.1. The main differences are the computation of a SVD in lines 6 and 26 versus the computation of the kernel and the computation of a stabilised RREF (see Algorithm 20) in lines 7 and 27 versus the computation of the usual RREF.

To keep our analysis simple and comparable to Algorithm 18 we choose ε and τ such that they are close to $\varepsilon_{machine}$, which means that except for rounding errors and some subtle differences

the AVI algorithm will degenerate to the BM algorithm for border bases. For larger values of ε and τ an improved runtime can be expected, because $\mathcal{O}$ and G will contain less elements and thus less iterations are needed. The cost measures for the computation of a SVD have been taken from Golub and van Loan (see [6, Section 5.4.5]). According to them the cost of a state of the art implementation is roughly in $O\left(4mn^2 + 8n^3\right)$ if only V and Σ are computed. Please note that we do not need the matrix U for our purposes, which saves us a significant amount of time. If U, V, and $\sum$ are computed the cost is in $O\left(4m^2n + 8mn^2 + 9n^3\right)$. If we recall from Proposition 2.4.4 that the cost for computing the kernel of a matrix via Gauss-Jordan elimination is essentially in $O\left(\min(m,n)^2 \max(m,n)\right)$, then we can conclude that the computation of a SVD is roughly 4 times as expensive. However, they share the same asymptotic complexity.

Now we will investigate the stabilised RREF more closely. Without going into too much detail we note that the core of the stabilised RREF is the computation of a somewhat modified Gram-Schmidt QR decomposition of the input matrix. The result is an upper triangular matrix which needs to be transformed into RREF. According to [5, Theorem 8.1] the cost is in $O\left(2mn^2\right)$ for computing the QR decomposition via Gram-Schmidt orthogonalisation. The cost to transform an upper triangular matrix into RREF is in $O\left(\frac{1}{6}n^3\right)$ if $m \geq n$ and in $O\left(\frac{1}{2}m^2n - \frac{1}{3}m^3\right) = O\left(\frac{1}{2}m^2n\right)$ otherwise (compare 2.4.1). So we conclude that the dominating cost factor in the stabilised RREF calculation is $2mn^2$ for arbitrary values of m and n. The cost for computing a traditional RREF is in $O\left(\min(m,n)^2 \max(m,n)\right)$ according to Proposition 2.4.5. In case $m \geq n$ we see that the computation of the stabilised RREF is roughly 2 times as expensive as the computation of the common RREF. When $m < n$, the computation of the stabilised RREF is $O\left(4\frac{n}{m}\right)$ times more expensive. As we have assumed that ε and τ are close to $\varepsilon_{machine}$ the stabilised RREF only has to be computed in the very last round because the (approximate) kernel is empty in all previous steps.

This means that the dominating cost factor in the AVI algorithm is the computation of the approximate kernel. As the runtime for the computation of the former and the exact kernel only differs by a constant factor, the conclusions which we drew in Subsection 3.4.1 for the case that the kernel was recomputed and not updated stay intact. This means that the runtime of the algorithm in total can be bounded by $O\left(s^3\left(2 + \frac{n(\sqrt[n]{s}-1)}{n+1} + n\right)\right)$, where s is the number of input points and n the number of indeterminates of the polynomial ring.

Remark 4.2.5. The runtime of the AVI algorithm can be improved if initially in lines 6 and 26 only the singular values Σ are computed. According to [6, Subsection 5.4.5] the cost is then approximately in $O\left(4mn^2 - \frac{4}{3}n^3\right)$. Only for the singular values which are smaller than ε we compute the corresponding singular vectors in V via inverse iteration (compare Algorithm 10). This will cost us $O\left(n^2\right)$ operations per vector.

4.2.2 Shortcomings of the AVI algorithm

Because the AVI algorithm processes all terms of degree d at a time and because we need to make sure that all polynomials in G have border prebasis shape (e.g. each g_i has to be of the form $g_i = b_i + h_i$ where $b_i \in \partial\mathcal{O}$ and $\operatorname{supp}(h_i) \subseteq \mathcal{O}$), it is necessary in lines 7 and 27 to compute

a (stabilised) RREF of the singular vectors stored in matrix B, which were computed in lines 6 and 26 of the algorithm. This has several consequences:

- As linear combinations of the original singular vectors are taken into account, the coefficients of the resulting border polynomials are no longer optimal in the total least squares sense. If necessary it is possible to compute the optimal coefficients for each polynomial $g_j \in G$ by computing the evaluation matrix A for the terms in the support of g_j with respect to $\mathbb{X}$. Then, for instance, the technique described in Section 2.13 can be used to obtain the optimal coefficients. However, this additional step is costly and should only be used if it is crucial that all polynomials in G have the best possible fit with respect to their support. In the implementation in the ApCoCoA library ([20]) this is an optional post-processing step.
- Another consequence of computing degree by degree is that in line 23 terms may slip into the order ideal, which would violate claim 2. This can happen because the stabilised RREF works from right to left and will check for (almost) linear dependence among the newly introduced columns first. However, it is not checked if there is an almost linear relation between the newly introduced columns and the already existing ones which belong to terms in the order ideal. That is why lines 26 to 28 are necessary in the algorithm to check if additional relations are present. In case such relations were detected the already existing polynomials are rewritten in lines 10 to 16, which again introduces an additional error.
- Given a constant $\kappa > 0$, there is no easy way to compute a set of polynomials which vanishes κ-approximately on $\mathbb{X}$ with the AVI algorithm because there is only a soft connection between the input parameter ε and the actual δ-approximate vanishing of the polynomials (compare Theorem 21 claim 1).
- There is only a weak control over the coefficients of the border terms. It is guaranteed that no coefficient has an absolute value of less than τ. However, in practice τ cannot be chosen a lot greater than 0.001 without loosing too much precision in the computation. If the polynomials shall be used as input for other algorithms which make use of the approximate border basis properties the distance to an exact border basis η (compare Theorem 4.2.2 claim 4) may become unacceptably large. Please note that this is not the average case behaviour and usually τ is considerably smaller than the worst case estimate.
- In some rare situations the set $\mathcal{O}$ may not be an order ideal of terms, meaning that for one term not all divisors are included in the set. If in a subsequent computation it is imperative that the output of the AVI algorithm is a proper approximate border basis, this can be cured by checking during the computation if the order ideal property is satisfied. If not the responsible polynomial will be added into the set G. Of course, this will weaken the bounds concerning δ and η. For further details please see Subsection 4.3.4.

4.2.3 Implementation in ApCoCoA

The computation of the stable RREF and the AVI algorithm were implemented by Daniel Heldt and the author in the ApCoCoA library ([20]). The implementation is built around the LAPACK

software library [17], which provides a state of the art implementation for computing a SVD of a matrix. Additionally, the BLAS software interface was used for operations like matrix-vector and matrix-matrix multiplications to allow for machine dependent optimisations by simply linking to an optimised implementation which makes use of the target computer architecture or additional equipment like graphics cards (e.g. CUDA or OpenCL). A detailed description of the parameters of the ApCoCoA implementation of the AVI algorithm can be found in Appendix 8.1.

4.3 The ABM Algorithm

The Approximate Buchberger-Möller (ABM) algorithm is again a variant of the Buchberger-Möller (BM) algorithm (4.5) for border bases. Therefore, it shares the same basic structure with the AVI algorithm (21). However, there are a few major points which differentiate AVI and ABM and which contribute to the different properties of the ABM algorithm. First of all, the terms are no longer processed degree by degree but one by one. If the approximate kernel would in this case be computed by a standard SVD computation, this would result in a major slowdown. That is why the conventional SVD computation is replaced by a sequence of computations which allows us to update the SVD iteratively. In fact this optimisation leads to a major speed up of the algorithm. One additional benefit of computing term by term is that it is no longer necessary to compute a stabilised RREF (compare Algorithm 20) of the vectors in the approximate kernel. This saves both time and additionally eliminates some of the obstacles which are associated with the computation of such a stabilised RREF (see Subsection 4.2.2).

We start by presenting the main algorithm followed by proving its most important properties. After we have had a look at an example computation, we further analyse the runtime of the ABM algorithm and describe some updating techniques that can be used to speed up the consecutive computations of the SVD. Additionally, we also point out some shortcomings of the ABM algorithm and propose further modifications of the algorithm in Subsections 4.3.3 and 4.3.4 to work around those issues. Finally, we give some details about the C++ implementation of the algorithm in the ApCoCoA library ([20]).

Algorithm 22: ABM Algorithm

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, a small number $\varepsilon \geq 0$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}$-border basis G (see Theorem 4.3.1 for details)

```
1  d := 1, O := [1], G := [], M := (1,...,1)^tr ∈ Mat_{s,1}(C);
2  L := [t_1,...,t_ℓ] := all terms of degree 1 ordered decreasingly w.r.t. σ;
3  repeat
4      for i := 1 to ℓ do
5          A := (eval_X(t_i), M);
6          B := A*A;
7          γ := smallest eigenvalue of B;
8          if √γ ≤ ε then
9              m := |O|;
10             s := (s_{m+1}, s_m, ..., s_1) := the norm one eigenvector of B w.r.t. to γ;
               // We assume that O = [o_m, ..., o_1]
11             g := s_{m+1} t_i + s_m o_m + ... + s_1 o_1;
12             G := concat(G, [g]);
13         else
14             O := concat([t_i], O);
15             M := A;
16         end
17     end
18     d := d + 1;
19     L := [t_1,...,t_ℓ] := all terms of degree d in ∂O ordered decreasingly w.r.t. σ;
20 until L = [];
21 return (G, O);
```

Theorem 4.3.1. *This is an algorithm which computes two sets $G = \{g_1, ..., g_\nu\}$ and $\mathcal{O} = \{t_1, ..., t_\mu\}$ which have the following properties:*

1. *All the polynomials in G are unitary and generate an ε-approximate vanishing ideal of $\mathbb{X}$.*
2. *There is no unitary polynomial in $\langle \mathcal{O} \rangle_{\mathbb{C}}$ which vanishes ε-approximately on $\mathbb{X}$.*
3. *If $\mathcal{O}$ is an order ideal of terms, then the set $\tilde{G} = \{(1/\operatorname{LC}_\sigma(g))\, g \,|\, g \in G\}$ is an $\mathcal{O}$-border prebasis.*
4. *If $\mathcal{O}$ is an order ideal of terms, then the set $\tilde{G}$ is an δ-approximate border basis with $\delta = 2\,\|\mathbb{X}\|_{\max} / \min_i |\gamma_i| + \nu / (\min_i |\gamma_i|)^2$. Here $\|\mathbb{X}\|_{\max}$ denotes the maximal absolute coordinate in $\mathbb{X}$ and $\min_i |\gamma_i|$ the minimal border coefficient of all polynomials in G.*
5. *If $\varepsilon = 0$, then the algorithm produces the same results as the Buchberger-Möller algorithm for border bases (18).*

Proof. First of all, we show that all steps of the algorithm are well defined. By Theorem 2.13.6 we know that in lines 6,7 and 10 we are actually computing a solution of the homogeneous

least squares problem $\min_x \|Ax\|$ subject to $\|x\| = 1$. Line 8 makes sure that $\|Ax\| \leq \varepsilon$ can be satisfied. However, we still need to show that the eigenvector which is computed in line 10 is uniquely determined. This is true because M^*M and A^*A are Hermitian matrices (see Proposition 2.3.24). We know by Proposition 2.9.23 that the eigenvalues of those two matrices interlace and by Proposition 2.13.5 that they only have non-negative eigenvalues. Let $\tilde{\lambda}_m$ be the smallest eigenvalue of the matrix $M^*M \in \mathrm{Mat}_m(\mathbb{C})$ and let $\lambda_m \geq \lambda_{m+1}$ be the two smallest eigenvalues of $A^*A \in \mathrm{Mat}_{m+1}(\mathbb{C})$. We know by construction that $\sqrt{\tilde{\lambda}_m} > \varepsilon$. If we reach line 10 of the algorithm this means additionally that $\sqrt{\lambda_{m+1}} \leq \varepsilon$. So by $\sqrt{\lambda_m} \geq \sqrt{\tilde{\lambda}_m} \geq \sqrt{\lambda_{m+1}}$ we may conclude that $\sqrt{\lambda_m} > \varepsilon$. This means that in our setting it cannot happen that the homogeneous least squares problem has more than one solution.

Next, we prove finiteness. Essentially the same arguments apply as in the case of the BM algorithm for border bases. The only difference is that we proceed term by term. The border of $\mathcal{O}$ in degree d can only contain elements if new elements are added to $\mathcal{O}$ in line 14. However, this can only happen finitely many times as when the number of columns of A is greater than the number of rows, then the (approximate) kernel cannot be trivial and no new elements will be added to $\mathcal{O}$.

Next we show 1. All polynomials are unitary because they are constructed to be norm one eigenvectors in line 10. They vanish ε-approximately on $\mathbb{X}$ because we are using the algorithmic steps given in Algorithm 16 and described in Theorem 2.13.6.

To prove 2 it suffices to point out that, if such a relation existed, it would be found because of Theorem 2.13.6. Then in line 8 the condition $\sqrt{\gamma} \leq \varepsilon$ would be satisfied and the corresponding term would not go into the order ideal in line 14.

Now we move on to prove 3. Again, by construction all polynomials g_i in $\tilde{G}$ are of the form $t_i - \sum_{i=1}^{|\mathcal{O}|} c_i o_i$, where $c_i \in \mathbb{C}$ and $o_i \in \mathcal{O}$. By how L is constructed we know that the t_i are all distinct and that they are exactly the elements in $\partial\mathcal{O}$. So if the set $\mathcal{O}$ formed an order ideal of terms, $\tilde{G}$ will be an $\mathcal{O}$-border prebasis.

It remains to be shown that $\tilde{G}$ forms a δ-approximate border basis where

$$\delta = 2\,\|\mathbb{X}\|_{\max} / \min_i |\gamma_i| + \nu / \left(\min_i |\gamma_i|\right)^2,$$

if the set $\mathcal{O}$ is an order ideal. We know that every $g_i \in G$ is of the form $g_i = \gamma_i b_i + h_i$ with $\gamma_i \in \mathbb{C}\setminus\{0\}$ and $|\gamma_i| \in]0,1]$, $b_i \in \partial\mathcal{O}$, and $\mathrm{supp}(h_i) \subseteq \mathcal{O}$. Then we let $\tilde{g}_i = b_i + h_i/\gamma_i = b_i + \tilde{h}_i$. Thus the polynomials $\tilde{g}_i$ will vanish $\varepsilon/|\gamma_i|$-approximately on $\mathbb{X}$. Additionally, for every coefficient $c_{ij} \in \mathbb{C}\setminus\{0\}$ of the monomials in $\tilde{h}_i$ the inequality $|c_{ij}| \leq 1/|\gamma_i|$ holds, as the coefficients of g_i were those of a unitary polynomial. Let us denote by $c \in \mathbb{C}^\varsigma$ the coefficient vector of an arbitrary polynomial g with $\mathrm{supp}(g) \subseteq \mathcal{O}$, and $\varsigma = |\mathcal{O}| \leq s$. We know that $\mathrm{eval}_{\mathbb{X}}(g) = Mc^{\mathrm{tr}}$ where $M \in \mathrm{Mat}_{s,\mu}(\mathbb{C})$ is the evaluation matrix of the elements in the order ideal $\mathcal{O}$ with respect to $\mathbb{X}$. This allows us to derive a relation between the coefficient vector c and the evaluation of the polynomial g. As M is guaranteed to have full rank μ, we further conclude that $M^+ \mathrm{eval}_{\mathbb{X}}(g) = M^+Mc^{\mathrm{tr}} = c^{\mathrm{tr}}$, where M^+ is the pseudoinverse of M (compare Definition 2.6.1 and Proposition 2.6.5). Note, that if $\mathrm{eval}_{\mathbb{X}}(g) = 0_s$, then c has to be equal to the zero vector as

well because M has full rank. Now we can bound the euclidean norm of the coefficient vector by using

$$\|c\| = \left\|M^+ \operatorname{eval}_{\mathbb{X}}(g)\right\| \leq \left\|M^+\right\| \|\operatorname{eval}_{\mathbb{X}}(g)\|.$$

Let $U\Sigma V^*$ be the singular value decomposition of M. Then the pseudoinverse of M can be computed as $M^+ = V\Sigma^+U^*$ (see Proposition 2.6.2), where Σ^+ is Σ with all diagonal elements inverted. Now we can conclude that

$$\left\|M^+\right\| = \left\|V\Sigma^+U^*\right\| = \left\|\Sigma^+\right\| = \frac{1}{\sigma_\varsigma} = \frac{1}{\sigma_{\min}}.$$

Additionally, we can bound the smallest singular value σ_ς of M by ε which follows from claim 2. So $\|M^+\| < \frac{1}{\varepsilon}$. Thus we arrive at $\|c\| < \frac{1}{\varepsilon}\|\operatorname{eval}(g_i)\|$. This means that once we can bound the evaluation of g_i, we are done.

We will first look at the across-the-street neighbours. Let $S_{ij} = x_k\tilde{g}_i - x_l\tilde{g}_j$ and $S'_{ij} = \operatorname{NR}_{\mathcal{O},\tilde{G}}(S_{ij}) = x_k\tilde{g}_i - x_l\tilde{g}_j - \sum_\nu c_\nu\tilde{g}_\nu$ where the c_ν are some coefficients of the polynomials $\tilde{h}_i$. Note that $\operatorname{supp}(S'_{ij}) \subseteq \mathcal{O}$. In case $\varepsilon = 0$ we know that $\operatorname{eval}_{\mathbb{X}}(S'_{ij}) = 0_s$. As the matrix M has only a trivial kernel we observe that $S'_{ij} = 0$. So let us assume that $\varepsilon > 0$. Now we know that $|c_v| \leq 1/\min_i|\gamma_i|$, where $\min_i|\gamma_i|$ is the minimal absolute value of all coefficients of all border terms in G. We conclude that

$$\begin{aligned}
\left\|S'_{ij}\right\| &\leq \left\|M^+\right\| \left\|\operatorname{eval}_{\mathbb{X}}\left(S'_{ij}\right)\right\| < \frac{1}{\varepsilon}\left\|\operatorname{eval}_{\mathbb{X}}\left(S'_{ij}\right)\right\| \\
&\leq \frac{1}{\varepsilon}\left(\|\operatorname{eval}_{\mathbb{X}}(x_k\tilde{g}_i)\| + \|\operatorname{eval}_{\mathbb{X}}(x_l\tilde{g}_j)\| + \left\|\operatorname{eval}_{\mathbb{X}}\left(\sum_\nu c_\nu\tilde{g}_\nu\right)\right\|\right) \\
&\leq \frac{1}{\varepsilon}\left(\|\|\mathbb{X}\|_{\max}\operatorname{eval}_{\mathbb{X}}(\tilde{g}_i)\| + \|\|\mathbb{X}\|_{\max}\operatorname{eval}_{\mathbb{X}}(\tilde{g}_j)\| + \left\|\operatorname{eval}_{\mathbb{X}}\left(\sum_\nu c_\nu\tilde{g}_\nu\right)\right\|\right) \\
&\leq \frac{1}{\varepsilon}\left(2\|\mathbb{X}\|_{\max}\varepsilon/\min_i|\gamma_i| + \frac{1}{\min_i|\gamma_i|}\sum_\nu \operatorname{eval}_{\mathbb{X}}(\tilde{g}_\nu)\right) \\
&\leq 2\|\mathbb{X}\|_{\max}/\min_i|\gamma_i| + \frac{\nu}{\min_i|\gamma_i|^2}.
\end{aligned}$$

Here $\|\mathbb{X}\|_{\max} = \max_{i,j}|p_{i,j}|$ denotes the maximal absolute value of all coordinates of all points in $\mathbb{X}$.

Finally, we analyse the next-door neighbours. The error bound is derived analogously to the case of the across-the-street neighbours as $S_{ij} = \tilde{g}_i - x_k\tilde{g}_j$ and $S'_{ij} = \operatorname{NR}_{\mathcal{O},\tilde{G}}(S_{ij}) = \tilde{g}_i - x_k\tilde{g}_j - \sum_\nu c_\nu\tilde{g}_\nu$ where the c_ν are again some coefficients of the polynomials $\tilde{h}_i$. In the case $\varepsilon = 0$ we know that $\operatorname{eval}_{\mathbb{X}}\left(S'_{ij}\right) = 0_s$. As the matrix M has only a trivial kernel we observe that $S'_{ij} = 0$. This means that if we let $\varepsilon = 0$ in the algorithm we obtain an exact border basis as both the normal remainder of the S-polynomials of the across-the-street and the next-door neighbours is 0, see

Theorem 3.1.15. Let us now assume that $\varepsilon > 0$. We compute

$$\begin{aligned}
\left\|S'_{ij}\right\| &\leq \|M^+\| \left\|\mathrm{eval}_{\mathbb{X}}\left(S'_{ij}\right)\right\| < \frac{1}{\varepsilon}\left\|\mathrm{eval}_{\mathbb{X}}\left(S'_{ij}\right)\right\| \\
&\leq \frac{1}{\varepsilon}\left(\|\mathrm{eval}_{\mathbb{X}}(\tilde{g}_i)\| + \|\|\mathbb{X}\|_{\max}\,\mathrm{eval}_{\mathbb{X}}(\tilde{g}_j)\| + \left\|\mathrm{eval}_{\mathbb{X}}\left(\sum_{\nu} c_\nu \tilde{g}_\nu\right)\right\|\right) \\
&\leq \ldots \\
&\leq 1/\min_i |\gamma_i| + \|\mathbb{X}\|_{\max}/\min_i |\gamma_i| + \frac{\nu}{(\min_i |\gamma_i|)^2}.
\end{aligned}$$

□

Remark 4.3.2. The result of the algorithm is strongly influenced by the value of the parameter ε. A reasonable choice depends on to the amount of noise which is present in the physical measurements $\mathbb{X}$. For practical purposes it is possible to mostly automate the process of finding a suitable ε value, by taking into account certain domain specific feasibility and optimisation criteria. We will describe this procedure in more detail in Section 6.1.

Remark 4.3.3. It is important to note that the parameter δ in claim 4 of Theorem 4.3.1 is in fact independent of the actual choice of ε. To illustrate this, consider the scenario that the matrix B in line 6 of Algorithm 22 has a smallest eigenvalue γ such that $\sqrt{\gamma} = \varepsilon + \Delta$ with $\Delta \in \mathbb{R}^+$. This means that no new element will be added to the list G and t_i will be added to $\mathcal{O}$. Furthermore, let us assume that in the next iteration of the for-loop, we will obtain a matrix $\tilde{B}$ with smallest eigenvalue $\tilde{\gamma}$ such that $\sqrt{\tilde{\gamma}} = \varepsilon$. Consequently, a new polynomial g with border coefficient s_{m+1} will be formed and added to G in line 12. If we now let $\lim \Delta \to 0$ the coefficient s_{m+1} will also tend to zero. This however means that the border coefficient with the minimal absolute value $\min_i |\gamma_i|$ in the parameter δ can be arbitrarily close to 0 and does not depend on ε. In Chapter 7 we sketch a variant of the ABM algorithm that does not suffer from this problem.

Remark 4.3.4. If we are facing the situation as described in Remark 4.3.3, then a small perturbation of the input parameter ε will most of the time remedy the problem as then either the original matrix B will have the property that $\sqrt{\gamma} \leq \varepsilon$ or both $\sqrt{\gamma}$ and $\sqrt{\tilde{\gamma}}$ will be greater than ε.

As a next step we analyse the accuracy of Algorithm 22.

Definition 4.3.5. Let $\mathbb{X}$ and $\tilde{\mathbb{X}}$ be two (ordered) tuples of s affine points in $\mathbb{C}^n$. If we write, by a slight abuse of notation, $\mathbb{X}$ and $\tilde{\mathbb{X}}$ in matrix form such that $\mathbb{X}, \tilde{\mathbb{X}} \in \mathrm{Mat}_{s,n}(\mathbb{C})$, then we say that $\tilde{\mathbb{X}}$ is a τ**-perturbation** of $\mathbb{X}$ if $\|\mathbb{X} - \tilde{\mathbb{X}}\|_2 = \tau$.

Remark 4.3.6. Let $\mathbb{X}$ and $\tilde{\mathbb{X}}$ be two tuples of s affine points in $\mathbb{C}^n$ such that $\tilde{\mathbb{X}}$ is a τ-perturbation of $\mathbb{X}$, with $\tau \in \mathbb{R}^+$. We now apply Algorithm 22 for a fixed $\varepsilon \in \mathbb{R}^+$ to both $\mathbb{X}$ and $\tilde{\mathbb{X}}$. Please note that even if τ is very small (e.g. in the magnitude of rounding errors) the check in line 8 of the algorithm may return different results for $\mathbb{X}$ and $\tilde{\mathbb{X}}$. If that is the case, the sets $\mathcal{O}$ and $\tilde{\mathcal{O}}$ will differ and the results are no longer directly comparable. We therefore need the additional assumption that $\mathcal{O}$ and $\tilde{\mathcal{O}}$ are identical for both $\mathbb{X}$ and $\tilde{\mathbb{X}}$ in order to be able to say something about the accuracy of the computed result (compare Subsection 2.7.2).

Theorem 4.3.7 (Accuracy of Algorithm 22). *Let $\mathbb{X}$ be a tuple of s affine points in $\mathbb{C}^n$ and let $\varepsilon \in \mathbb{R}^+$. Furthermore, let us denote by G and $\mathcal{O}$ the theoretically exact results computed by the ABM algorithm and let us denote by $\tilde{G}$ and $\tilde{\mathcal{O}}$ the results computed by a floating point implementation of the algorithm. Furthermore, let us assume that $\mathcal{O} = \{t_1, ..., t_\mu\} = \tilde{\mathcal{O}}$ (see Remark 4.3.6), such that $G = \{g_1, ..., g_\nu\}$ and $\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_\nu\}$ have the same structure. Additionally, we denote the coefficient vector of g_i by c_i and the coefficient vector of $\tilde{g}_i$ by $\tilde{c}_i$ for each $1 \leq i \leq \nu$ and we let $\tilde{A}_i = \text{eval}_{\mathbb{X}}(\text{supp}(\tilde{g}_i)) \in \text{Mat}_{s,k}$, with $k \in \{1, ..., \mu+1\}$, be the floating point evaluation matrix of the terms in the support of $\tilde{g}_i$ with respect to $\mathbb{X}$.*
Let θ_i be the angle between the coefficient vectors c_i and $\tilde{c}_i$, i.e. $\cos(\theta_i)\|c_i\|\|\tilde{c}_i\| = \cos(\theta_i) = |\langle c_i, \tilde{c}_i\rangle|$. If $\text{gap}_k\left(\tilde{A}^*\tilde{A}\right) \neq 0$ (see Definition 2.9.20) and if the used computer satisfies Assumption 2.7.5, the following error estimate holds:

$$\frac{1}{2}\sin(\theta_i) \in O\left(\frac{\varepsilon_{machine}\left\|\tilde{A}_i\right\|_2^2}{\text{gap}_k\left(\tilde{A_i}^*\tilde{A}_i\right)}\right).$$

Proof. First of all we let $A_i = \text{eval}_{\mathbb{X}}(\text{supp}(\tilde{g}_i))$ be the exact result of evaluating the elements in the support of $\tilde{g}_i$ on $\mathbb{X}$ and we let $\tilde{A}_i$ (as defined above) be the floating point counterpart. Then we can define $E_i = A_i - \tilde{A}_i \in \text{Mat}_{s,\mu}(\mathbb{C})$ as the error matrix. Please note that because we assume that the used computer has backward stable implementations of the basic floating point operations we can conclude that $\frac{\|E_i\|_2}{\|A_i\|_2} \in O(\varepsilon_{machine})$. This means that we can directly apply Theorem 2.13.8, because the homogeneous least squares problem in Algorithm 22 is solved via Algorithm 16. □

We will now illustrate the workings of the algorithm on a small example in which the parameter ε is chosen in such a way that the order ideal contains four elements after the algorithm terminates.

Example 4.3.8. Let $P = \mathbb{R}[x_1, x_2]$, $\mathbb{X} = [(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)]$, and $\varepsilon = 0.2$ be given and let σ be the `DegRevLex` term ordering.

- $d = 1, \mathcal{O} = [1]$, $G = [\,]$, $M = (1, ..., 1)^{\text{tr}}$ and $L = [x_1, x_2]$
- $A = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0.5 & 1 \end{pmatrix}$, solve $\min_x \|Ax\|$ with $\|x\| = 1$, $A^*A = \begin{pmatrix} 2.25 & 2.5 \\ 2.5 & 5 \end{pmatrix}$, $\sqrt{e} \approx 0.878$, $\mathcal{O} = [x_2, 1]$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 2.25 & 1.25 & 2.5 \\ 1.25 & 2.25 & 2.5 \\ 2.5 & 2.5 & 5 \end{pmatrix}$, $\sqrt{e} \approx 0.796, \mathcal{O} = [x_2, x_1, 1]$
- $d = 2$, $\partial\mathcal{O} = \{x_1^2, x_1x_2, x_2^2\}$, and $L = [x_1^2, x_1x_2, x_2^2]$

- $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 2.0625 & 1.125 & 2.125 & 2.25 \\ 1.125 & 2.25 & 1.25 & 2.5 \\ 2.125 & 1.25 & 2.25 & 2.5 \\ 2.25 & 2.5 & 2.5 & 5 \end{pmatrix}$,
$\sqrt{e} \approx 0.154$, $s \approx (-0.697, 0, 0.715, -0.044)$
- $(-0.697, 0, 0.715, -0.044)\left(x_1^2, x_2, x_1, 1\right)^{\text{tr}} = -0.697x_1^2 + 0.715x_1 - 0.044 = g_1$,
$G = [g_1]$, $\mathcal{O} = [x_2, x_1, 1]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 1.0625 & 1.125 & 1.125 & 1.25 \\ 1.125 & 2.25 & 1.25 & 2.5 \\ 1.125 & 1.25 & 2.25 & 2.5 \\ 1.25 & 2.5 & 2.5 & 5 \end{pmatrix}$,
$\sqrt{e} \approx 0.380$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.25 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 2.0625 & 1.0625 & 2.125 & 1.125 & 2.25 \\ 1.0625 & 1.0625 & 1.125 & 1.125 & 1.25 \\ 2.125 & 1.125 & 2.25 & 1.25 & 2.5 \\ 1.125 & 1.125 & 1.25 & 2.25 & 2.5 \\ 2.25 & 1.25 & 2.5 & 2.5 & 5 \end{pmatrix}$,
$\sqrt{e} \approx 0.154$, $s \approx (0.685, 0.041, -0.724, -0.021, 0.054)$
- $(0.685, 0.041, -0.724, -0.021, 0.054)\left(x_2^2, x_1x_2, x_2, x_1, 1\right)^{\text{tr}} = 0.685x_2^2 + 0.041x_1x_2 - 0.724x_2 - 0.021x_1 + 0.054 = g_2$, $G = [g_1, g_2]$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$
- $d = 3$, $\partial\mathcal{O} = \left\{x_1^2x_2, x_1x_2^2, x_1^2, x_2^2\right\}$, and $L = \left[x_1^2x_2, x_1x_2^2\right]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.125 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$,
$A^*A = \begin{pmatrix} 1.015625 & 1.03125 & 1.0625 & 1.0625 & 1.125 \\ 1.03125 & 1.0625 & 1.125 & 1.125 & 1.25 \\ 1.0625 & 1.125 & 2.25 & 1.25 & 2.5 \\ 1.0625 & 1.125 & 1.25 & 2.25 & 2.5 \\ 1.125 & 1.25 & 2.5 & 2.5 & 5 \end{pmatrix}$, $\sqrt{e} = 0.077$,
$s \approx (-0.698, 0.715, -0.008, -0.008, -0.013)$
- $(-0.698, 0.715, -0.008, -0.008, -0.013)\left(x_1^2x_2, x_1x_2, x_2, x_1, 1\right)^{\text{tr}} = -0.698x_1^2x_2 + 0.715x_1x_2 - 0.008x_2 - 0.008x_1 - 0.013 = g_3$,
$G = [g_1, g_2, g_3]$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$

- $A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.125 & 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$,
  $A^*A = \begin{pmatrix} 1.015625 & 1.03125 & 1.0625 & 1.0625 & 1.125 \\ 1.03125 & 1.0625 & 1.125 & 1.125 & 1.25 \\ 1.0625 & 1.125 & 2.25 & 1.25 & 2.5 \\ 1.0625 & 1.125 & 1.25 & 2.25 & 2.5 \\ 1.125 & 1.25 & 2.5 & 2.5 & 5 \end{pmatrix}$, $\sqrt{e} = 0.077$,
  $s \approx (-0.698, 0.715, -0.008, -0.008, -0.013)$
- $(-0.698, 0.715, -0.008, -0.008, -0.013)\left(x_1x_2^2, x_1x_2, x_2, x_1, 1\right)^{\mathrm{tr}} =$
  $-0.698x_1x_2^2 + 0.715x_1x_2 - 0.008x_2 - 0.008x_1 - 0.013 = g_4$, $G = [g_1, g_2, g_3, g_4]$,
  $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$
- $d = 4$, $\partial\mathcal{O} = \left\{x_1^2x_2, x_1x_2^2, x_1^2, x_2^2\right\}$, and $L = \emptyset$
- $G = \{g_1, g_2, g_3, g_4\}$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$

We observe that $\|\mathrm{eval}_{\mathbb{X}}(g_1)\| = 0.156$, $\|\mathrm{eval}_{\mathbb{X}}(g_2)\| = 0.155$, $\|\mathrm{eval}_{\mathbb{X}}(g_3)\| = 0.078$, and $\|\mathrm{eval}_{\mathbb{X}}(g_4)\| = 0.078$, which is in line with claim 1 of Theorem 4.3.1. As a rule of thumb the polynomials g_i which are constructed at a later stage of the computation process vanish better than those obtained earlier in the process. Finally, we compute the normal remainder of the S-polynomials of (g_1, g_3), (g_2, g_4), and (g_3, g_4) whose border terms are neighbours in $\mathcal{O}$:

$$\begin{aligned}
\left\|\mathrm{NR}_{\mathcal{O},\tilde{G}}(S_{13})\right\| &\approx \left\|x_1^2x_2 - 1.0258x_1x_2 + 0.0631x_2\right. \\
&\quad \left. - \left(x_1^2x_2 - 1.0244x_1x_2 + 0.0115x_2 + 0.0115x_1 + 0.0186\right)\right\| \\
&\approx \|-0.0014x_1x_2 + 0.0516x_2 - 0.0115x_1 - 0.0186\| \approx 0.056, \\
\left\|\mathrm{NR}_{\mathcal{O},\tilde{G}}(S_{24})\right\| &\approx \ldots \approx 0.050, \\
\left\|\mathrm{NR}_{\mathcal{O},\tilde{G}}(S_{34})\right\| &\approx \ldots \approx 0.044.
\end{aligned}$$

So, $\tilde{G}$ is a 0.056-approximate $\mathcal{O}$-border basis.

Additionally, we have a look at an example that demonstrates the effect of multiple (almost) identical points, which will give us an idea why the algorithm is relatively robust against outliers in the measured input data.

Example 4.3.9. Let $P = \mathbb{R}[x_1, x_2]$, $\mathbb{X}_1 = [(0,1), (0.9, 2.1), (2,3)]$, and let σ be the `DegRevLex` term ordering. If we apply the Buchberger-Möller algorithm for border bases (18) we obtain $\mathcal{O} = \{1, x_1, x_2\}$ and $G = \{g_1, g_2, g_3\}$ with

$$\begin{aligned}
g_1 &= x_1^2 - 6.95x_1 + 4.95x_2 - 4.95, \\
g_2 &= x_1x_2 - 7.05x_1 + 4.05x_2 - 4.05, \\
g_3 &= x_2^2 - 4.95x_1 + 0.95x_2 - 1.95.
\end{aligned}$$

Now duplicating the first point such that $\mathbb{X}_2 = [(0,1),(0,1),(0.9,2.1),(2,3)]$ does not alter the result, as one would expect. However, if we run the ABM algorithm with $\varepsilon = 0.2$ on $\mathbb{X}_1$ we obtain the sets $\mathcal{O} = \{1, x_2\}$ and $G = \{g_1, g_2, g_3\}$ with

$$\begin{aligned} g_1 &= x_1 - 1.006x_2 + 1.083, \\ g_2 &= x_2^2 - 4.094x_2 + 3.548, \\ g_3 &= x_1x_2 - 3.264x_2 + 4.082. \end{aligned}$$

If we run the algorithm again with $\varepsilon = 0.2$ and input data $\mathbb{X}_2$ we obtain the sets $\mathcal{O} = \{1, x_2\}$ and $G = \{g_1, g_2, g_3\}$ with

$$\begin{aligned} g_1 &= x_1 - 0.986x_2 + 1.028, \\ g_2 &= x_2^2 - 3.977x_2 + 3.228, \\ g_3 &= x_1x_2 - 3.066x_2 + 3.531. \end{aligned}$$

We can observe that the coefficients of the equations have slightly changed. The reason is that during one run of Algorithm 22 several homogeneous least squares problem are solved. As the point $(0,1)$ shows up twice the coefficients of the polynomials change to minimise the total residual error. In a real world situation we would expect that points which are representative of a physical system show up more often than outliers. As the total residual is minimised, the algorithm is quite robust with respect to measurement noise and outliers. Figure 4.1 also illustrates that the duplication of point A moves the zero set of all polynomials in the direction of A.

4.3.1 Runtime Complexity of the ABM Algorithm

In this subsection we analyse the runtime complexity of the ABM algorithm more closely. Just like in the case of the Buchberger-Möller algorithm for border bases 3.4, a significant amount of time can be saved if the matrix decomposition which reveals the approximate kernel of the matrix in question is updated in each step (compare Remark 3.4.5). We first discuss how this updating process can be performed and afterwards detail the runtime of the ABM algorithm.

Remark 4.3.10. [Efficiently Updating the SVD]

Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let $\tilde{A} \in \mathrm{Mat}_{m,n+1}(\mathbb{C})$ be the matrix which we obtain by prepending an additional column c to A as a new first column. A significant speed up can be achieved by updating $\tilde{A}^*\tilde{A}$ and its eigendecomposition instead of completely recomputing it.

Let us first consider the computation of the matrix-matrix product $\tilde{A}^*\tilde{A}$. Assuming that we have computed the matrix product A^*A in a previous step, we only have to add one row and one column, which are in fact complex conjugates of each other, as $\tilde{A}^*\tilde{A}$ is Hermitian (compare Proposition 2.3.24), to this result. So the cost per update is in $O(m(n+1))$ which is a distinct improvement over $O(\frac{1}{2}mn^2)$ (if no advanced techniques like the Schönhage-Strassen or Coppersmith-Winograd algorithm are used for the matrix multiplication, see [8, Section 28.2]).

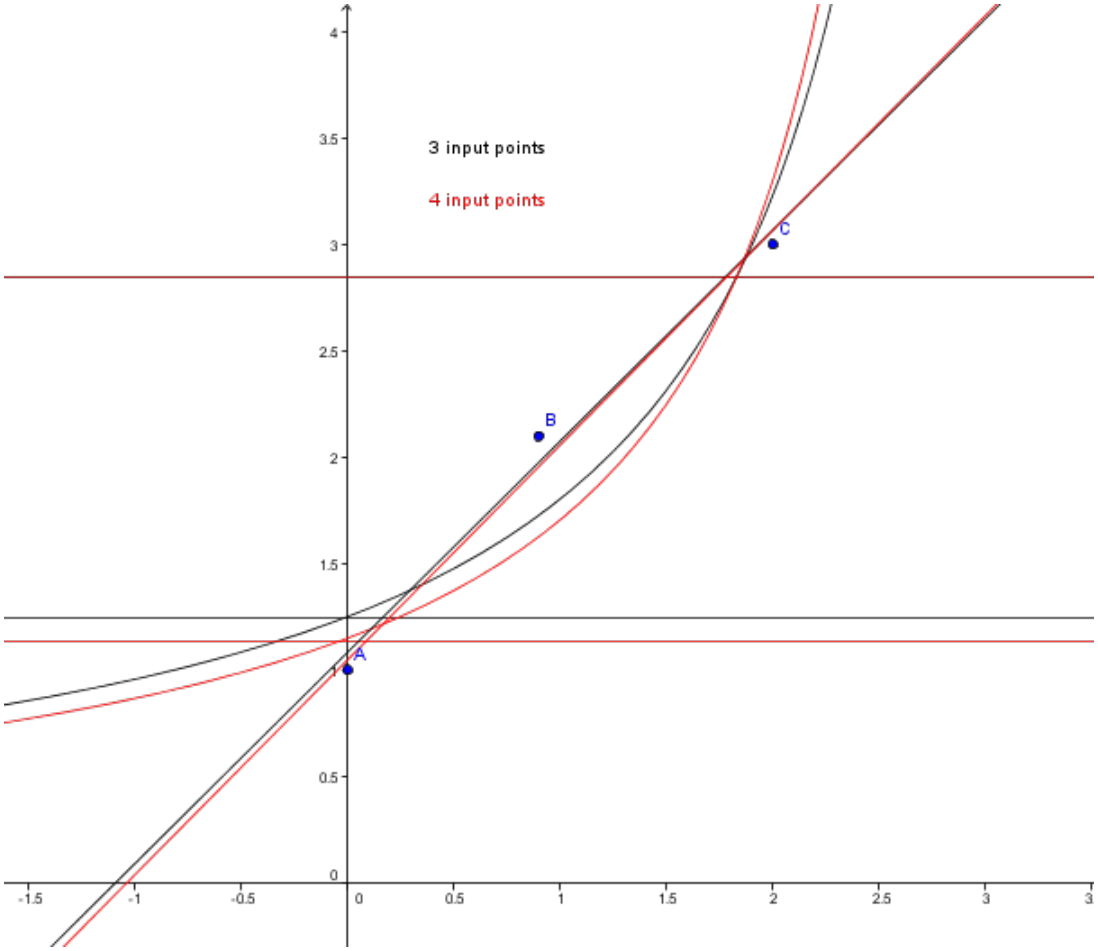

Figure 4.1: The effect of several identical input points on the output of the ABM algorithm.

This process can be visualised in the following way:

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{A^*} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{A^*A}$$

$$\underbrace{\begin{bmatrix} \bar{c}_1 & \bar{c}_2 & \cdots & \bar{c}_{m-1} & \bar{c}_m \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{\tilde{A}^*} \underbrace{\begin{bmatrix} c_1 & \times & \times & \times & \times \\ c_2 & \times & \times & \times & \times \\ \vdots & \times & \times & \times & \times \\ c_{m-1} & \times & \times & \times & \times \\ c_m & \times & \times & \times & \times \end{bmatrix}}_{\tilde{A}} = \underbrace{\begin{bmatrix} r_1 & r_2 & \cdots & r_{n+1} \\ \bar{r}_2 & & & \\ \vdots & & A^*A & \\ \bar{r}_{n+1} & & & \end{bmatrix}}_{\tilde{A}^*\tilde{A}}.$$

In real world examples, especially in the context of the oil industry in which the ABM algorithm has been successfully applied, we may assume that $m \gg n$, which means that the number of input points by far exceeds the number of elements in the order ideal $\mathcal{O}$. This means that the

most significant cost factor is the matrix-matrix multiplication. As the size of the order ideal usually does not exceed 50 elements, the cost for computing the eigendecomposition of $\tilde{A}^*\tilde{A}$ is small enough such that no further optimisations are required to achieve a runtime which is satisfactory from a practical point of view.

However, it is also possible to update the eigendecomposition in an economic way. Let us briefly describe the individual steps of this process.
First we assume that we have computed the matrices Σ and V of a SVD (compare Definition 2.5.31) of A.

1. Form the matrix $\tilde{V} \in \mathrm{Mat}_{n+1}(\mathbb{C})$ such that the first row and the first column contain the unit vectors $(1,0,...,0)$ and $(1,0,...,0)^{\mathrm{tr}}$. All other entries are copied over from matrix V. Please note that V is unitary and so is $\tilde{V}$. This means that $\tilde{V}$ can be inverted without any computation, as we can see by
$$\tilde{V}^*\tilde{V} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & V^* & \\ 0 & & & \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & V & \\ 0 & & & \end{bmatrix} = I_{n+1}.$$

2. Compute $\tilde{V}^*\tilde{A}^*\tilde{A}\tilde{V}$. To achieve this, we do not have to perform two full matrix multiplications as
$$\tilde{V}^*\tilde{A}^*\tilde{A}\tilde{V} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n+1} \\ \bar{\alpha}_2 & & & \\ \vdots & & \Sigma^2 & \\ \bar{\alpha}_{n+1} & & & \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n+1} \\ \bar{\alpha}_2 & \sigma_1^2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \bar{\alpha}_{n+1} & 0 & 0 & \sigma_n^2 \end{bmatrix} = \bar{\Sigma}.$$
In fact, we only have to perform two matrix-vector multiplications in $O(2(n+1)^2)$ to get the entries α_i of the new matrix. So the matrix $\bar{\Sigma}$ is sparse, Hermitian and contains only $3n+1$ entries. By using its symmetry we can store $\bar{\Sigma}$ utilising only $2n+1$ entries.

3. Now we have to transform $\bar{\Sigma}$ into tridiagonal form. One possibility is to use the Lanczos algorithm 13. As the matrix $\bar{\Sigma}$ is sparse, the cost for this step is in $O((n+1)^2)$. Please note that if techniques like Householder reflections (compare Algorithm 12) were used, the runtime would be in $O(\frac{4}{3}(n+1)^3)$. Alternatively, it is also possible to use Givens rotations to transform $\bar{\Sigma}$ into tridiagonal form in an economic way. Let us denote the resulting tridiagonal matrix by T with $T = Q^*\bar{\Sigma}Q$.

4. Compute the eigenvalues of T, preferably with the QR algorithm. The cost is then in $O((n+1)^2)$, as T is tridiagonal. Using inverse iteration (see Algorithm 10) we can compute the eigenvectors $l_1,...,l_{n+1} \in \mathrm{Mat}_{n+1,1}(\mathbb{C})$ of this tridiagonal matrix in $O((n+1)^2)$ as well. Let $(l_1,...,l_{n+1}) = L \in \mathrm{Mat}_{n+1}(\mathbb{C})$. Please note that these eigenvectors are not identical to those of $\tilde{A}^*\tilde{A}$. To obtain those one could compute $\tilde{V}QL$ which is in $O(2(n+1)^3)$ because all matrices involved are dense. Fortunately though, it is not necessary to perform these calculations explicitly - at least not in every step (we will elaborate on this in more detail

later in Remark 4.3.11). In our setting we are only interested in the eigenvector associated to the smallest eigenvalue σ_{n+1} if it is smaller than ε. So only in this case we compute the eigenvector $\tilde{v}_{n+1}$ associated to σ_{n+1} of $\tilde{A}^*\tilde{A}$. A reasonably fast solution is to form $\tilde{v}_{n+1} = \tilde{V}Ql_{n+1}$, which will only cost us $O(2(n+1)^2)$ operations.

5. Basically, we are now done with one step of updating the matrix factorisation. In order to update the matrix factorisation in the future again we need to be careful though as we had assumed in the beginning of the updating process that we had both Σ and V explicitly given. The situation is now changed in the way that we only have the singular values Σ directly available. We can only access the singular vectors V in an implicit way. Forming them explicitly would require a series of matrix multiplications. This is fortunately not necessary as we will see. Let us denote by $\bar{A} \in \mathrm{Mat}_{m,n+2}(\mathbb{C})$ the matrix which we obtain by prepending an additional column c as a new first column to $\tilde{A}$. Let

$$\bar{V} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \tilde{V}QL & \\ 0 & & & \end{bmatrix} \quad \text{and} \quad \bar{V}^* = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & L^*Q^*\tilde{V}^* & \\ 0 & & & \end{bmatrix}.$$

Then we know that

$$\bar{V}^*\tilde{A}^*\tilde{A}\bar{V} = \bar{V}^* \begin{bmatrix} r_1 & r_2 & \cdots & r_{n+1} \\ \bar{r}_2 & & & \\ \vdots & & \tilde{A}^*\tilde{A} & \\ \bar{r}_{n+1} & & & \end{bmatrix} \bar{V} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n+2} \\ \bar{\alpha}_2 & \tilde{\sigma}_1^2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \bar{\alpha}_{n+2} & 0 & 0 & \tilde{\sigma}_{n+1}^2 \end{bmatrix}.$$

But as the singular values $\tilde{\sigma}_1, ..., \tilde{\sigma}_{n+1}$ are known we only have to compute $\alpha_1, ..., \alpha_{n+2}$ which can be obtained by forming $\bar{V}^*(r_1, ..., \bar{r}_{n+1})^{\mathrm{tr}}\bar{V}$ explicitly. The cost for this operation will only be in $O(6(n+1)^2)$. One should keep in mind though that with every subsequent update six additional matrix-vector multiplications will be necessary.

What remains to be done is to sum up the runtime of all individual steps. We thus arrive at a total runtime of

$$\begin{aligned} & O\left(2(n+1)^2 + (n+1)^2 + (n+1)^2 + 2(n+1)^2 + 6(n+1)^2\right) \\ = \; & O\left(12(n+1)^2\right) = O\left(12n^2\right). \end{aligned}$$

Remark 4.3.11. It should be noted that updating a SVD several times will introduce (additional) numerical instability. In a practical implementation, after a certain number of steps, the eigendecomposition of $\tilde{A}^*\tilde{A}$ should be recomputed from scratch, using, for example, the QR algorithm (compare Subsection 2.9.1 and Algorithm 8). This does of course impact the performance in a negative way, but it represents a good compromise between stability and speed.

Remark 4.3.12. Even if the matrix A^*A and its eigendecomposition are not updated, as described in Remark 4.3.10, but completely recomputed in every step, for instance, for reasons of numerical stability or to be able to use off the shelf eigenvalue/eigenvector revealing implementations in highly optimised libraries like LAPACK ([17]), it is not advisable to compute all

the eigenvalues and eigenvectors of A^*A at once. First A^*A should be tridiagonalised with Algorithm 12 resulting in costs of $O(\frac{4}{3}n^3)$. Then we use the bisection algorithm 15 to check if an eigenvalue σ with $\sqrt{\sigma} \leq \varepsilon$ exists and only then we compute its actual value. For this step we can expect costs of $O(n)$. Now we can compute the eigenvector v corresponding to σ of the tridiagonal matrix using inverse iteration 10 in $O(n)$. To obtain the actual eigenvector of A^*A we still need to apply the elementary reflectors used in the tridiagonalisation process to v. This will cost us additionally $O(\frac{4}{3}n^3)$ operations.

Now that we know how to update a SVD of a matrix in our setting essentially in $O\left(12n^2\right)$, it is time to move on to analysing the runtime of the ABM algorithm.

The ABM algorithm and the Buchberger-Möller algorithm for border bases (18) share the same basic structure, with the major difference though that terms are treated one by one and the approximate kernel of the matrix is computed via Algorithm 16. As treating every term separately represents a notable change we will give a more detailed analysis here than in the case of the AVI algorithm.

Proposition 4.3.13. *The runtime of the ABM algorithm is cubic in the number of input points if the procedure that is described in Remark 4.3.10 is used to update the matrix factorisation of the evaluation matrix in lines 6, 7, and 10 of the algorithm.*

Proof. Just like in the proof of Proposition 3.4.7, where we analysed the complexity of the Buchberger-Möller algorithm for border bases, we assume that the number of input points is

$$s = \sum_{i=0}^{p} \binom{n+i-1}{i} = \binom{n+p}{p}$$

and additionally that $\varepsilon = 0$, for which the ABM algorithm will degenerate to the BM algorithm for border bases. The assumption about the number of points makes it easier for us to analyse the individual steps of the algorithm as $\binom{n+p}{p}$ equals the number of terms up to and including degree p in the polynomial ring $P = K[x_1, ..., x_n]$. If the number of points is in between $\binom{n+p}{p}$ and $\binom{n+p+1}{p+1}$ essentially the same arguments apply. This will become apparent once we discuss the complexity of the individual steps. Now we explain why it suffices to treat the case $\varepsilon = 0$. For larger values of ε an improved runtime can be expected, because $\mathcal{O}$ and G will contain less elements and thus less iterations will be needed. So if we consider $\varepsilon = 0$, we obtain an upper bound. Furthermore, to achieve a maximum of computational complexity, we examine the case in which no low degree relations between the input points exists. I.e. the points are chosen in such a way that the number of columns of the evaluation matrix will have to exceed the number of rows until the matrix will have a non-trivial kernel. In this way we are again in a worst case situation, as the number of operations will decrease if already in a lower degree relations are found.
After we have clarified the setting, we can now start with the actual analysis. As $\varepsilon = 0$ the order ideal $\mathcal{O}$ will contain s elements and the border of $\mathcal{O}$ will consist of $\binom{n+p}{p+1}$ elements once the algorithm has terminated. We now have a look at the individual steps:

Lines 1 and 2 are only executed once and have a cost of $O(s+n)$.
Lines 4 to 19 are inside a loop which will be executed $p+1$ times. This is true because by assumption no polynomial relations between the points can be found until the number of columns exceeds the number of rows in the evaluation matrix.
Lines 5 to 16 are again inside a loop and are executed $\binom{n+d-1}{d}$ times where d is the current degree, as essentially every term is treated.
The cost for line 5 is in $O(s)$ as one new term needs to be evaluated on all points. Note that t_i is of the form $t_i = x_k t$ for some $t \in \mathcal{O}$. So we obtain the evaluation of the new term as the component-wise product of an already existing column in the evaluation matrix with $\mathrm{eval}_{\mathbb{X}}(x_k)$. If we would evaluate the whole term every time the costs would exceed $O(s)$. Compare also [46, Remark 6.3.12].
The cost for line 6 is in $O\left(s\left(\binom{n+d-1}{d-1}+i\right)\right)$ for $1 \le d \le p$ and in $O(s(s+1))$ for $d = p+1$, if Remark 4.3.10 is used. Otherwise the cost would be in $O\left(\frac{1}{2}s\left(\binom{n+d-1}{d-1}+i\right)^2\right)$.
The cost for line 7 is in $O\left(\left(\binom{n+d-1}{d-1}+i\right)^2\right)$ for $1 \le d \le p$ and in $O((s+1)^2)$ for $d = p+1$, if again the technique described in Remark 4.3.10 is used. If we would apply, for example, the bisection algorithm (15) to matrix B directly to compute the eigenvector corresponding to the smallest eigenvalue γ, this would lead to costs of $O\left(\frac{4}{3}\left(\binom{n+d-1}{d-1}+i\right)^3\right)$ for $1 \le d \le p$. Please note that the dominating cost factor would be the reduction to tridiagonal form, whereas the actual computation of the eigenvalues would only be quadratic in the size of the input matrix.
As we have chosen $\varepsilon = 0$ this means that the instructions in lines 9 to 12 will only be executed if the smallest eigenvalue γ equals 0, so only if the matrix B has a non-trivial kernel. This happens $\binom{n+p}{p+1}$ times which is exactly the number of elements we will finally have in G. In line 10 the eigenvector of B corresponding to σ is computed. If we use again the instructions from Remark 4.3.10 to perform this operation, it is possible to achieve a runtime of $O\left(2(s+1)^2\right)$.
The cost of line 11 is in $O(s+1)$ as we only form a vector-vector product.
Lines 12, 14, 15, and 18 only involve a constant amount of work and are therefore in $O(1)$.
Finally, line 19 is in $O\left(\binom{n+d-1}{d}\right)$ as all terms of degree d are computed.

If we put all parts together we have total average case costs of

$$
\begin{aligned}
& O\left(\sum_{d=1}^{p}\sum_{i=1}^{\binom{n+d-1}{d}}\left(\underbrace{s}_{5}+\underbrace{s\left(\binom{n+d-1}{d-1}+i\right)}_{6}+\underbrace{\left(\binom{n+d-1}{d-1}+i\right)^2}_{7}+\underbrace{1}_{14}+\underbrace{1}_{15}\right)\right) \\
+\ & O\left(\sum_{i=1}^{\binom{n+p}{p+1}}\left(\underbrace{s}_{5}+\underbrace{s(s+1)}_{6}+\underbrace{(s+1)^2}_{7}+\underbrace{2(s+1)^2}_{9}+\underbrace{(s+1)}_{10}+\underbrace{1}_{12}\right)\right) \\
+\ & O\left(\underbrace{s}_{1}+\underbrace{n}_{2}+\sum_{d=1}^{p+1}\left(\underbrace{1}_{18}+\underbrace{\binom{n+d-1}{d}}_{19}\right)\right).
\end{aligned}
$$

We observe that the most expensive steps in the computation are in lines 6,7, and 10 (which

essentially correspond to the computation of the exact kernel in the BBM algorithm). It suffices to focus on them because for large enough s the cost of the other steps can be neglected. So we focus on the expression

$$\begin{aligned}
& O\left(\sum_{d=1}^{p}\left(\sum_{i=1}^{\binom{n+d-1}{d}}\left(s\left(\binom{n+d-1}{d-1}+i\right)+\left(\binom{n+d-1}{d-1}+i\right)^2\right)\right)\right) \\
+ \;& O\left(\binom{n+p}{p+1}\left(s\left(s+1\right)+3\left(s+1\right)^2\right)\right) \\
= \;& O\left(s\sum_{i=1}^{s} i+\sum_{i=1}^{s} i^2+\binom{n+p}{p}\frac{n}{p}\left(s\left(s+1\right)+3\left(s+1\right)^2\right)\right) \\
= \;& O\left(\frac{1}{2}s^2\left(s+1\right)+\frac{1}{6}s\left(s+1\right)\left(2s+1\right)+s\frac{n}{p}\left(s\left(s+1\right)+3\left(s+1\right)^2\right)\right) \\
= \;& O\left(\frac{1}{2}\left(s^3+s^2\right)+\frac{1}{6}\left(2s^3+3s^2+s\right)+\frac{n}{p}\left(s^3+s^2+3s^3+6s^2+3s\right)\right) \\
= \;& O\left(s^3\left(\frac{5}{6}+4\frac{n}{p}\right)\right).
\end{aligned}$$

As $p \geq 1$ we can conclude that the runtime is in

$$O\left(s^3\left(\frac{5}{6}+4n\right)\right) = O\left(s^3\left(1+4n\right)\right).$$

Thus the dominating cost factor is $s^3\left(1+4n\right)$ and the algorithm is cubic in the number of input points, just like the BBM algorithm for border bases (18) and the AVI algorithm (21). It should be noted that if we do not use Remark 4.3.10 to update the matrix factorisation we would have to expect costs in $O(s^4)$. □

4.3.2 Enhancing the Numerical Stability of the ABM Algorithm

So far we have used Theorem 2.13.6 and Algorithm 16 for computing the solution of the homogeneous least squares problem in the ABM algorithm. For practical purposes the algorithm that follows from the theorem and which we have outlined in great detail in the previous section is a good compromise between numerical stability (see Theorem 4.3.7) and speed, especially since it allows to update the factorisation of A^*A without too much effort. However, the downside is that A^*A needs to be calculated explicitly. As we will see in the following subsection, there is a well-known direct relation between the homogeneous least squares solution of an equation system given by a matrix A and its SVD. Fortunately, there exist advanced techniques for the computation of the former which do not require the explicit formation of A^*A, which further improve on the numerical stability of the computed solution. In case A has several very small singular values and we have chosen a very small ε the additional accuracy may be needed.

The Relationship of the Eigendecomposition of A^*A and AA^* with the SVD

Proposition 4.3.14. *Let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ and let U, Σ, and V be matrices such that $A = U\Sigma V^*$ is a singular value decomposition of A. The matrices U, Σ, and V can be computed via the*

eigendecomposition of A^*A and AA^*. The relations $A^*A = V\Sigma^{\mathrm{tr}}\Sigma V^*$ and $AA^* = U\Sigma\Sigma^{\mathrm{tr}}U^*$ hold.

Proof. Let $U\Sigma V^* = A$ be a singular value decomposition of A. By substituting A with $U\Sigma V^*$ we obtain

$$\begin{aligned} A^*A &= (U\Sigma V^*)^* U\Sigma V^* \\ &= V\Sigma^* U^* U\Sigma V^* = V\Sigma^{\mathrm{tr}}\Sigma V^* \end{aligned}$$

and

$$\begin{aligned} AA^* &= U\Sigma V^* (U\Sigma V^*)^* \\ &= U\Sigma V^* V\Sigma^* U = U\Sigma\Sigma^{\mathrm{tr}}U^*. \end{aligned}$$

Now it can be seen that via the eigendecompositions of AA^* and A^*A the matrices U, Σ and V can be computed. □

Stable Computation of the SVD

We will now sketch the central ideas behind a backward stable algorithm which allows to compute a SVD of a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$. For more details please see [6, Section 8.6]. To compute the matrices U, Σ, and V it is not necessary to form AA^* and A^*A explicitly. The method was originally proposed by Golub and Kahan in [12]. The aim is to apply the symmetric or respectively the Hermitian QR algorithm "implicitly" to A^*A. For this purpose the matrix A is first reduced to upper bidiagonal form using e.g. Algorithm 11. So

$$U_B^* A V_B = \begin{bmatrix} B \\ 0 \end{bmatrix} \text{ with } B = \begin{bmatrix} d_1 & f_1 & \cdots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & f_{n-1} \\ 0 & \cdots & 0 & d_n \end{bmatrix} \in \mathrm{Mat}_n(\mathbb{C}).$$

The problem is now reduced to computing a SVD of matrix B. Let us assume that we form $T = B^*B$ explicitly. Then we compute its eigendecomposition via the Hermitian QR algorithm such that $B^*B = \bar{V}\left(\Sigma^{\mathrm{tr}}\Sigma\right)\bar{V}^*$, where $\bar{V}$ contains the eigenvectors of B as its columns. However, we would still be in the same numerically unfavourable situation as in the beginning. Golub and Kahan could show that it is possible to apply iteratively a series of left- and right-handed Householder transformations to the matrix B with the effect that the accumulated right-handed transformations are up to rounding errors identical to the matrix $\bar{V}$. Let us denote the accumulated left-handed transformations by $\bar{U}$ and the right-handed transformations by $\bar{V}$, then we can write $\bar{B} = \bar{U}B\bar{V}$. The iterative procedure terminates once the entries $\bar{f}_i$ of the matrix $\bar{B}$ have become almost zero. At this stage the matrix $\bar{B}$ contains the singular values of A. Note that the matrices $\bar{U}$ and $\bar{V}$ are unitary, which explains why the computational procedure is numerically stable. In our case the matrix V is usually of interest as well. For this purpose we can obtain V by forming $V_B\bar{V}$. Similarly, the matrix U can be obtained by computing $U_B\bar{U}$. An essential part of the algorithm is the construction of the involved Householder transformations. We will not give details here. However, these can be found in [12].

4.3.3 Shortcomings of the ABM Algorithm

- As for the AVI algorithm, in some rare situations the set $\mathcal{O}$ returned by the ABM algorithm may not be an order ideal of terms. This means that for at least one term not all divisors are included in the set $\mathcal{O}$. If in a subsequent computation it is imperative that the output of the ABM algorithm is a proper approximate border basis, this can be cured by checking during the computation in line 8 if adding a new term t_i to $\mathcal{O}$ would violate the order ideal property. If so, we need to form a polynomial using t_i and the other terms which are currently in $\mathcal{O}$, via the instructions in lines 9 to 11, and append the resulting polynomial g to the set G. The coefficients of this polynomial are again computed via the solution of the homogeneous least squares problem (compare Theorem 2.13.6) in line 10. Of course this will weaken the bounds concerning ε and δ. In the following Subsection 4.3.4 we sketch how practical error bounds can be derived for ε and δ in this modified version of the ABM algorithm.
- Even if the set $\mathcal{O}$ is an order ideal, we have no direct control over the coefficients of the border terms (see Remark 4.3.3). If very small border coefficients show up in G, the bound δ may become too large to be of much practical value, as the approximate border basis will then be far away from an exact border basis. In Subsection 4.5 we present an algorithm which mitigates this problem, by providing more practical bounds for δ.

4.3.4 A Modified ABM Algorithm and a Practical Error Bound

First of all we present an example which demonstrates that the output $\mathcal{O}$ of the standard version of the ABM algorithm is not necessarily an order ideal.

Example 4.3.15. Let $P = \mathbb{R}[x_1, x_2]$, $\varepsilon = 0.12$,
and $\mathbb{X} = [(-0.3715, 0.9734), (-0.9548, 1), (-1, 0.905), (0.9502, 0.603), (0.0965, 0.8715)]$ be given and let σ be the `DegRevLex` term ordering.

- $d = 1, \mathcal{O} = [1]$, $G = [\,]$, $M = (1, ..., 1)^{\text{tr}}$ and $L = [x_1, x_2]$
- $A = \begin{pmatrix} 0.9734 & 1 \\ 1 & 1 \\ 0.905 & 1 \\ 0.603 & 1 \\ 0.8715 & 1 \end{pmatrix}$, solve $\min_x \|Ax\|$ subject to $\|x\| = 1$, $A^*A \approx \begin{pmatrix} 3.889 & 4.352 \\ 4.352 & 5 \end{pmatrix}$,
  $\sqrt{e} \approx 0.238$, $\mathcal{O} = [x_2, 1]$
- $A = \begin{pmatrix} -0.3715 & 0.9734 & 1 \\ -0.9548 & 1 & 1 \\ -1 & 0.905 & 1 \\ 0.9502 & 0.603 & 1 \\ 0.0965 & 0.8715 & 1 \end{pmatrix}$, $A^*A \approx \begin{pmatrix} 2.961 & -1.564 & -1.279 \\ -1.564 & 3.889 & 4.352 \\ -1.279 & 4.352 & 5 \end{pmatrix}$,
  $\sqrt{e} \approx 0.116$, $s \approx (-0.13, -0.763, 0.632)$
- $(-0.13, -0.763, 0.632)\,(x_1, x_2, 1)^{\text{tr}} = -0.13x_1 - 0.763x_2 + 0.632 = g_1$, $G = [g_1]$, $\mathcal{O} = [x_2, 1]$

- $d = 2$, $\partial\mathcal{O} = \{x_2^2, x_1x_2\}$, and $L = [x_2^2, x_1x_2]$
- $A \approx \begin{pmatrix} -0.3616 & 0.9734 & 1 \\ -0.9548 & 1 & 1 \\ -0.905 & 0.905 & 1 \\ 0.5729 & 0.603 & 1 \\ 0.0840 & 0.8715 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 2.196 & -1.707 & -1.564 \\ -1.707 & 3.889 & 4.352 \\ -1.564 & 4.352 & 5 \end{pmatrix}$, $\sqrt{e} \approx 0.133$,
  $\mathcal{O} = [x_1x_2, x_2, 1]$

Now we can observe that x_1 is not in $\mathcal{O}$ which violates the order ideal property. We stop here and do not show the further steps of the algorithm.

Fortunately though, the ABM algorithm (22) can be easily modified such that it always returns an approximate $\mathcal{O}$-border basis.

Line 8 in the algorithm needs to be changed to contain an additional check which makes sure that all divisors of t_i are already in $\mathcal{O}$. We obtain the following version of the algorithm:

Algorithm 23: ABM-OI

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, a small number $\varepsilon \geq 0$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}$-border basis G (see Theorem 4.3.1 and the following text for details)

1 $d := 1$, $\mathcal{O} := [1]$, $G := [\,]$, $M := (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{C})$;
2 $L := [t_1, ..., t_\ell] :=$ all terms of degree 1 ordered decreasingly w.r.t. σ;
3 **repeat**
4 **for** *i := 1* **to** ℓ **do**
5 $A := (\mathrm{eval}_{\mathbb{X}}(t_i), M)$;
6 $B := A^*A$;
7 $\gamma :=$ smallest eigenvalue of B;
8 **if** $\sqrt{\gamma} \leq \varepsilon$ **or** *one divisor of* t_i *is not in* $\mathcal{O}$ **then**
9 $m := |\mathcal{O}|$;
10 $s := (s_{m+1}, s_m, ..., s_1) =$ norm one eigenvector of B w.r.t. to γ;
  `// We assume that` $\mathcal{O} = [o_m, ..., o_1]$
11 $g := s_{m+1}t_i + s_mo_m + ... + s_1o_1$;
12 $G := \mathrm{concat}(G, [g])$;
13 **else**
14 $\mathcal{O} := \mathrm{concat}([t_i], \mathcal{O})$;
15 $M := A$;
16 **end**
17 **end**
18 $d := d + 1$;
19 $L := [t_1, ..., t_\ell] :=$ all terms of degree d in $\partial\mathcal{O}$ ordered decreasingly w.r.t. σ;
20 **until** $L := [\,]$;
21 **return** $(G, \mathcal{O})$;

An immediate consequence of this change is that G may contain polynomials which do no longer vanish ε-approximately on $\mathbb{X}$. Note that the bound δ of the δ-approximate border basis (compare claim 4 of Theorem 4.3.1) has to be adapted as well. It is possible to derive theoretical bounds for ε and δ which are independent of the actual input data. However, these will be too crude to be of any practical value. That is why we choose a more pragmatic approach which requires knowledge about the input data tuple $\mathbb{X}$. First of all, we observe that gaps in the order ideal can only occur in degree two and beyond. We now construct the evaluation matrix $(\mathrm{eval}_{\mathbb{X}}(x_k), ..., \mathrm{eval}_{\mathbb{X}}(x_1), \mathrm{eval}_{\mathbb{X}}(1)) = A \in \mathrm{Mat}_{s,k}(\mathbb{C})$ in such a way that the smallest singular value of A, which we denote by s_r, is still greater than ε, and either $k = n$ or the smallest singular value of the matrix $(\mathrm{eval}_{\mathbb{X}}(x_{k+1}), \mathrm{eval}_{\mathbb{X}}(x_k), ..., \mathrm{eval}_{\mathbb{X}}(x_1), \mathrm{eval}_{\mathbb{X}}(1))$ is smaller than or equal to ε. This means that we choose the largest set of sequential degree one terms (with respect to the chosen term ordering) which has no ε-approximate kernel in its associated evaluation matrix. Because of this construction, the set $\{x_k, ..., x_1, 1\}$ will always be a subset of the set $\mathcal{O}$ for the given value of ε. By Proposition 2.5.35, we know that by adding an additional column to A the smallest singular value $\tilde{s}_{\tilde{r}}$ of this new matrix will be $\leq s_r$. So we have now established that the polynomials in G vanish $\tilde{s}_{\tilde{r}}$-approximately with respect to $\mathbb{X}$. Let us denote $\tilde{s}_{\tilde{r}}$ by $\tilde{\varepsilon}$. Because the steps are the same as in the proof of claim 4 of Theorem 4.3.1, it is now straightforward to derive an upper bound for $\tilde{\delta}$, the bound for the approximate border basis which we obtain for Algorithm 23. We derive $\tilde{\delta} = 2\|\mathbb{X}\|_{\max}\tilde{\varepsilon}/\min_i|\gamma_i|\,\varepsilon + \nu\tilde{\varepsilon}/(\min_i|\gamma_i|)^2\,\varepsilon$. Here $\|\mathbb{X}\|_{\max}$ denotes the maximal absolute coordinate in $\mathbb{X}$ and $\min_i|\gamma_i|$ the minimal border coefficient of all polynomials in $G = \{g_1, ..., g_\nu\}$.

4.3.5 Implementation in ApCoCoA

The ABM algorithm was implemented by the author in the ApCoCoA C++ library ([20]). For reasons of efficiency both a real (double) and complex (complex double) version of the ABM algorithm are available.

At the time of writing only a basic version has been implemented which does not use Remark 4.3.10 to update the matrix factorisation. For the computation of matrix-matrix and matrix-vector products the BLAS software interface was used, which allows for machine dependent optimisations by simply linking to an optimised implementation that makes use of the target computer architecture or additional processing equipment such as graphics cards (e.g. CUDA or OpenCL). For example, $A^{\mathrm{tr}}A$ is computed via the command *dsyrk* or respectively A^*A is computed via the command *zherk*.

For the computation of the eigenvalues and eigenvectors of $A^{\mathrm{tr}}A$ and A^*A the LAPACK software library ([17]) is used. All eigenvalues and eigenvectors are computed at the same time, although this could also be optimised as pointed out in Remark 4.3.12. The commands used for the computation are *dsyev* and *zheev*.

A detailed description of the parameters of the ApCoCoA implementation of the ABM algorithm can be found in Appendix 8.1.

4.4 Approximation by Polynomial Functions

In this section we will discuss some problems arising in the context of industrial applications and how the ABM algorithm can be adapted to provide an adequate solution.

Sometimes we may be in a situation where we want to find out if a measurement can be expressed as a polynomial function in other measurements up to a certain degree and up to a certain error tolerance. The AVI and ABM algorithm do not always give a satisfactory answer to this task as we can see in the following example.

Example 4.4.1. Let $P = \mathbb{R}[x_1, x_2]$ and let

$$\mathbb{X} = [(-2,-1), (-1,0), (-0.01, 0.99), (0,1), (0.01, 0.99), (1,0), (2,-1)].$$

Furthermore, let us assume that $x_2 \approx \sum_{i=0}^{d} c_i x_1^i$. For $d = 6$ we could obtain an exact equality by performing univariate interpolation. However, this is not our goal and we are interested in solutions where $d < 6$. In this simple case we could just solve a least squares problem to obtain the coefficients c_i in $x_2 \approx \sum_{i=0}^{5} c_i x_1^i$ (compare Example 2.10.3). However, in general a set of terms which is well-suited for approximate interpolation is not known up front. For instance if we have measured data in $n \gg 1$ indeterminates we have to work in a polynomial ring with n indeterminates. From a numerical point of view it is not desirable or sometimes even feasible to solve a least squares problem that involves all terms up to degree d, as the underlying problem could be ill-conditioned or even ill-posed. Additionally, we are facing an efficiency problem as there are $\binom{n+d}{d}$ terms up to degree d in the polynomial ring in n indeterminates. These issues represent some of the major challenges that we are facing and trying to address.
We now apply the ABM algorithm (22) for $\varepsilon = 0.1$ and obtain the sets $G = \{g_1, ..., g_4\}$ and $\mathcal{O} = \{1, x_2, x_1, x_2^2, x_1x_2\}$ with

$$\begin{aligned} g_1 &\approx x_1^2 - 0.99x_2^2 + 1.99x_2 - 1 \\ g_2 &\approx x_2^3 - 0.99x_2 \\ g_3 &\approx x_1x_2^2 + 0.99x_1x_2 \\ g_4 &\approx x_1^2x_2 + 2.00x_2^2 - 1.99x_2. \end{aligned}$$

As we can see there is no polynomial of the desired form in G and it is also not possible to rewrite one polynomial using the other almost vanishing relations to accomplish the task.

First of all, we try to reformulate the problem by splitting the input data set into two parts. So, we start out with a tuple of measurements $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, $\mathbb{V} = [v_1, ..., v_s]$ with $v_i \in \mathbb{C}$, and $P = \mathbb{C}[x_1, ..., x_n, x_{n+1}]$. Furthermore, we assume that v_i can be expressed approximately by a polynomial function in the indeterminates $x_1, ..., x_n$ evaluated on the coordinates of p_i, e.g.

$$\begin{aligned} f : \mathbb{C}^n &\to \mathbb{C} \\ (x_1, ..., x_n) &\mapsto f(x_1, ..., x_n) \\ f(p_i) &\approx v_i \end{aligned}$$

for all $1 \leq i \leq s$. We assume that the datasets $\mathbb{X}$ and $\mathbb{V}$ contain measurement errors. That is why multivariate polynomial interpolation, for instance, via the Buchberger-Möller algorithm is not an option as the resulting polynomial(s) would also encode the noise that is present in the input data and therefore obfuscate the underlying relations. We know that in general AVI and ABM compute an approximate border basis w.r.t. a tuple of input points $\mathbb{X}$, so each of the polynomials in the set G returned by the algorithm has the property that $f(p_i) \approx 0$ for all $1 \leq i \leq s$. The naive approach of just merging $\mathbb{X}$ and $\mathbb{V}$ in the tuple $\tilde{\mathbb{X}}$ by adding v_i as a new coordinate to p_i and then to apply the AVI or ABM algorithm to $\tilde{\mathbb{X}}$ does in general not produce useful results as only polynomial relations f in the indeterminates $x_1, ..., x_n, x_{n+1}$ will be found, such that $f(p_i, v_i) \approx 0$ holds, which cannot be easily solved for x_{n+1}. This is precisely the behaviour that we have also observed in Example 4.4.1. Even if relations of the desired form, which can be solved explicitly for x_{n+1}, show up in G, then the coefficients by which we may have to divide to get the explicit representation of x_{n+1} could be very small. If we evaluate this polynomial on p_i the actual value may differ significantly from v_i. This can happen because the original polynomial was not guaranteed initially to vanish at the point (p_i, v_i). This exemplifies that there is no easy way to use the polynomials contained in the approximate border basis to obtain relations of the the form $f(p_i) \approx v_i$.

One possible solution is to use the set $\mathcal{O} = \{t_1, ..., t_\mu\}$ as an interpolation basis. So after the ABM or AVI algorithm has terminated we solve the inhomogeneous least squares problem

$$\min_s \left\| \left(\mathrm{eval}_{\mathbb{X}}(t_1), ..., \mathrm{eval}_{\mathbb{X}}(t_\mu)\right) s - \mathbb{V}^{\mathrm{tr}} \right\|_2 = \min_s \left\| Ms - \mathbb{V}^{\mathrm{tr}} \right\|_2$$

with $s \in \mathbb{C}^\mu$. Thus it is always assured that we obtain a polynomial function f for which $f(p_i) \approx v_i$. However, the disadvantage of this approach is that we have no direct control over the residual error $\left\| Ms - \mathbb{V}^{\mathrm{tr}} \right\|_2$ as the terms contained in $\mathcal{O}$ and the tuple $\mathbb{V}$ may have been unrelated. As a rule of thumb we can assume that for smaller values of ε we obtain a set $\mathcal{O}$ which contains more elements and so the residual will be smaller whereas for larger values of ε we will obtain a set $\mathcal{O}$ containing less elements which leads to a larger residual error. Unfortunately though, in general it may happen that this rule of thumb does not hold as the following example shows.

Example 4.4.2. Let $\mathbb{X}$ be a tuple of input points and let $0 < \varepsilon_1 < \varepsilon_2$. Additionally, let $\mathcal{O}_1$ be the order ideal associated with the computation using ε_1 and let $\mathcal{O}_2$ be the order ideal associated with ε_2. Now we assume that $|\mathcal{O}_1| > |\mathcal{O}_2|$ and that $\mathcal{O}_2 \setminus (\mathcal{O}_1 \cap \mathcal{O}_2) = \{t_j, ..., t_k\} \neq \emptyset$. If $\mathbb{V} = \mathrm{eval}_{\mathbb{X}}(t_k)$, then $\mathcal{O}_2$ will in fact be the order ideal producing the better approximation of though it contains less elements than $\mathcal{O}_1$.

For practical purposes it is desirable to be able to specify a maximal allowed tolerance for the residual error upfront, so this will be one of the design goals for the extended ABM algorithm.

Another problem of the described approach is that it will provide exactly one solution for a given set $\mathcal{O}$. Even though $\left\| Ms - \mathbb{V}^{\mathrm{tr}} \right\|$ may be smaller than a given threshold number τ there may be several non-identical subsets of $\mathcal{O}$ which would still meet our requirement. One could in theory test several or even all possible subsets but this is very inefficient in practice. In the extended

ABM algorithm we will also try to mitigate this problem by making use of the already computed matrix decompositions which we utilise to handle the homogeneous least squares problem. This will allow us to solve the inhomogeneous least squares problem more efficiently, than by keeping the computations of $\mathcal{O}$ and G separated from the computation of the approximating polynomial functions for $\mathbb{V}$.

In the following section we will now present a modification of the ABM algorithm which can be used in a direct way to solve the task at hand. The idea is the following: We still compute the approximate vanishing ideal of $\mathbb{X}$ with respect to a parameter ε, thus making sure that the evaluations of the elements in the set $\mathcal{O}$ remain approximately linearly independent. What we do additionally is that whenever the list $\mathcal{O}$ is enlarged we check if we can construct already with these elements a solution to the inhomogeneous least squares problem such that $\left\|Ms - \mathbb{V}^{\mathrm{tr}}\right\| < \tau$. On the one hand this will provide us with a set of solutions (if they exist) that have guaranteed fitting properties. Moreover we have a basis of almost vanishing polynomials which can be used to modify the computed solutions via linear combinations.

4.4.1 The Extended ABM Algorithm

Now present the so-called extended ABM algorithm and discuss its properties.

Algorithm 24: Extended ABM Algorithm

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, $\mathbb{V} = [v_1, ..., v_s]$ with $v_i \in \mathbb{C}$ and $\|\mathbb{V}\| > \varepsilon_{machine}$, small numbers $\varepsilon \geq 0, 0 \leq \tau < \|\mathbb{V}\|$, $D \in \mathbb{N}$ such that $D < s$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}$-border basis G, a list H of polynomials (see Theorem 4.4.3 for details)

$d := 1$, $\mathcal{O} := [1]$, $G := [\,]$, $H := [\,]$, $M := (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{C})$;
$L := [t_1, ..., t_\ell] =$ all terms of degree 1 ordered decreasingly w.r.t. σ;
repeat
  for *i := 1* **to** ℓ **do**
    $m := |\mathcal{O}|$;
    `// We assume that` $\mathcal{O} = [o_m, ..., o_1]$
    $A := (\mathrm{eval}_{\mathbb{X}}(t_i), M)$;
    $B := A^*A$;
    $\gamma :=$ smallest eigenvalue of B;
    if $\sqrt{\gamma} \leq \varepsilon$ **then**
      $s := (s_{m+1}, s_m, ..., s_1) :=$ norm one eigenvector of B w.r.t. to γ;
      $g := s_{m+1}t_i + s_m o_m + ... + s_1 o_1$;
      $G := \mathrm{concat}(G, [g])$;
    else
      $s := (s_{m+1}, s_m, ..., s_1) =$ solution of the least squares problem $\min_s \left\|As - \mathbb{V}^{\mathrm{tr}}\right\|$ (compare algorithms in Section 2.11);
      if $\left\|As - \mathbb{V}^{\mathrm{tr}}\right\| \leq \tau$ **then**
        $h := s_{m+1}t_i + s_m o_m + ... + s_1 o_1$;
        $H := \mathrm{concat}(H, [h])$;
      end
      $\mathcal{O} := \mathrm{concat}([t_i], \mathcal{O})$;
      $M := A$;
    end
  end
  $d := d + 1$;
  $L := [t_1, ..., t_\ell] :=$ all terms of degree d in $\partial\mathcal{O}$ ordered decreasingly w.r.t. σ;
until $L = [\,]$ *or* $d > D$;
return $(G, \mathcal{O}, H)$;

Theorem 4.4.3. *This algorithm computes three sets $G = \{g_1, ..., g_\nu\}$, $\mathcal{O} = \{t_1, ..., t_\mu\}$, and $H = \{h_1, ..., h_\kappa\}$ which have the following properties:*

1. *All the polynomials in G are unitary and generate an ε-approximate vanishing ideal of $\mathbb{X}$.*
2. *There is no unitary polynomial in $\langle\mathcal{O}\rangle_{\mathbb{C}}$ which vanishes ε-approximately on $\mathbb{X}$.*

3. *For all polynomials* h_i *in* H *we have* $\|\mathrm{eval}_{\mathbb{X}}(h_i) - \mathbb{V}\| \leq \tau$.

4. *The condition number* κ *of the inhomogeneous least squares problem which is solved in line 14 is bounded by*

$$\frac{s \cdot \|\mathbb{X}\|_{\max}^{D}}{\varepsilon} + \left(\frac{s \cdot \|\mathbb{X}\|_{\max}^{D}}{\varepsilon}\right)^2 \sqrt{1 - \frac{(\|\mathbb{V}\| - \tau)^2}{\|\mathbb{V}\|^2}} \Big/ \frac{\|\mathbb{V}\| - \tau}{\|\mathbb{V}\|}.$$

If additionally the parameter D *is chosen large enough, e.g.* $D = s - 1$*, such that the algorithm does not terminate prematurely, then also the following properties hold:*

5. *If* $\mathcal{O}$ *is an order ideal of terms, then the set* $\tilde{G} = \{(1/\mathrm{LC}_\sigma(g))\, g \,|\, g \in G\}$ *is an* $\mathcal{O}$*-border prebasis.*

6. *If* $\mathcal{O}$ *is an order ideal of terms, then the set* $\tilde{G}$ *is an* δ*-approximate border basis with*

$$\delta = 2\,\|\mathbb{X}\|_{\max} / \min_i |\gamma_i| + \nu / \left(\min_i |\gamma_i|\right)^2.$$

Here $\|\mathbb{X}\|_{\max}$ *denotes the maximal absolute coordinate in* $\mathbb{X}$ *and* $\min_i |\gamma_i|$ *the minimal border coefficient of all polynomials in* G*.*

Proof. First of all we note that the ABM and the extended ABM algorithm share the same basic structure. The latter algorithm only contains some additional processing steps in lines 14 to 18 and an additional degree check in line 25.

The individual steps are well-defined because of the same reasons as in the ABM algorithm. An explanation is contained in the first part of the proof of Theorem 4.3.1.

Next we discuss finiteness. In line 25 termination is assured by checking if the current degree d exceeds a user specified upper bound D. If so the execution will stop and a possibly partial result, i.e. no complete approximate border basis, will be returned. However, even if we would drop the additional check in line 25 the algorithm would still terminate. This is shown in the second part of the proof of Theorem 4.3.1.

Claims 1, 2, 5, and 6 in fact refer to properties of the result of the ABM algorithm, which we have already discussed and proven in Theorem 4.3.1. Note, that we have to assume that D is chosen large enough, i.e. $D = s - 1$, such that a full approximate border basis will be returned in the end.

Claim 3 holds because the polynomials which are added to H are tested to have this property in line 15.

For each polynomial $h_i \in H$ we let $A_i = \mathrm{eval}_{\mathbb{X}}(\mathrm{supp}(h_i))$. Let us denote the coefficient vector of h_i with respect to $\mathrm{supp}(h_i)$ by c_i. This means that $\mathrm{eval}_{\mathbb{X}}(h_i) = A_i c_i$ holds. To prove claim 4 let us first cast a result from Theorem 2.12.1 into our setting which states that the general condition number of the inhomogeneous least squares problem is

$$\kappa(A_i) + \frac{\tan(\theta_i)\,\kappa(A_i)^2}{\eta_i},$$

where $\cos(\theta_i) = \frac{\|\mathrm{eval}_{\mathbb{X}}(h_i)\|}{\|\mathbb{V}\|} = \frac{\|A_i c_i\|}{\|\mathbb{V}\|}$ for all $h_i \in H$ and $\eta_i = \frac{\|A_i\|\|c_i\|}{\|A_i c_i\|}$. First we observe that $1 \leq \eta_i \leq \kappa(A)$. The first inequality follows from $\|A\|\|c_i\| \geq \|Ac_i\|$. For the second inequality, note that c_i is not the zero vector as $\|\mathbb{V}\| \neq 0$ and A has only a trivial kernel which means that $\|Ac_i\| \neq 0$. With the help of Proposition 2.6.5 we can now compute $\kappa(A_i) = \|A_i\|\left\|A_i^+\right\| = \|A_i\| \frac{\left\|A_i^+\right\|\|A_i c_i\|}{\|A_i c_i\|} \geq \|A_i\| \frac{\left\|A_i^+ A_i c_i\right\|}{\|A_i c_i\|} = \frac{\|A_i\|\|c_i\|}{\|A_i c_i\|}$. Therefore, we obtain

$$\kappa(A_i) + \frac{\tan(\theta_i)\,\kappa(A_i)^2}{\eta_i} \leq \kappa(A_i) + \tan(\theta_i)\,\kappa(A_i)^2 .$$

Next we use the equation $\tan\left(\cos^{-1}(x)\right) = \frac{\sqrt{1-x^2}}{x}$ in order to obtain

$$\tan(\theta_i) = \sqrt{1 - \frac{\|A_i c_i\|^2}{\|\mathbb{V}\|^2}} \Bigg/ \frac{\|A_i c_i\|}{\|\mathbb{V}\|} .$$

Because of the triangle inequality and because of claim 5 we can establish that $\|A_i c_i\| \geq \|\mathbb{V}\| - \|A_i c_i - \mathbb{V}\| \geq \|\mathbb{V}\| - \tau > 0$. So

$$\tan(\theta_i) \leq \sqrt{1 - \frac{(\|\mathbb{V}\| - \tau)^2}{\|\mathbb{V}\|^2}} \Bigg/ \frac{\|\mathbb{V}\| - \tau}{\|\mathbb{V}\|} .$$

Now it remains to bound $\kappa(A_i) = \|A_i\|\left\|A_i^+\right\|$. Because we make sure during the computation that the evaluation matrices A_i have no singular values smaller than ε we thus know that $\left\|A_i^+\right\| < \frac{1}{\varepsilon}$. Additionally, for any $A_i \in \mathrm{Mat}_{m,n}(\mathbb{C})$ the inequality $\|A_i\| \leq \sqrt{mn}\,\|A_i\|_{\max}$ holds (see Proposition 2.3.20). In our case the matrix dimensions can at most be $m = n = s$. The entries of the evaluation matrix are products of powers of the entries of $\mathbb{X}$. As $\mathcal{O}$ can at most contain s elements the maximal degree is naturally bounded by s, but as we are using an artificial degree bound in form of the input parameter D (see line 25 of the algorithm) we can be more precise and state that the limit is in fact D. So $\|A_i\| \leq s \cdot \|\mathbb{X}\|_{\max}^D$ where $\|\mathbb{X}\|_{\max}$ is the maximal absolute entry in the input data set $\mathbb{X}$. Now we have established that $\kappa(A_i) \leq s \cdot \|\mathbb{X}\|_{\max}^D \frac{1}{\varepsilon}$. If we collect all the facts we obtain the inequality

$$\kappa = \kappa(A_i) \leq \frac{s \cdot \|\mathbb{X}\|_{\max}^D}{\varepsilon} + \left(\frac{s \cdot \|\mathbb{X}\|_{\max}^D}{\varepsilon}\right)^2 \sqrt{1 - \frac{(\|\mathbb{V}\| - \tau)^2}{\|\mathbb{V}\|^2}} \Bigg/ \frac{\|\mathbb{V}\| - \tau}{\|\mathbb{V}\|} ,$$

which concludes the proof. □

Remark 4.4.4. In case $\|\mathbb{V}\|$ is very small, e.g. $\|\mathbb{V}\| \approx \varepsilon_{machine}$, we are essentially considering a homogeneous least squares problem. Thus, the ABM algorithm (22) should be used instead of the extended ABM algorithm (24), as the polynomials in H (if any) have very small coefficients.

Now we tend to have a look at an example which demonstrates the operation of the extended ABM algorithm.

Example 4.4.5. Let $P = \mathbb{R}[x_1]$, $\mathbb{X} = [(-2), (-1), (-0.01), (0), (0.01), (1), (2)]$, $\mathbb{V} = [(-1), (0), (0.99), (1), (0.99), (0), (-1)]$, and let σ be the `DegRevLex` term ordering. We now apply the extended ABM algorithm (24) with $\varepsilon = 0.1$, $\tau = 10^{-3}$, and $D = 5$:

- $d = 1$, $\mathcal{O} = \{1\}$, $G = \emptyset$, $M = (1, ..., 1)^{\mathrm{tr}}$, and $L = [x_1]$
- $A = \begin{pmatrix} -2 & -1 & -0.01 & 0 & 0.01 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{tr}}$, $\sqrt{\gamma} \approx 1.62 > \varepsilon = 0.1$
- $\min_x \left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\|$, $x \approx (0, 0.140)$, $\left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\| = 2.196$, $\mathcal{O} = \{x_1, 1\}$, and $M = A$
- $d = 2$, $\partial\mathcal{O} = \left\{x_1^2\right\}$, and $L = \left[x_1^2\right]$
- $A = \begin{pmatrix} 4 & 1 & 0.01^2 & 0 & 0.01^2 & 1 & 4 \\ -2 & -1 & -0.01 & 0 & 0.01 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{tr}}$, $\sqrt{\gamma} \approx 1.39 > 0.1$
- $\min_x \left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\|$, $x \approx (-0.476, 0, 0.821)$, $\left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\| = 0.583$, $\mathcal{O} = \left\{x_1^2, x_1, 1\right\}$, and $M = A$
- $d = 3$, $\partial\mathcal{O} = \left\{x_1^3\right\}$, and $L = \left[x_1^3\right]$
- $A = \begin{pmatrix} -8 & -1 & -0.01^3 & 0 & 0.01^3 & 1 & 8 \\ 4 & 1 & 0.01^2 & 0 & 0.01^2 & 1 & 4 \\ -2 & -1 & -0.01 & 0 & 0.01 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{tr}}$, $\sqrt{\gamma} \approx 1.00 > 0.1$
- $\min_x \left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\|$, $x \approx (0, -0.476, 0, 0.821)$, $\left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\| = 0.583$, $\mathcal{O} = \left\{x_1^3, x_1^2, x_1, 1\right\}$, and $M = A$
- $d = 4$, $\partial\mathcal{O} = \left\{x_1^4\right\}$, and $L = \left[x_1^4\right]$
- $A = \begin{pmatrix} 16 & 1 & 0.01^4 & 0 & 0.01^4 & 1 & 16 \\ -8 & -1 & -0.01^3 & 0 & 0.01^3 & 1 & 8 \\ 4 & 1 & 0.01^2 & 0 & 0.01^2 & 1 & 4 \\ -2 & -1 & -0.01 & 0 & 0.01 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{tr}}$, $\sqrt{\gamma} \approx 0.885 > 0.1$
- $\min_x \left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\|$, $x \approx (0.165, 0, -1.158, 0, 0.993)$, $\left\|Ax - \mathbb{V}^{\mathrm{tr}}\right\| = 0.008$, $h_1 \approx -0.165x_1^4 + 1.158x_1^2 - 0.993$, $H = \{h_1\}$, $\mathcal{O} = \left\{x_1^4, x_1^3, x_1^2, x_1, 1\right\}$, and $M = A$
- $d = 5$, $\partial\mathcal{O} = \left\{x_1^5\right\}$, and $L = \left[x_1^5\right]$
- $A = \begin{pmatrix} -32 & -1 & -0.01^5 & 0 & 0.0^5 & 1 & 32 \\ 16 & 1 & 0.01^4 & 0 & 0.01^4 & 1 & 16 \\ -8 & -1 & -0.01^3 & 0 & 0.01^3 & 1 & 8 \\ 4 & 1 & 0.01^2 & 0 & 0.01^2 & 1 & 4 \\ -2 & -1 & -0.01 & 0 & 0.01 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathrm{tr}}$, $\sqrt{\gamma} \approx 0.0087 < 0.1$,
  subject to $\|x\| = 1$, $x \approx (0.154, 0, -0.771, 0, 0.617, 0)$, $g_1 \approx 0.154x_1^5 - 0.771x_1^3 + 0.617x_1$
- The algorithm terminates because L is empty. Finally, we obtain the sets $G = \{g_1\}$, $H = \{h_1\}$ and $\mathcal{O} = \left\{x_1^4, x_1^3, x_1^2, x_1, 1\right\}$, where G is an approximate $\mathcal{O}$-border basis.

Remark 4.4.6. The extended ABM algorithm can be used to check if a polynomial function (in general or up to a specified degree D) exists such that $\mathbb{V}$ can be expressed τ-approximately

by the input data $\mathbb{X}$. For this purpose we need to set the parameter ε to 0. If H is empty upon termination of the algorithm, then we have the guarantee that no such relations exists.

Next, we look at an example that emphasises that in general no solution needs to exist, which means that H will contain no polynomials.

Example 4.4.7. [Existence of Solutions]

Let $P = \mathbb{C}[x_1, x_2]$, $\mathbb{X} = (0,0)$, $\mathbb{V} = (1,0)$, $\varepsilon = 0$ and $\tau = 0.1$. When we use the extended ABM algorithm, all least squares problems we have to solve in line 14 are of the form

$$\min_s \left\| \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \end{pmatrix} s - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\|$$

with $s = (s_{m+1}, ..., s_1)^{\mathrm{tr}} \in \mathbb{C}^{m+1}$. A polynomial is added to H if it satisfies

$$\sqrt{|s_1 - 1|^2 + |s_1|^2} \quad < \quad 0.1.$$

This is equivalent to the following conditions:

$$\begin{aligned} (s_1 - 1)(\bar{s}_1 - 1) + s_1\bar{s}_1 &< 0.01 \\ 2s_1\bar{s}_1 - s_1 - \bar{s}_1 + 1 &< 0.01 \\ 2\left(s_1\bar{s}_1 - 0.5s_1 - 0.5\bar{s}_1 + 0.5^2 + 0.25\right) &< 0.01 \\ 2(s_1 - 0.5)(\bar{s}_1 - 0.5) + 0.5 &< 0.01 \\ 2|s_1 - 0.5|^2 &< -0.49. \end{aligned}$$

As we can see, there exists no number s that would satisfy this inequality.

4.4.2 Runtime Complexity of the Extended ABM Algorithm

Proposition 4.4.8. *The runtime of the extended ABM algorithm is cubic in the number of input points if both the SVD and the QR decomposition are updated during the computation. Details about how to update the SVD are contained in Remark 4.3.10. An updating procedure for the QR decomposition can be found in [6, Section 12.5.2].*

Proof. The algorithm shares the basic structure with the ABM algorithm; additional work is performed in lines 14 to 20. The most expensive new step is the computation of the solution of the inhomogeneous least squares problem in line 14. It can be solved efficiently in $O\left(2mn^2 - \frac{2}{3}n^3\right)$ using the QR decomposition of $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, compare Proposition 2.11.3 and Remark 2.11.4 for details. The process can again be accelerated if the QR decomposition is not recomputed in each step but updated accordingly. It is possible to achieve this task in $O(mn)$ via a sequence of Givens rotations (see [6, Section 12.5.2]). After we have computed the QR decomposition we still need to solve the least squares problem in $O\left(mn + \frac{1}{2}n(n+1)\right)$. Assuming that we are in the same setting as when investigating the runtime of the ABM algorithm (the same number of input points s with the same properties and $\varepsilon = 0$), we can take for granted that the

QR decomposition is at most updated s times, as in this case the matrix A will have full rank s and no new elements will go into $\mathcal{O}$. This means that we can expect additional costs in

$$O\left(\sum_{i=1}^{s}\left(2si+\frac{1}{2}i\left(i+1\right)\right)\right)$$

to update the QR decomposition and to compute the least squares solution. We can simplify the expression to

$$\begin{aligned}
& O\left(\sum_{i=1}^{s}\left(2si+\frac{1}{2}i\left(i+1\right)\right)\right)\\
= \;& O\left(s^2\left(s+1\right)+\frac{1}{6}s\left(s+1\right)\left(s+2\right)\right)\\
= \;& O\left(s^3+s^2+\frac{1}{6}\left(s^3+3s^2+2s\right)\right)\\
= \;& O\left(s^3\right).
\end{aligned}$$

Thus we conclude that the runtime of the algorithm remains cubic in s. □

4.4.3 Enhancing the Numerical Stability of the Extended ABM Algorithm

Sometimes it may be necessary to improve the accuracy of Algorithm 24 e.g. when we are dealing with an ill conditioned least squares problem in line 14. Even though the solution of the inhomogeneous least squares problem via QR decomposition is provably backward stable, it is possible to compute an empirically more accurate solution with the help of QR decomposition with column pivoting or with the help of the SVD. This behaviour and the associated accuracy performance tradeoff is investigated in [7, Sections 3.5 and 3.6].

Now we explain the necessary steps and the theoretical background for solving the inhomogeneous least squares problem with the help of the SVD in more detail. For further elaborations consider [5, pages 83-84]). Please note that this enhancement can be combined with the ideas from Subsection 4.3.2, which concern the stable computation of the solution of the homogeneous least squares problem and therefore the polynomials in the set G returned by the ABM and the extended ABM algorithm.

First, we briefly recall the setup of the inhomogeneous least squares problem (compare Definition 2.10.1). Let $A \in \mathrm{Mat}_{m,n}\left(\mathbb{C}\right)$ with $m \geq n$ and $b \in \mathbb{C}^m \setminus \{0_m\}$. In our setting we know additionally that A has full rank n. We are interested in finding vectors $x \in \mathbb{C}^n$ such that $\left\|Ax-b\right\|_2$ is minimised. Let $U\Sigma V^*$ be the reduced SVD (see Definition 2.5.33) of A. Please note, that $\Sigma \in \mathrm{Mat}_n\left(\mathbb{C}\right)$ is invertible because A has full rank. The orthogonal projector onto $\mathrm{im}\left(A\right)$ is then given by $P_A = UU^*$. In the context of Theorem 2.11.1, which was concerned with characterising the solutions of the inhomogeneous least squares problem, we can write

$$\begin{aligned}
Ax &= P_A b \quad \Longleftrightarrow\\
Ax &= UU^* b \quad \Longleftrightarrow\\
U\Sigma V^* x &= UU^* b
\end{aligned}$$

and consequently

$$x = V\Sigma^{-1}U^*b.$$

This means that in order to solve the least squares problem, we first have to compute the reduced SVD of A and then apply the individual matrices U^*, Σ^{-1} and V to b. The cost is dominated by the computation of the SVD, which is in general more expensive than solving the problem with the help of the QR decomposition. However, if additional numerical stability is required the SVD provides it at a reasonable overhead. The total cost for solving an inhomogeneous least squares problem with the SVD is according to [5, page 84] in $O\left(2mn^2 + 11n^3\right)$.

Remark 4.4.9. If we have used the ideas from Subsection 4.3.2 to enhance the numerical stability of the polynomials in G, then this means that we have already computed a part of a SVD of A in form of the matrices Σ and V. Of course these can be reused in the solution process of the inhomogeneous least squares problem to further speed up the computation. The missing matrix U can for example be computed as $U = AV\Sigma^{-1}$.

4.4.4 Shortcomings of the Extended ABM Algorithm

- As for the AVI and ABM algorithm, in some cases the set $\mathcal{O}$ may not be an order ideal of terms, meaning that for at least one term not all divisors are included in the set $\mathcal{O}$. If in a subsequent computation it is imperative that the output of the Extended ABM algorithm is a proper approximate border basis, this can be prevented by checking during the computation if adding a new term t_i to $\mathcal{O}$ in line 19 of the algorithm would violate the order ideal property. If so, we need to form a polynomial using t_i and the other terms which are currently in $\mathcal{O}$, via the instructions in lines 10 and 11, and append the resulting polynomial g to the set G. The coefficients of this polynomial are again computed via the solution of the homogeneous least squares problem (compare Theorem 2.13.6) in line 10. Of course this will weaken the bounds concerning ε and δ. If so, we need to form a polynomial from t and the other terms which are currently in $\mathcal{O}$ and add the resulting polynomial to the set G. The coefficients of this polynomial are again computed via the solution of the homogeneous least squares problem (compare Theorem 2.13.6). Of course this will weaken the bounds concerning ε and δ. The consequences which also apply in this case have already been analysed in Subsection 4.3.4.
- Just like in the case of the ABM algorithm, even if the set $\mathcal{O}$ is an order ideal, we have no direct control over the coefficients of the border terms. If very small border coefficients show up in G, the bound δ may become too large to be of much practical value, as the approximate border basis will then be far away from an exact border basis. In Subsection 4.5 we present an algorithm which partly mitigates this problem.

4.4.5 Implementation in ApCoCoA

The extended ABM algorithm has been implemented by the author in the ApCoCoA C++ library ([20]). A real (double) and complex (double) version of the algorithm is available.

At the time of writing only a basic version was implemented which does not use Remark 4.3.10 or the ideas from [6, Section 12.5.2] to update the matrix factorisations. For the computation of matrix-matrix and matrix-vector products we us again the BLAS software interface, which allows for machine dependent optimisations by simply linking to an optimised implementation which makes use of the target computer architecture or additional equipment like graphics cards (e.g. CUDA or OpenCL). For instance, $A^{\mathrm{tr}}A$ is computed via the command *dsyrk* or respectively A^*A is computed via the command *zherk*.

For the computation of the eigenvalues and eigenvectors of the symmetric matrix $A^{\mathrm{tr}}A$, and of the Hermitian matrix A^*A as well as for the solution of the inhomogeneous least squares problem, the LAPACK software library [17] is used.

All eigenvalues and eigenvectors are computed at the same time though this could also be optimised as pointed out in Remark 4.3.12. The LAPACK commands used are *dsyev* and *zheev*.

The commands used for the solution of the least squares problem are *dgels* and *zgels* which are based upon the QR decomposition. As pointed out in Subsection 4.4.3 we could switch to solving the least squares problem via the SVD by using the commands *dgelss* and *zgelss* instead.

A detailed description of the parameters of the ApCoCoA implementation of the extended ABM algorithm can be found in Appendix 8.1.

4.5 The BB ABM Algorithm

The algorithm which we now present is in large parts similar to the one proposed in [29] by C. Fassino. However, we investigate a variant for border bases and not for Gröbner bases. Accordingly, we focus on the properties of approximate border bases which are not discussed in [29] for obvious reasons. The Border Bases Approximate Buchberger-Möller (BB ABM) algorithm has the property that the error δ can be better bounded compared to the AVI and ABM algorithm (see Remark 4.3.3). This makes the algorithm a suitable choice if a guaranteed and possibly small error bound on δ is required, for example, if the set G is supposed to be used as input for one of the rational recovery algorithms (cf. Chapter 5).

Before we can detail and discuss the algorithm we first have to recall some relations between the inhomogeneous least squares problem and the homogeneous least squares problem.

Theorem 4.5.1. *Let $m \geq n \geq 1$, let $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be of full rank, and let additionally $b \in \mathbb{C}^m \setminus \{0_m\}$. By $[b, A] \in \mathrm{Mat}_{m,n+1}(\mathbb{C})$ we denote the matrix which we obtain if we prepend b as a new column to A. Then let $r = AA^+b - b$, where A^+ is the pseudoinverse of A, be the residual of the ordinary least squares problem, and let $\sigma_k(M)$ be the k-th singular value of the matrix M. In this setting the inequality*

$$\frac{\|r\|_2}{\|[b, A]\|_2} \leq \frac{\sigma_{n+1}([b, A])}{\sigma_n(A)}$$

holds.

Proof. See [31, Corollary 6.1] where we use the parameter of the scaled TLS problem $\gamma = 1$. □

Corollary 4.5.2. *Let $\varepsilon \in \mathbb{R}_0^+$ and let $A = (a_n, ..., a_1) \in \mathrm{Mat}_{m,n}(\mathbb{C})$ be of full rank with $m \geq n \geq 1$ such that for all $1 \leq i < n$ the relation $\left\| A_{1:m,1:i} A_{1:m,1:i}^+ a_{i+1} - a_{i+1} \right\|_2 > \varepsilon$ holds. This means that the least squares residual of the first i columns of A with respect to the $i+1$-th column of A is larger than ε. Then the smallest singular value of A has a lower bound of*

$$\sigma_{\min}(A) > \frac{\varepsilon^{n-1} \|a_1\|}{\sqrt{m}^{n-1} \Pi_{i=2}^n \left(\sqrt{i} \|A_{1:m,1:i}\|_{\max} \right)},$$

where $\|\cdot\|_{\max}$ denotes the max matrix norm of A (see Definition 2.3.18).

Proof. Let $r_i = A_{1:m,1:i} A_{1:m,1:i}^+ a_{i+1} - a_{i+1}$ where A^+ denotes the pseudoinverse of a matrix A. First we observe that $\varepsilon < \|r_i\|$. If we use Theorem 4.5.1 together with Proposition 2.3.20 we derive the inequality

$$\sigma_n(A_{1:m,1:n}) \geq \frac{\|r_{n-1}\| \sigma_{n-1}(A_{1:m,1:n-1})}{\|A_{1:m,1:n}\|} > \frac{\varepsilon \sigma_{n-1}(A_{1:m,1:n-1})}{\sqrt{mn} \|A_{1:m,1:n}\|_{\max}}.$$

Clearly, we can apply Theorem 4.5.1 together with Proposition 2.3.20 again to bound $\sigma_{n-1}(A_{1:m,1:n-1})$. If we do this iteratively in total $n-1$ times we obtain the following sequence of inequalities

$$\begin{aligned}
\sigma_n(A_{1:m,1:n}) &\geq \frac{\|r_{n-1}\| \sigma_{n-1}(A_{1:m,1:n-1})}{\|A_{1:m,1:n}\|} > \frac{\varepsilon \sigma_{n-1}(A_{1:m,1:n-1})}{\sqrt{mn} \|A_{1:m,1:n}\|_{\max}} \\
&> \frac{\varepsilon}{\sqrt{mn} \|A_{1:m,1:n}\|_{\max}} \frac{\varepsilon \sigma_{n-2}(A_{1:m,1:n-2})}{\sqrt{m(n-1)} \|A_{1:m,1:n-1}\|_{\max}} \\
&> \ldots > \frac{\varepsilon^{n-1} \sigma_1(a_1)}{\sqrt{m}^{n-1} \prod_{i=2}^n \left(\sqrt{i} \|A_{1:m,1:i}\|_{\max} \right)} \\
&= \frac{\varepsilon^{n-1} \|a_1\|}{\sqrt{m}^{n-1} \prod_{i=2}^n \left(\sqrt{i} \|A_{1:m,1:i}\|_{\max} \right)}.
\end{aligned}$$

□

After these preparations we are now able to state the BB ABM algorithm and to prove some of its properties.

Algorithm 25: BB ABM Algorithm

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, a small number $\varepsilon \geq 0$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}$-border basis G (see Theorem 4.5.3 for details)

```
1  d := 1, O := [1], G := [], A := (1, ..., 1)^tr ∈ Mat_{s,1}(C);
2  L := [t_1, ..., t_ℓ] := all terms of degree 1 ordered decreasingly w.r.t. σ;
3  repeat
4      for i := 1 to ℓ do
5          m := |O|;
6          b := eval_X(t_i);
7          s := (s_m, s_{m-1}, ..., s_1) := solution of the least squares problem min_s ||As − b||
           (compare algorithms in Section 2.11);
8          if ||As − b|| ≤ ε then
               // We assume that O = [o_m, ..., o_1]
9              g := t_i − s_m o_m − ... − s_1 o_1;
10             G := concat(G, [g]);
11         else
12             O := concat([t_i], O);
13             A = (b, A);
14         end
15     end
16     d := d + 1;
17     L := [t_1, ..., t_ℓ] := all terms of degree d in ∂O ordered decreasingly w.r.t. σ;
18 until L = [];
19 return (G, O);
```

Theorem 4.5.3. *This algorithm computes two sets $G = \{g_1, ..., g_\nu\}$ and $\mathcal{O} = \{t_1, ..., t_\mu\}$ which have the following properties:*

1. *For every polynomial g_i in G we have $\|\mathrm{eval}_{\mathbb{X}}(g_i)\| \leq \varepsilon$.*
2. *There is no subset of the elements in $\mathcal{O}$ such that a linear combination of these elements with leading coefficient one (with respect to σ) vanishes ε-approximately when evaluated at the points in $\mathbb{X}$.*
3. *If the set $\mathcal{O}$ is an order ideal, then the set G is an $\mathcal{O}$-border prebasis.*
4. *If the set $\mathcal{O}$ is an order ideal, then the set G is an δ-approximate border basis with $\delta = \frac{\varepsilon}{\zeta}(2\|\mathbb{X}\|_{\max} + \gamma\nu)$, where $\zeta = \frac{\varepsilon^{s-1}}{\sqrt{s}^{s-2}\Pi_{i=2}^{s}(\sqrt{i}\|\mathbb{X}\|_{\max}^{i-1})}$, and $\gamma = \frac{\sqrt{s}^{s-1}\Pi_{i=2}^{s}(\sqrt{i}\|\mathbb{X}\|_{\max}^{s+i-1})}{\varepsilon^{s-1}}$.*

Proof. First of all we prove termination of the algorithm. In line 7 a inhomogeneous least squares problem of the form $\min_s \|As - b\|$ is solved. This guarantees together with the check in line 8 that only linearly independent columns are prepended to A in line 13. This means that the rank

of A will grow by one whenever a new column is prepended. So the rank of A can grow at most up to s. When we arrive at this situation the equation system $Ax = b$ has at least one solution. This means that no new elements will go into $\mathcal{O}$ and thus the program will eventually stop in line 18 because L will be empty.

Claim 1 follows directly from how the polynomials are constructed. They are only appended to G if the inequality $\|As - b\| \leq \varepsilon$ in line 8 holds.

Next we show claim 2. Note that only one element is prepended to $\mathcal{O}$ at a time in line 12. In line 8 we check if an ε-approximately vanishing relation with leading coefficient one (with respect to σ) exists. Only if this is not the case, the term t_i will be added to $\mathcal{O}$. Looking at subsets $\tilde{\mathcal{O}}$ of $\mathcal{O}$ means that the corresponding evaluation matrix $\tilde{A} = \operatorname{eval}_{\mathbb{X}}(\tilde{\mathcal{O}})$ has even fewer columns and we can be sure that also the minimum of $\|\tilde{A}s - b\|$ will be greater than ε. However, this claim does not need to hold when we allow the leading term to have a coefficient which is not equal to one!

Now let us look at claim 3. If we assume that the set $\mathcal{O}$ is an order ideal, it follows from line 9 that all polynomials in G are of the form $t_i - \sum_{k=1}^{m} c_k o_k$, with $c_k \in \mathbb{C}$, $o_k \in \mathcal{O}$. Additionally, we know from line 2 and line 17 that the t_i are exactly the elements in the border of $\mathcal{O}$. So, for each $t_i \in \partial\mathcal{O}$, we have a corresponding polynomial g_i in G.

To prove claim 4 we use a similar strategy as when proving the bound for the ABM algorithm, compare Theorem 4.3.1. The major difference will be that we have to bound the coefficients of the polynomials in G.

First, let us recall that every $g_i \in G$ is of the form $g_i = b_i + h_i$ with $b_i \in \partial\mathcal{O}$ and $\operatorname{supp}(h_i) \subseteq \mathcal{O}$. The polynomials g_i vanish ε-approximately with respect to $\mathbb{X}$. Let us denote by c the coefficient vector of a polynomial h_i and by A the evaluation matrix of $\operatorname{supp}(h_i)$ with respect to $\mathbb{X}$. Then c is the solution of the inhomogeneous least squares problem $\min_c \|Ac - \operatorname{eval}_{\mathbb{X}}(b_i)\|$. So $c = A^+ \operatorname{eval}_{\mathbb{X}}(b_i)$ where A^+ is the pseudoinverse of A. Thus $\|c\| = \|A^+ \operatorname{eval}_{\mathbb{X}}(b_i)\| \leq \|A^+\| \|\operatorname{eval}_{\mathbb{X}}(b_i)\| = \frac{\|\operatorname{eval}_{\mathbb{X}}(b_i)\|}{\sigma_{\min}(A)}$. With the help of Theorem 4.5.1 we can bound the smallest singular value $\sigma_{\min}$ of A. In order to be able to apply the theorem, we first need to analyse matrix A more closely. By Proposition 2.5.35 we know that with each additional column which we prepend to A, the smallest singular value of the new matrix will be smaller than or equal to the smallest singular value of A alone. So we may assume that $\sigma_{\min}(\operatorname{eval}_{\mathbb{X}}(\mathcal{O})) \leq \sigma_{\min}(A)$ in order to obtain a lower bound for the smallest singular value of A. The set $\mathcal{O}$ contains at most s elements, because only up to $s-1$ new elements can be added to $\mathcal{O}$ in line 12. This means that A can have at most s columns. Using Corollary 4.5.2, $\|a_1\| = \|(1, ..., 1)\| = \sqrt{s}$, and the fact that matrix A contains products up to degree $i-1$ of the coordinates of the input points $\mathbb{X}$, we obtain the bound

$$\zeta := \frac{\varepsilon^{s-1}\sqrt{s}}{\sqrt{s}^{s-1}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{i-1}\right)} = \frac{\varepsilon^{s-1}}{\sqrt{s}^{s-2}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{i-1}\right)} < \sigma_{\min}(A).$$

Thus we conclude that, for $\varepsilon \neq 0$, the inequalities

$$\begin{aligned}
\|c\| &\leq \frac{\|\mathrm{eval}_{\mathbb{X}}(b_i)\|}{\sigma_{\min}(A)} < \|\mathrm{eval}_{\mathbb{X}}(b_i)\| \frac{\sqrt{s}^{s-2}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{i-1}\right)}{\varepsilon^{s-1}} \\
&\leq \sqrt{s}\,\|\mathbb{X}\|_{\max}^{s} \frac{\sqrt{s}^{s-2}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{i-1}\right)}{\varepsilon^{s-1}} \\
&= \|\mathbb{X}\|_{\max}^{s} \frac{\sqrt{s}^{s-1}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{i-1}\right)}{\varepsilon^{s-1}} = \frac{\sqrt{s}^{s-1}\Pi_{i=2}^{s}\left(\sqrt{i}\,\|\mathbb{X}\|_{\max}^{s+i-1}\right)}{\varepsilon^{s-1}} := \gamma
\end{aligned}$$

hold.

Now that we have finished these preparations, we are able to bound the coefficient vectors of the normal remainders of the S-polynomials S_{ij} where t_i, t_j are neighbours. For this purpose we let $M = \mathrm{eval}_{\mathbb{X}}(\mathcal{O})$. We first look at the across-the-street neighbours. Furthermore $S_{ij} = x_k g_i - x_l g_j$ and $S'_{ij} = \mathrm{NR}_{\mathcal{O},G}(S_{ij}) = x_k g_i - x_l g_j - \sum_\nu c_\nu g_\nu$ where the c_ν are some coefficients of the polynomials h_i. Note that $\mathrm{supp}(S'_{ij}) \subseteq \mathcal{O}$. First, let us consider the case $\varepsilon = 0$. In this situation we know that $\left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| = 0$ because $\|\mathrm{eval}_{\mathbb{X}}(g_i)\| = 0$ for all $g_i \in G$. Because the matrix M only has a trivial kernel, $\left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| = 0$ implies that $\|S'_{ij}\| = 0$ and therefore $S'_{ij} = 0$ must be satisfied. Next, we consider the case $\varepsilon \neq 0$. In this situation the inequality $|c_v| \leq \|c\| < \gamma$ holds. We now conclude that

$$\begin{aligned}
\left\|S'_{ij}\right\| &\leq \left\|M^+\right\| \left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| < \frac{1}{\zeta}\left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| \\
&\leq \frac{1}{\zeta}\left(\|\mathrm{eval}_{\mathbb{X}}(x_k g_i)\| + \|\mathrm{eval}_{\mathbb{X}}(x_l g_j)\| + \left\|\mathrm{eval}_{\mathbb{X}}\left(\sum_\nu c_\nu g_\nu\right)\right\|\right) \\
&\leq \frac{1}{\zeta}\left(\|\mathrm{eval}_{\mathbb{X}}(g_i)\,\|\mathbb{X}\|_{\max}\| + \|\mathrm{eval}_{\mathbb{X}}(g_j)\,\|\mathbb{X}\|_{\max}\| + \left\|\mathrm{eval}_{\mathbb{X}}\left(\sum_\nu c_\nu g_\nu\right)\right\|\right) \\
&< \frac{1}{\zeta}\left(2\,\|\mathbb{X}\|_{\max}\varepsilon + \gamma\sum_\nu \mathrm{eval}_{\mathbb{X}}(g_\nu)\right) \leq \frac{\varepsilon}{\zeta}\left(2\,\|\mathbb{X}\|_{\max} + \gamma\nu\right).
\end{aligned}$$

Finally, we analyse the next-door neighbours. The error bound is derived analogously to the case of across-the-street neighbours. We have $S_{ij} = g_i - x_k g_j$ and $S'_{ij} = \mathrm{NR}_{\mathcal{O},G}(S_{ij}) = g_i - x_k g_j - \sum_\nu c_\nu g_\nu$ where the c_ν are again some coefficients of the polynomials h_i. In case $\varepsilon = 0$ the same arguments apply as for the across-the-street neighbours. So we know that $\|S'_{ij}\| = 0$ and consequently $S'_{ij} = 0$ and we are thus dealing with an exact border basis as the S-polynomials of all neighbours reduce to 0. In the case $\varepsilon \neq 0$, we obtain the inequalities

$$\begin{aligned}
\left\|S'_{ij}\right\| &\leq \left\|M^+\right\| \left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| < \frac{1}{\zeta}\left\|\mathrm{eval}_{\mathbb{X}}(S'_{ij})\right\| \\
&\leq \frac{1}{\zeta}\left(\|\mathrm{eval}_{\mathbb{X}}(\tilde{g}_i)\| + \|\mathrm{eval}_{\mathbb{X}}(\tilde{g}_j)\,\|\mathbb{X}\|_{\max}\| + \left\|\mathrm{eval}_{\mathbb{X}}\left(\sum_\nu c_\nu \tilde{g}_\nu\right)\right\|\right) \\
&< \ldots \leq \frac{\varepsilon}{\zeta}\left(1 + \|\mathbb{X}\|_{\max} + \gamma\nu\right).
\end{aligned}$$

□

The following example illustrates the individual steps of the BB ABM algorithm.

Example 4.5.4. Let $P = \mathbb{R}[x_1, x_2]$, $\mathbb{X} = [(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)]$, and $\varepsilon = 0.25$ be given and let σ be the `DegRevLex` term ordering.

- $d = 1, \mathcal{O} = [1]$, $G = [\,]$, $M = (1, ..., 1)^{\text{tr}}$, and $L = [x_1, x_2]$
- $A = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0.5 \end{pmatrix}$, solve $\min_x \|Ax - b\|$, $x \approx (0.499)$, $\|Ax - b\| \approx 1$, $\mathcal{O} = [x_2, 1]$
- $A = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0.5 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0.5 \end{pmatrix}$, solve $\min_x \|Ax - b\|$, $x \approx (0, 0.499)$, $\|Ax - b\| \approx 1$, $\mathcal{O} = [x_2, x_1, 1]$
- $d = 2$, $\partial\mathcal{O} = \{x_1^2, x_1x_2, x_2^2\}$, and $L = [x_1^2, x_1x_2, x_2^2]$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0.25 \end{pmatrix}$, solve $\min_x \|Ax - b\|$, $x \approx (0, 1, -0.049)$, $\|Ax - b\| \approx 0.223$
- $g_1 = x_2^2 - x_1 + 0.049$, $G = [g_1]$, $\mathcal{O} = [x_2, x_1, 1]$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0.5 & 0.5 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0.25 \end{pmatrix}$, solve $\min_x \|Ax - b\|$, $x \approx (0.5, 0.5, -0.249)$, $\|Ax - b\| \approx 0.5$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0.25 \end{pmatrix}$, solve $\min_x \|Ax - b\|$, $x \approx (0, 1, 0, -0.05)$, $\|Ax - b\| \approx 0.223$
- $g_2 = x_1^2 - x_2 + 0.05$, $G = [g_1, g_2]$, $\mathcal{O} = [x_1x_2, x_2, x_1, 1]$
- $d = 3$, $\partial\mathcal{O} = \{x_1^2x_2, x_1x_2^2, x_1^2, x_2^2\}$, and $L = [x_1^2x_2, x_1x_2^2]$

- $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0.125 \end{pmatrix}$,
  solve $\min_x \|Ax - b\|$, $x \approx (0.999, 0, 0, -0.025)$, $\|Ax - b\| \approx 0.111$
- $g_3 = x_1^2 x_2 - x_1 x_2 + 0.025$, $G = [g_1, g_2, g_3]$, $\mathcal{O} = [x_1 x_2, x_2, x_1, 1]$
- $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0.25 & 0.5 & 0.5 & 1 \end{pmatrix}$,$b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0.125 \end{pmatrix}$,
  solve $\min_x \|Ax - b\|$, $x \approx (0.999, 0, 0, -0.025)$, $\|Ax - b\| \approx 0.111$
- $g_4 = x_1 x_2^2 - x_1 x_2 + 0.025$, $G = [g_1, g_2, g_3, g_4]$, $\mathcal{O} = [x_1 x_2, x_2, x_1, 1]$
- $d = 4$, $\partial\mathcal{O} = \{x_1^2 x_2, x_1 x_2^2, x_1^2, x_2^2\}$, and $L = \emptyset$
- $G = [g_1, g_2, g_3, g_4]$, $\mathcal{O} = [x_1 x_2, x_2, x_1, 1]$

Remark 4.5.5. Just like the extended ABM algorithm the BB ABM algorithm can achieve greater numerical stability if the least squares solution in line 7 is computed with the help of the SVD. Details can be found in Subsection 4.4.3.

4.5.1 Runtime Complexity of the BB ABM Algorithm

Proposition 4.5.6. *The runtime of the BB ABM algorithm is cubic in the number of input points if the QR decomposition/SVD which is used to solve the least squares problem is updated during the computation.*

Proof. The algorithm shares the same basic structure with the ABM algorithm, the major change involves the computation of the solution of the inhomogeneous least squares problem in line 7. This task can be solved efficiently in $O\left(2mn^2 - \frac{2}{3}n^3\right)$ using the QR decomposition of $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$, compare Proposition 2.11.3 and Remark 2.11.4 for details. This process can again be accelerated if the QR decomposition is not recomputed in each step but updated accordingly. Details on this subject can, for instance, be found in [6, Section 12.5.2]. Via a sequence of Givens rotations it is possible to achieve the task in $O(mn)$. After we have computed the QR decomposition we still need to solve the least squares problem in $O\left(mn + \frac{1}{2}n(n+1)\right)$. Assuming that we are in the same setting as when investigating the runtime of the ABM algorithm, i.e. $\varepsilon = 0$, we can take for granted that the QR decomposition is at most updated s times, as in this case the matrix A will have full rank s and no new elements will go into $\mathcal{O}$. For more details compare Subsection 4.4.2. □

4.5.2 Shortcomings of the BB ABM Algorithm

- Similarly like the AVI and ABM algorithm, in some cases the BB ABM algorithm may return a set $\mathcal{O}$ which is not an order ideal of terms. This means that for at least one term not all divisors are included in the set $\mathcal{O}$. If in a subsequent computation it is imperative that the output of the BB ABM algorithm is a proper approximate border basis, this can be cured by checking during the computation if adding a new term t to $\mathcal{O}$ would violate the order ideal property. If so, we need to form a polynomial from t and the other terms which are currently in $\mathcal{O}$ and add the resulting polynomial to the set G. The coefficients of this polynomial are again computed via the solution of the homogeneous least squares problem (compare Theorem 2.13.6). Of course this will weaken the bounds concerning ε and δ. The consequences which also apply in this case have already been analysed in Subsection 4.3.4.
- Even though the BB ABM allows to better bound δ compared to the AVI or ABM algorithm (see Theorem 4.2.2, Theorem 4.3.1 and Remark 4.3.3), the worst case estimates are still impractical for some applications which require δ to be small.

4.5.3 Implementation in ApCoCoA

The BB ABM algorithm has been implemented by the author in the ApCoCoA C++ library ([20]). Currently only a real (double) version of the algorithm is implemented.

For the computation of matrix-matrix and matrix-vector products the BLAS software interface was used. This allows machine dependent optimisations by simply linking to an optimised implementation which makes use of the target computer architecture or additional equipment like graphics cards (e.g. CUDA or OpenCL).

For the solution of the inhomogeneous least squares problem, the LAPACK software library [17] is used. The command used for the solution of the least squares problem is *dgels*, which is based on the QR decomposition.

A detailed description of the parameters of the ApCoCoA implementation of the BB ABM algorithm can be found in Appendix 8.1.

4.6 Practical Considerations and Extensions

In a practical implementation of these algorithms, it is also important to take care of terms in a polynomial which, after evaluation on the input data sets, do not contribute significantly to the total evaluation of the polynomial. These terms have in general no reasonable physical meaning and should therefore be removed to facilitate interpretation. The fastest but also the most problematic option to achieve this goal is to remove terms whose coefficients have a smaller absolute value than a given number $\tau \in \mathbb{R}^+$. The obvious shortcoming is that the coefficient of the term is only partly related to the actual average value that it contributes to the total evaluation of

the polynomial. Consequently, it is a better strategy to look at the average evaluation of the monomial in question with respect to the points in $\mathbb{X}$.

So to achieve our goal, the following steps could be executed after the ABM, the extended ABM, or the BB ABM algorithm have terminated:

1. Let $\tau \in \mathbb{R}^+$ be a small positive number. For every monomial m of every polynomial p_j in G calculate $v = \frac{1}{s} \left\| \text{eval}_{\mathbb{X}}(m) \right\|$. If $v \leq \tau$ delete m from p_j.
2. Reproject every polynomial in G that was modified with respect to $\mathbb{X}$. "Reprojecting" has to be interpreted in the context of the actual algorithm that is used.

4.6.1 Subideal Variants

All algorithms discussed earlier, namely the ABM, extended ABM, and BB ABM algorithm, can be combined with the subideal border basis concept developed in [32] by Kreuzer and Poulisse. The main practical advantage of the "subideal approach" is that it is possible to incorporate additional a priori knowledge. In the setting of exact border bases, this translates mathematically to an intersection of the ideal of points of $\mathbb{X}$ and a given ideal $J \subseteq \mathbb{C}[x_1, ..., x_n]$. How this maps to approximate border bases is described in detail in [32]. We start by repeating the most important definitions.

Definition 4.6.1. Let $I \subset P = K[x_1, ..., x_n]$ be a zero-dimensional ideal, let $J = \langle f_1, ..., f_m \rangle$ be a polynomial ideal of P, where $F = \{f_1, ..., f_m\} \subset P \backslash \{0\}$, and let $\mathcal{O}$ be an order ideal of terms in $\mathbb{T}^n$ whose residue classes form a vector space basis of P/I.

1. For $1 \leq i \leq m$, let $\mathcal{O}_i \subseteq \mathcal{O}$ be an order ideal. Then the set $\mathcal{O}_F = \mathcal{O}_1 \cdot f_1 \cup ... \cup \mathcal{O}_m \cdot f_m$ is called an **F-order ideal**. Its elements tf_i with $t \in \mathcal{O}_i$ are called **F-terms**.
2. If $\mathcal{O}_F$ is an F-order ideal whose residue classes form a vector space basis of $J/(I \cap J)$, we say that the ideal I has an **$\mathcal{O}_F$-subideal border basis**.

Definition 4.6.2. Let $F = \{f_1, ..., f_m\} \subset P \backslash \{0\}$, let J be the ideal generated by F, and let $\mathcal{O}_F$ be an F-order ideal. We write $\mathcal{O}_F = \{t_1 f_{\alpha_1}, ..., t_\mu f_{\alpha_\mu}\}$ with $\alpha_i \in \{1, ..., m\}$ and $t_i \in \mathcal{O}_{\alpha_i}$.

1. The set of all polynomials $x_i t_j f_{\alpha_j}$ such that $i \in \{1, ..., n\}$, $j \in \{1, ..., \mu\}$, and $x_i t_j f_{\alpha_j} \notin \mathcal{O}_{\alpha_j} f_{\alpha_j}$ is called the **border** of $\mathcal{O}_F$ and denoted by $\partial \mathcal{O}_F$.
2. Let $\partial \mathcal{O}_F = \{b_1 f_{\beta_1}, ..., b_\nu f_{\beta_\nu}\}$. A tuple of polynomials $G = (g_1, ..., g_\nu)$ is called an **$\mathcal{O}_F$-subideal border prebasis** if $g_j = b_j f_{\beta_j} - \sum_{i=1}^{\mu} c_{ij} t_i f_{\alpha_i}$ with $c_{ij}, ..., c_{\mu j} \in K$ for $1 \leq j \leq \nu$.
3. An $\mathcal{O}_F$-subideal border prebasis is called an **$\mathcal{O}_F$-subideal border basis** of an ideal I if the elements of G are contained in I and the residue classes of the elements of $\mathcal{O}_F$ form a K-vector space basis of $J/(I \cap J)$.

Definition 4.6.3. Let $\mathcal{O}_F = \{t_1 f_{\alpha_1}, ..., t_\mu f_{\alpha_\mu}\}$ be an F-order ideal, let $\partial \mathcal{O}_F = \{b_1 f_{\beta_1}, ..., b_\nu f_{\beta_\nu}\}$ be its border, and let $G = (g_1, ..., g_\nu)$ be an $\mathcal{O}_F$-subideal border prebasis. This means that every g_j is of the form $g_j = b_j f_{\beta_j} - \sum_{i=1}^{\mu} c_{ij} t_i f_{\alpha_i}$ with $c_{ij} \in \mathbb{C}$.

For every pair (i,j) such that $b_i f_{\beta_i}$, $b_j f_{\beta_j}$ are neighbours in $\partial\mathcal{O}_F$, i.e. $\beta_i = \beta_j$ and b_i, b_j are neighbours in the usual sense, we compute the normal remainder $S'_{ij} = \mathrm{NR}_{\mathcal{O}_F,G}(S_{ij})$ of the S-polynomial of g_i and g_j with respect to G. We say that G is an **ε-approximate $\mathcal{O}_F$-subideal border basis** if $\|S'_{ij}\| \leq \varepsilon$ holds for all pairs (i,j).

Definition 4.6.4. A polynomial f is called **1-unitary** if the 1-norm of its coefficient vector equals one: $\|f\|_1 = 1$.

Now we adapt the ABM algorithm to obtain a subideal variant of it. All other algorithms can be modified in a similar fashion.

Algorithm 26: Subideal ABM Algorithm

Input: A tuple of affine points $\mathbb{X} = [p_1, ..., p_s]$ with $p_i \in \mathbb{C}^n$, a polynomial ring $P = \mathbb{C}[x_1, ..., x_n]$, a set of $\|\cdot\|_1$-unitary polynomials $F = \{f_1, ..., f_m\} \subset P \setminus \{0\}$ generating an ideal $J = \langle F \rangle$, a small number $\varepsilon \geq 0$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An approximate $\mathcal{O}_F$-subideal border basis G (see [32, Algorithm 5.4] and Theorem 4.3.1 for details)

$d := \min(\deg(f_1), ..., \deg(f_m))$, $\mathcal{O}_F := [\,]$, $G := [\,]$, $M \in \mathrm{Mat}_{s,0}(\mathbb{C})$;
$L := [t_1 f_{\alpha_1}, ..., t_\ell f_{\alpha_\ell}] :=$ all terms of degree d in $F \cup \partial\mathcal{O}_F$ ordered decreasingly w.r.t. σ;
repeat
  for *i := 1* **to** ℓ **do**
    $A := (\mathrm{eval}_{\mathbb{X}}(t_i f_{\alpha_i}), M)$;
    $B := A^* A$;
    $\gamma :=$ smallest eigenvalue of B;
    if $\sqrt{\gamma} \leq \varepsilon$ **then**
      $m' := |\mathcal{O}_F|$;
      `// We assume that` $\mathcal{O}_F = [o_{F,m'}, ..., o_{F,1}]$
      $s := (s_{m'+1}, s_{m'}, ..., s_1) :=$ norm one eigenvector of B w.r.t. to γ;
      $g := s_{m'+1} t_i f_{\alpha_i} + s_{m'} o_{F,m'} + \ldots + s_1 o_{F,1}$;
      $G := \mathrm{concat}(G, [g])$;
    else
      $\mathcal{O}_F := \mathrm{concat}([t_i f_{\alpha_i}], \mathcal{O}_F)$;
      $M := A$;
    end
  end
  $d := d + 1$;
  $L := [t_1 f_{\alpha_1}, ..., t_\ell f_{\alpha_\ell}] :=$ all terms of degree d in $F \cup \partial\mathcal{O}_F$ ordered decreasingly w.r.t. σ;
until $L := [\,]$ **and** $d \geq \max(\deg(f_1), ..., \deg(f_m))$;
return $(G, \mathcal{O}_F)$;

We do not discuss or prove the properties of this algorithm. They can be derived by combining the ideas of the proof of the ABM algorithm (22) and the proofs presented in [32].

For the AVI, ABM, extended ABM and BB ABM algorithm subideal variants have been implemented by the author in the ApCoCoA library ([20]). They can be called by putting the prefix Sub in front of the non-subideal versions, e.g. SubABM. Additionally, the given ideal F has to be provided as a parameter.

Remark 4.6.5. If we let $F = \{1\}$, the subideal version is identical to the ordinary version of the algorithms.

Finally, we present a small example which demonstrates the individual steps of the Subideal ABM algorithm.

Example 4.6.6. Let $P = \mathbb{R}[x_1, x_2]$, $\mathbb{X} = \{(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)\}$, and $\varepsilon = 0.2$ be given and let σ be the `DegRevLex` term ordering. Furthermore, we consider the ideal $J = \langle f_1 \rangle$ with $f_1 = x_1$.

- $d = \min(\deg(f_1)) = 1$, $\mathcal{O}_F = [\,]$, $G = [\,]$, and $M \in \mathrm{Mat}_{s,0}(\mathbb{C})$
- $L = [f_1] = [x_1]$, and $i = 1$
- $A = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0.5 \end{pmatrix}$, solve $\min_x \|Ax\|$ subject to $\|x\| = 1$, $A^*A = \begin{pmatrix} 2.25 \end{pmatrix}$, $\sqrt{\gamma} = 1.5 > \varepsilon$,
  $\mathcal{O}_F = [x_1]$
- $d = 2$, $F \cup \partial\mathcal{O}_F = \{x_1, x_1x_2, x_1^2\}$, $L = [x_1x_2, x_1^2]$, and $i = 1$
- $A = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0.25 & 0.5 \end{pmatrix}$, $A^*A = \begin{pmatrix} 1.0625 & 1.125 \\ 1.125 & 2.25 \end{pmatrix}$, $\sqrt{\gamma} \approx 0.617 > \varepsilon$,
  $\mathcal{O}_F = [x_1, x_1x_2]$, and $i = 2$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0.25 & 0.25 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 2.0625 & 1.0625 & 2.125 \\ 1.0625 & 1.0625 & 1.125 \\ 2.125 & 1.125 & 2.25 \end{pmatrix}$, $\sqrt{\gamma} \approx 0.170 < \varepsilon$,
  $g_1 \approx 0.717x_1^2 + 0.021x_1x_2 - 0.696x_1$, $G = \{g_1\}$
- $d = 3$, $F \cup \partial\mathcal{O}_F = \{x_1, x_1^2x_2, x_1x_2^2\}$, $L = [x_1^2x_2, x_1x_2^2]$, and $i = 1$
- $A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0.125 & 0.25 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 1.01562 & 1.03125 & 1.0625 \\ 1.03125 & 1.0625 & 1.125 \\ 1.0625 & 1.125 & 2.25 \end{pmatrix}$, $\sqrt{\gamma} \approx 0.083$,
  $g_2 \approx -0.702x_1^2x_2 + 0.712x_1x_2 - 0.025x_1$, $G = \{g_1, g_2\}$, $i = 2$

- $A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0.125 & 0.25 & 1 \end{pmatrix}$, $A^*A = \begin{pmatrix} 1.01562 & 1.03125 & 1.0625 \\ 1.03125 & 1.0625 & 1.125 \\ 1.0625 & 1.125 & 2.25 \end{pmatrix}$, $\sqrt{\gamma} \approx 0.083$,
  $g_3 \approx -0.702x_1x_2^2 + 0.712x_1x_2 - 0.025x_1$, $G = \{g_1, g_2, g_3\}$, $i = 3$
- $d = 4$, $L = [\,]$
- The algorithm returns the sets $G = \{g_1, g_2, g_3\}$ and $\mathcal{O}_F = \{x_1, x_1x_2\}$.

4.7 Comparison with other Approaches

4.7.1 Approximate H-Bases

The concept of H-Bases was originally introduced by F. S. Macaulay in [37, page 39]. This is the reason why some authors, for instance Kreuzer and Robbiano in [46], call H-Bases also Macaulay bases.
Let K be a field of characteristic zero. Given a finite set of points $\mathbb{X} \subset K^n$, it is possible, similarly like for Gröbner and border bases, to construct H-bases efficiently with a variant of the Buchberger-Möller algorithm. Recently it has been shown by Sauer in [41] that H-bases can also be applied to numerical problems. For that purpose he generalised the concept of H-bases to approximate H-bases.

Following Möller and Sauer in [38] and Kreuzer and Robbiano in [45] we introduce the following definitions.

Let $n \in \mathbb{N}$ and let $P = K[x_1, ..., x_n]$ be the polynomial ring over K in n indeterminates.

Definition 4.7.1. [H-Basis]

A finite set $G = \{g_1, ..., g_m\} \subset P \setminus \{0\}$ of polynomials is called an **H-basis** of the ideal $I = \langle g_1, ..., g_m \rangle$ if for all $0 \neq p \in I$ there exist $h_1, ..., h_m \in P$ such that

$$p = \sum_{i=1}^{m} h_i g_i \quad \text{and} \quad \deg(h_i) + \deg(g_i) \leq p \text{ for all } 1 \leq i \leq m.$$

Definition 4.7.2. [Homogeneous Polynomials]

Let

$$P_d^0 = \{f \in P \mid \deg(t) = d \quad \text{for all} \quad t \in \operatorname{supp}(f)\}.$$

For $d \geq 0$, we call the elements of P_d^0 the **homogeneous polynomials** of degree d. Additionally, we let $P_d = \bigoplus_{j=0}^{d} P_j^0$ and $P^0 = \bigcup_{j \in \mathbb{N}_0} P_j^0$.
We define the map $\Lambda : P \to P^0$ which associates with each $f \in P$ its **homogeneous leading term** as

$$\Lambda(f)(x) = \sum_{|\alpha| = \deg(f)} f_\alpha x^\alpha \quad \text{where} \quad f(x) = \sum_{|a| \leq \deg(f)} f_\alpha x^\alpha.$$

Definition 4.7.3. Let $F \subset P$ be a finite set of polynomials in P. Then we define $\deg(F) = \max_{f \in G}(\deg(f))$. Let $\lambda \in [0,1]$, let $\mathbb{X} \subset K^n$ be a finite set of points, let $\mathcal{I}(\mathbb{X})$ be the associated vanishing ideal, and let $P_{\mathbb{X}} = P/\mathcal{I}(\mathbb{X})$. Then we can consider all elements of $\mathcal{I}(\mathbb{X})$ of degree at most $\lambda \cdot \deg(P_{\mathbb{X}})$ and the ideal generated by these polynomials. For this purpose let

$$\mathcal{I}_\lambda(\mathbb{X}) = \langle f \in \mathcal{I}(\mathbb{X}) \mid \deg(f) \leq \lambda(\deg(P_{\mathbb{X}}) + 1) \rangle .$$

The points $\mathbb{X}$ are said to lie on a **variety of relative degree** λ if $\mathcal{I}_\lambda(\mathbb{X}) \neq \emptyset$.

For the sake of simplicity, from now on let $K = \mathbb{R}$.

Definition 4.7.4. Let $\langle \cdot, \cdot \rangle : P \times P \to \mathbb{R}$ be an inner product defined on the polynomials in P. Furthermore let $f \in P_j^0$ and $g \in P_k^0$. We say that $\langle \cdot, \cdot \rangle$ **separates degrees** if $j \neq k$ implies that $\langle f, g \rangle = 0$. For instance, the **Macaulay inner product** of the coefficients given by

$$\langle f, g \rangle = \sum_{\alpha \in \mathbb{N}_0^d} f_\alpha g_\alpha \quad \text{where} \quad f(x) = \sum_{\alpha \in \mathbb{N}_0^d} f_\alpha x^\alpha \quad \text{and} \quad g(x) = \sum_{\alpha \in \mathbb{N}_0^d} g_\alpha x^\alpha$$

separates degrees.

For $F \subset P$ and $k \in \mathbb{N}_0$ we define the vector spaces

$$V_k^0(F) = \left\{ \sum_{f \in F} g_f \Lambda(f) \,|\, g_f \in P_{k - \deg f}^0, f \in F \right\} \subseteq P_k^0$$

and their orthogonal complements with respect to $\langle \cdot, \cdot \rangle$ by

$$W_k^0(F) = P_k^0 \ominus V_k^0(F),$$

such that $P_k^0 = V_k^0(F) \oplus W_k^0(F)$ and $\langle V_k^0(F), W_k^0(F) \rangle = 0$. Consequently we define $V_k(F) = \oplus_{j=0}^k V_k^0(F)$ and $W_k(F) = \oplus_{j=0}^k W_k^0(F)$.

As pointed out in [38] it is possible to compute H-bases for a given finite set $F \subset P$ in finitely many steps. Moreover, it is possible to construct algorithmically an H-basis for $\mathcal{I}_\lambda(\mathbb{X})$. Further details about H-bases and how they can be constructed can, for instance, be found in [38], [39] and [40].

Definition 4.7.5. Let $\varepsilon > 0$, let $\mathbb{X} \subset \mathbb{R}^n$, and let $1 \leq p \leq \infty$. Then the **p-approximate ideal of accuracy** ε with respect to $\mathbb{X}$ is defined as

$$\mathcal{I}_{p,\varepsilon}(\mathbb{X}) = \left\{ f \in P \,\middle|\, \frac{\|f(\mathbb{X})\|_p}{\|f\|} \leq \varepsilon \right\},$$

where $\|f\|$ is the Euclidean norm of the coefficient vector of f.

Sauer has pointed out in [41] that these approximate ideals have no particularly nice structure. For instance, they are no convex sets.

Remark 4.7.6. Please note that if we let $p = 2$, then all unitary polynomials (see Definition 4.1.2) in $\mathcal{I}_{p,\varepsilon}(\mathbb{X})$ vanish ε-approximately with respect to $\mathbb{X}$.

Definition 4.7.7. [Approximate H-Basis]

Let $\varepsilon > 0$, let $\mathbb{X} \subset \mathbb{R}^n$, and let $1 \leq p \leq \infty$. A finite set $H \subset P$ is called a **p-approximate H-basis of accuracy** ε for $\mathcal{I}(\mathbb{X})$ if $H \subset \mathcal{I}_{p,\varepsilon}(\mathbb{X})$ and if it is an H-basis.

Definition 4.7.8. Let $F = \{f_1, ..., f_m\} \subset P$ and let $\mathbb{X} = \{p_1, ..., p_s\} \subset \mathbb{R}^n$. We abbreviate the evaluation (or **Vandermonde**) matrix of F with respect to $\mathbb{X}$ by

$$F(\mathbb{X}) = \mathrm{eval}_{\mathbb{X}}(f_1, ..., f_m) \in \mathrm{Mat}_{s,m}(\mathbb{R}).$$

In [41] Sauer has proposed a special version of the QR decomposition with column pivoting that plays a crucial role in his approximate H-basis algorithm. We will not give details here, those can be found in [41, Section 5]. However, we state the properties of the computed QR decomposition as those are important to understand the approximate H-basis algorithm.

Proposition 4.7.9. *Let $F_d \subset P_d \backslash P_{d-1}$ be a set of polynomials that are orthonormal with respect to the Macaulay inner product. Furthermore, let $\mathbb{X}_d \subseteq \mathbb{X} \subset \mathbb{R}^n$, and let $\varepsilon \geq 0$. Let $F_d(\mathbb{X}_d)$ be the evaluation matrix of F_d with respect to $\mathbb{X}_d$. With the help of the QR decomposition algorithm that is presented in [41, Section 5], it is possible to compute matrices Q_d and P_d such that we obtain*

$$Q_d^{\mathrm{tr}} F_d^{\mathrm{tr}}(\mathbb{X}_d) P_d^{\mathrm{tr}} = \begin{bmatrix} R_d & A_d \\ 0 & B_d \end{bmatrix}$$

where $|r_{11}| \geq ... \geq |r_{kk}| > \varepsilon$ and $\|B\|_{\max} \leq \varepsilon$. Then there exist two sets F_d^+ and F_d^0 of normalised, linearly independent polynomials in the vector space spanned by F_d, and a subset $\mathbb{X}_d^+ \subseteq \mathbb{X}_d$ with $|\mathbb{X}_d^+| = |F_d^+|$ such that

$$F_d^+(\mathbb{X}_d^+) = R_d \quad \text{and} \quad F_d^0(\mathbb{X}_d^+) = 0$$

while $F_d^0 \subset \mathcal{I}_{\infty,\varepsilon}(\mathbb{X})$. We obtain F_d^+ by multiplying the first k rows of Q_d^{tr} with F_d and F_d^0 by multiplying the remaining rows of Q_d^{tr} with F_d. Furthermore, we obtain $\mathbb{X}_d^+$ by taking the first k points from $P_d^{\mathrm{tr}} \mathbb{X}_d$.

Proof. See [41, pages 308-310]. □

The following algorithm was proposed by Sauer in [41].

Algorithm 27: Approximate H-basis Algorithm

Input: A set of affine points $\mathbb{X} = \{p_1, ..., p_s\}$ with $p_i \in \mathbb{R}^n$, a small number $\varepsilon \geq 0$

Output: An approximate H-basis for $\mathcal{I}(\mathbb{X})$

```
d := 0;
F_0^0 := ∅;
while true do
    G := a basis of the homogeneous vector space W_d(F_0^0 ∪ ... ∪ F_{d-1}^0);
    foreach g ∈ G do
        for k := 0 to d − 1 do
            g := g − g(X_k^+)^tr R_k^{-1} F_k^+ ;
        end
    end
    F_d := an orthonormal basis w.r.t. ⟨·,·⟩ for the span of G;
    X_d := X \ (X_0^+ ∪ ... ∪ X_{d-1}^+);
    if X_d = ∅ then return (F_0^+, ..., F_{d-1}^+, F_0^0, ..., F_{d-1}^0) ;
    Compute F_d^+, F_d^0, R_d, and X_d^+ according to Proposition 4.7.9 using F_d(X_d) and ε as
    input;
    if F_d^+ = ∅ then return (F_0^+, ..., F_{d-1}^+, F_0^0, ..., F_{d-1}^0) ;
    d := d + 1;
end
```

Theorem 4.7.10. *Let $\mathbb{X} \subset \mathbb{R}^n$ be a finite set of real points and let $\varepsilon \geq 0$.*

1. *The above procedure terminates after a finite number of steps.*
2. *If $\varepsilon = 0$, then the polynomials F_k^+, $k \leq n$, span a degree reducing interpolation space for $\mathbb{X}$ (see [42]). Furthermore, the polynomials F_k^0, $k \leq n$, form an H-basis for $\mathcal{I}(\mathbb{X})$.*
3. *Suppose that the algorithm terminates in degree d and let $f \in F_0$. Then there exists $\tilde{f} \in \mathcal{I}(\mathbb{X}^+)$, where $\mathbb{X}^+ = \mathbb{X}_0^+ \cup ... \cup \mathbb{X}_d^+$, such that*

$$\left\| f - \tilde{f} \right\| \leq \sum_{k=\deg(f)}^{d} 2^{|F_k^+|} \max_j \left\| (R_k)_{jj}^{-1} \right\| .$$

Proof. See [41, Theorem 4 and Theorem 5]. □

Remark 4.7.11. As pointed out for instance in [40, Section 4] H-bases for ideals of points $\mathcal{I}(\mathbb{X})$ are also rather stable with respect to perturbations in the input data set $\mathbb{X}$ (compare Subsection 3.2). This is due to the fact that similarly like border bases, they are parametrised by more parameters than Gröbner bases.

In order to better illustrate the properties of approximate H-bases, of the approximate H-basis algorithm and to be able to relate them to approximate border bases and the ABM algorithm, we provide a few simple examples. All computations with the approximate H-basis algorithm

where performed by Johannes Czekansky form Giessen University ([43]). In each example we start with noisy points that either lie approximately on a line in $\mathbb{R}^3$, on a surface in $\mathbb{R}^3$, or on a parabola in $\mathbb{R}^2$. The parameter ε is chosen for all algorithms in such a way that the underlying simple geometric relations are uncovered. Here, we only compare the low degree equations and not the higher degree ones.

Example 4.7.12. Let $P = \mathbb{R}[x_1, x_2, x_3]$ and let

$$\begin{aligned}\mathbb{X} = \{&(1.005, 0.959, 1.046), \quad (1.979, 1.967, 2.035), \quad (2.962, 3.002, 3.005),\\ &(4.024, 4.008, 3.950), \quad (5.014, 4.986, 5.003), \quad (5.978, 6.004, 5.981),\\ &(6.996, 7.023, 7.026), \quad (8.031, 7.999, 7.966), \quad (9.005, 8.974, 9.048),\\ &(10.022, 9.979, 9.951), \quad (11.005, 11.039, 10.986), \quad (12.023, 12.005, 11.954),\\ &(12.971, 13.044, 12.981), \quad (13.967, 13.963, 14.038), \quad (14.965, 15.009, 15.048),\\ &(15.978, 16.013, 15.984), \quad (16.974, 16.982, 16.987), \quad (18.030, 17.964, 18.027),\\ &(19.010, 19.043, 18.995), \quad (19.984, 19.950, 19.995)\}\end{aligned}$$

be a set of points that lie approximately on a line in $\mathbb{R}^3$. First, we apply the ABM algorithm (22) with $\varepsilon = 0.16$ and the `DegRevLex` term ordering. The first two polynomials in the approximate border basis are of degree 1 and given by

$$\begin{aligned} g_1 &\approx x_2 - x_3 + 0.012,\\ g_2 &\approx x_1 - x_3 + 0.004.\end{aligned}$$

These equations encode that the points lie approximately on a line. For $\varepsilon = 0.15$ the approximate H-basis algorithm (27) returns the following polynomials in degree 1:

$$\begin{aligned} \tilde{g}_1 &\approx -0.059 - 0.267x_1 - 0.533x_2 + 0.801x_3,\\ \tilde{g}_2 &\approx 0.022 - 0.772x_1 + 0.617x_2 + 0.155x_3.\end{aligned}$$

Also these equations encode that the points lie approximately on a line. For instance, if we form a linear combinations of $\tilde{g}_1$ and $\tilde{g}_2$ such that we eliminate x_1 and x_2 we obtain the polynomials $x_2 - 1.0021z + 0.0890$ and $x_1 - 1.0015x_3 + 0.0426$. Clearly, those are very similar to the ones that we have obtained with the help of the ABM algorithm. The polynomials found by the ABM algorithm are sparser compared to $\tilde{g}_1$ and $\tilde{g}_2$ because we are not looking for almost vanishing relations in a complete degree but we rather look for almost vanishing relations term by term as dictated by a term ordering on $\mathbb{T}^n$.

Example 4.7.13. Let $P = \mathbb{R}[x_1, x_2, x_3]$ and let

$$\begin{aligned}\mathbb{X} = \{&(0.005, -0.040, 1.646), \quad (-0.020, 0.967, 2.835), \quad (-0.037, 2.002, 4.005),\\ &(0.024, 3.008, 5.150), \quad (1.014, -0.013, 1.203), \quad (0.978, 1.004, 2.381),\\ &(0.996, 2.023, 3.626), \quad (1.031, 2.999, 4.766), \quad (2.005, -0.025, 0.848),\\ &(2.022, 0.979, 1.951), \quad (2.005, 2.039, 3.186), \quad (2.023, 3.005, 4.354),\\ &(2.971, 0.044, 0.381), \quad (2.967, 0.963, 1.638), \quad (2.965, 2.009, 2.848),\\ &(2.978, 3.013, 3.984)\}\end{aligned}$$

be a set of points that lie approximately on a surface in $\mathbb{R}^3$. For $\varepsilon = 0.08$ and the `DegRevLex` term ordering the ABM algorithm returns the degree 1 equation

$$g_1 \approx x_1 - 2.860x_2 + 2.433x_3 - 4.025.$$

If we normalise g_1 by dividing through the norm of its coefficient vector we obtain

$$\frac{g_1}{\|g_1\|} \approx 0.178x_1 - 0.511x_2 + 0.434x_3 - 0.719.$$

With the help of the approximate H-basis algorithm we obtain

$$\tilde{g}_1 \approx 0.171x_1 - 0.498x_2 + 0.432x_3 - 0.732$$

for $\varepsilon = 0.1$. Note that the computed results are again comparable. The difference in the coefficient vectors can be explained with the fact that each algorithm minimises a different norm.

Example 4.7.14. Let $P = \mathbb{R}[x_1, x_2]$ and let

$$\begin{aligned}\mathbb{X} = \{&(-0.942, 19.487), \quad (-0.766, 17.723), \quad (-0.685, 16.090),\\ &(-0.614, 14.385), \quad (-0.492, 12.971), \quad (-0.397, 11.263),\\ &(-0.272, 10.896), \quad (-0.190, 9.560), \quad (-0.073, 8.506),\\ &(-0.011, 8.080), \quad (0.089, 7.457), \quad (0.204, 7.051),\\ &(0.298, 6.946), \quad (0.397, 6.851), \quad (0.467, 7.075),\\ &(0.574, 6.930), \quad (0.702, 7.274), \quad (0.816, 7.990),\\ &(0.939, 8.872), \quad (0.997, 9.633)\}\end{aligned}$$

be a set of points that lie approximately on a parabola in $\mathbb{R}^2$. With the help of the ABM algorithm we compute the degree 2 polynomial

$$g_1 \approx x_1^2 + 0.093x_1x_2 + 0.007x_2^2 - 1.677x_1 - 0.273x_2 + 1.747$$

for $\varepsilon = 0.09$ and the `DegRevLex` term ordering. If normalise g_1 we get

$$\frac{g_1}{\|g_1\|} \approx 0.379x_1^2 + 0.035x_1x_2 + 0.002x_2^2 - 0.636x_1 - 0.103x_2 + 0.662.$$

For $\varepsilon = 0.1$ we obtain with the help of the approximate H-basis algorithm

$$\tilde{g}_1 \approx 0.589x_1^2 + 0.010x_1x_2 - 0.560x_1 - 0.071x_2 + 0.579.$$

Structurally, the polynomials are similar and both results serve their purpose. Once again the difference in the coefficient vectors is related to the different norms that are minimised by the algorithms.

Remark 4.7.15. Please note that Algorithm 27 does not require a term ordering on $\mathbb{T}^n$ as an input parameter. This is a difference to the Buchberger-Möller algorithm for border bases and the derived algorithms like the AVI and ABM algorithm. Clearly, an algorithm is easier to use if the user does not have to choose a term ordering. However, the term ordering allows us to influence

the computed result as it determines in which order the elements in each degree are processed. In this way we can tailor term orderings for specific applications and obtain sparser solutions. It is well-known that also border bases exist which are not induced by a term ordering. However, the border basis transformation algorithm (19), gives us the necessary flexibility to transform a given border basis into a border basis with respect to an order ideal that is not induced by a term ordering.

A major conceptual difference between the approximate H-basis algorithm and for instance the ABM algorithm is that the approximate H-basis algorithm produces a sequence of H-bases $F_0^0, ..., F_{d-1}^k$ where each set of polynomials posses an increasing number of exact zeros (up to numerical accuracy) with respect to subsets $\mathbb{X}_0, ..., \mathbb{X}_{d-1}$ of the input data, such that $\mathbb{X}_0 \subseteq \mathbb{X}_1 \subseteq ... \subseteq \mathbb{X}_{d-1} = \mathbb{X}$. This is in general not true for the ABM algorithm as the polynomials g in the approximate border basis have no common zeros. It is rather guaranteed that the Euclidean norm of the evaluation vector is smaller than or equal to ε with respect to the input data, which means that $\|\text{eval}_{\mathbb{X}}(g)\|_2 \leq \varepsilon$ for each $g \in G$.

Another major difference between the ABM family of algorithms and the approximate H-basis algorithm is that different norms are minimised during the computation. As pointed out before, the ABM family of algorithms constructs an approximate $\mathcal{O}$-border basis G such that for each $g \in G$ we have $g \in \mathcal{I}_{2,\varepsilon}(\mathbb{X})$. However, the approximate H-basis algorithm computes sets of polynomials F_d^0 such that each for each $g \in F_d^0$ we have $g \in \mathcal{I}_{\infty,\varepsilon}(\mathbb{X})$. Recall, that an aim of all the algorithms in question is that they are able to handle noisy input data. However, the ∞-norm is rather sensitive with respect to outliers which makes it advisable to pre-process the input data first in order to remove them. It should be noted that the approximate H-basis algorithm can be easily modified by replacing the computation of the modified QR decomposition with a SVD decomposition of the involved evaluation matrices to return polynomials in $\mathcal{I}_{2,\varepsilon}(\mathbb{X})$. Similarly, we can modify the ABM algorithm by replacing the computation of the solution of the homogeneous least squares with the computation of the modified QR decomposition of Sauer in case we want to construct polynomials that minimise the ∞-norm.

Even though we have pointed out some major conceptual differences between the algorithms, we can see from the example computations that both algorithms are capable of recovering simple geometric relations between the points, which makes both the ABM algorithm and the approximate H-basis algorithm viable for practical applications (compare Section 6.1). A comparison of the runtime of both algorithms could not be carried out as, at the time of writing, there was no publicly available version of the approximate H-basis algorithm.

4.7.2 The SOI Algorithm

In [34] Abbott et al. have proposed the so-called Stable Order Ideal (SOI) algorithm which is also concerned with the computation of border basis of ideals of points in the presence of (measurement) errors in the input data. However, the SOI algorithm is only concerned with the computation of a "stable" order ideal $\mathcal{O}$ and does not compute (almost) vanishing polynomials. In this section we will present the underlying ideas of the SOI algorithm and point out the

differences to the algorithms which were introduced in this thesis. Additionally, we apply the algorithms to the same input data sets and compare the results.

As in [34] and [49], we introduce the following definitions:

Definition 4.7.16. [Empirical point]

Let $p \in \mathbb{R}^n$ be a real point and let $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$, with each $\varepsilon_i \in \mathbb{R}_0^+$, be a vector. We call the entries of ε the **component-wise tolerances**. An **empirical point** is the pair (p, ε) and will be denoted by $p^{(\varepsilon)}$. The point p is called the **specified value** and ε is called the **tolerance**.

Definition 4.7.17. Let $p^{(\varepsilon)}$ be an empirical point. Its **ellipsoid of perturbations** is defined as

$$N(p^{(\varepsilon)}) = \{\tilde{p} \in \mathbb{R}^n \mid \|\tilde{p} - p\|_W \leq 1\}$$

where $\|\cdot\|_W = \|W\cdot\|$ is the **weighted 2-norm** (see [36] for details) defined by the diagonal weight matrix

$$W = \operatorname{diag}(1/\varepsilon_1, ..., 1/\varepsilon_n) \in \operatorname{Mat}_n(\mathbb{R}).$$

Definition 4.7.18. Let $\mathbb{X}^{(\varepsilon)} = \left\{p_1^{(\varepsilon)}, ..., p_s^{(\varepsilon)}\right\}$ be a finite set of empirical points. Each set of points $\tilde{\mathbb{X}} = \{\tilde{p}_1, ..., \tilde{p}_s\}$ that satisfies $\tilde{p}_i \in N\left(p_i^{(\varepsilon)}\right)$ for all $1 \leq i \leq s$ is called an **admissible perturbation** of $\mathbb{X}^{(\varepsilon)}$.

Definition 4.7.19. A finite set of empirical points $\mathbb{X}^{(\varepsilon)} = \left\{p_1^{(\varepsilon)}, ..., p_s^{(\varepsilon)}\right\}$ is called **distinct** if

$$N\left(p_i^{(\varepsilon)}\right) \cap N\left(p_j^{(\varepsilon)}\right) = \emptyset$$

for all $1 \leq i < j \leq s$.

Definition 4.7.20. [Stable order ideal]

An order ideal $\mathcal{O}$ is called **stable** w.r.t. $\mathbb{X}^{(\varepsilon)}$ if the evaluation matrix $\operatorname{eval}_{\tilde{\mathbb{X}}}(\mathcal{O})$ has full rank for each admissible perturbation $\tilde{\mathbb{X}}$ of $\mathbb{X}^{(\varepsilon)}$.

Definition 4.7.21. [Stable border basis]

Let $\mathbb{X}^{(\varepsilon)}$ be a finite set of distinct empirical points, let $\mathbb{X}$ be the set of specified values of $\mathbb{X}^{(\varepsilon)}$, and let $\mathcal{O}$ be a quotient basis for the vanishing ideal $\mathcal{I}(\mathbb{X})$. If $\mathcal{O}$ is stable w.r.t. $\mathbb{X}^{(\varepsilon)}$, then the $\mathcal{O}$-border basis for $\mathcal{I}(\mathbb{X})$ is called **stable** w.r.t. $\mathbb{X}^{(\varepsilon)}$.

The following definitions are related to first order approximation and first order error analysis. Fur further details please consider [34].

Let $e = (e_1, ..., e_m)$ be indeterminates and let $F = \mathbb{R}(e)$ be the field of rational functions.

Definition 4.7.22. [Multivariate Taylor Expansion]

Using multi-index notation the formal **Taylor expansion** of $f \in F$ at 0 is given by

$$f = \sum_{|\alpha| \geq 0} \frac{D^\alpha f(0)\, e^\alpha}{\alpha!},$$

where $\alpha = (\alpha_1, ..., \alpha_m) \in \mathbb{N}_0^m$, $|\alpha| = \alpha_1 + ... + \alpha_m$, and $\alpha! = \alpha_1!...\alpha_m!$. Furthermore, $D^\alpha = D_1^{\alpha_1}...D_m^{\alpha_m}$ with $D_i^j = \partial^j / \partial e_i^j$ and $e^\alpha = e_1^{\alpha_1}...e_m^{\alpha_m}$.

Each $f \in F$ can be decomposed into components of homogeneous degree such that

$$f = \sum_{k \geq 0} f_k \quad \text{where} \quad f_k = \sum_{|\alpha|=k} \frac{D^\alpha f(0) e^\alpha}{\alpha!},$$

where $D^{(0...0)} f = f$. Each polynomial f_k is called the **homogeneous component** of degree k of f.

This concept can also be extended to matrices that contain entries from F.

Definition 4.7.23. Let $M \in \mathrm{Mat}_{r,c}(F)$ and let us denote the entries of M by m_{ij}. We define M_k, the **homogeneous component** of degree k of M, as the matrix whose (i,j) entry is the homogeneous component of degree k of m_{ij}.

Let $\mathbb{X}^{(\varepsilon)} = \left\{ p_1^{(\varepsilon)}, ..., p_s^{(\varepsilon)} \right\}$ be a finite set of distinct empirical points with specified values $\mathbb{X} \subset \mathbb{R}^n$. It is possible to express admissible perturbations of $\mathbb{X}^{(\varepsilon)}$ with the help of sn (error) variables $e = (e_{11}, ..., e_{s1}, ..., e_{1n}, ..., e_{sn})$.

For this purpose, we let

$$\tilde{\mathbb{X}}(e) = \{\tilde{p}_1(e), ..., \tilde{p}_s(e)\},$$

where $\tilde{p}_i(e) = (p_{i1} + e_{i1}, ..., p_{in} + e_{in})$. The coordinates of each perturbed point $\tilde{p}_i(e)$ are elements of the polynomial ring $\mathbb{R}[e]$. Naturally, $\tilde{\mathbb{X}}$ is an admissible perturbation of $\mathbb{X}^{(\varepsilon)}$ if the condition $\|(e_{i1}, ..., e_{in})\|_W \leq 1$ on the values of the e_{kj} holds for all $1 \leq i \leq s$, where $W = \mathrm{diag}(1/\varepsilon_1, ..., 1/\varepsilon_n)$.

After these definitions we are now able to present the SOI algorithm.

Algorithm 28: Stable Order Ideal (SOI) Algorithm

Input: A set of distinct empirical points $\mathbb{X}^{(\varepsilon)} = \left\{p_1^{(\varepsilon)}, ..., p_s^{(\varepsilon)}\right\}$ with specified values $\mathbb{X} \subset \mathbb{R}^n$ and tolerance $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$, $\gamma \geq 0$, (error) variables $e = (e_{11}, ..., e_{sn})$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An order ideal $\mathcal{O}$

$\mathcal{O} := [1]$, $M_0 := (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{R})$, $M_1 := (0, ..., 0)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{R}[e])$;
$L := [t_1, ..., t_\ell] =$ all terms of degree 1 ordered decreasingly w.r.t. σ, $C := [\,]$;
while $L \neq [\,]$ **do**
  for *i := 1* **to** ℓ **do**
    $v_0 :=$ homogeneous components of degree 0 of $t_i(\tilde{\mathbb{X}}(e))$;
    $v_1 :=$ homogeneous components of degree 1 of $t_i(\tilde{\mathbb{X}}(e))$;
    $\alpha_0 := \left(M_0^{\mathrm{tr}} M_0\right)^{-1} M_0^{\mathrm{tr}} v_0$;
    $\alpha_1 := \left(M_0^{\mathrm{tr}} M_0\right)^{-1} \left(M_0^{\mathrm{tr}} v_1 + M_1^{\mathrm{tr}} v_0 - M_0^{\mathrm{tr}} M_1 \alpha_0 - M_1^{\mathrm{tr}} M_0 \alpha_0\right)$;
    $\varrho_0 := v_0 - M_0 \alpha_0$;
    $\varrho_1 := v_1 - M_0 \alpha_1 - M_1 \alpha_0$;
    $C_t \in \mathrm{Mat}_{s,sn} :=$ coefficient matrix of ϱ_1;
    $k :=$ the maximal integer such that σ_k, the minimal singular value of $C_{1:k,1:sn}$, is greater than $\|\varepsilon\|$;
    $\tilde{\varrho} := \varrho_{1:k}$;
    $\tilde{C}_t := C_{1:k,1:sn}$;
    // $\tilde{C}_t^+$ is the pseudoinverse of $\tilde{C}_t$
    $\tilde{\delta} := \tilde{C}_t^+ \tilde{\varrho}$;
    if $\|\tilde{\delta}\| > (1 + \gamma) \|\varepsilon\|$ **then**
      $M_0 := (v_0, M_0)$;
      $M_1 := (v_1, M_1)$;
      $\mathcal{O} :=$ concat$([t_i], \mathcal{O})$;
      Add to L all elements of $\{x_1 t_i, ..., x_n t_i\}$ which are not divisible by an element in L or C;
    else
      $C :=$ concat$([t_i], C)$;
      Remove all multiples of t_i from L;
    end
  end
end
return $\mathcal{O}$;

Theorem 4.7.24. *This is an algorithm which stops after finitely many steps and returns an order ideal $\mathcal{O} \subset \mathbb{T}^n$. If γ satisfies $\sup_{\delta \in D_\varepsilon} \|\varrho_{2+}(\delta)\| \leq \gamma \sqrt{s} \|\varepsilon\|^2$, then $\mathcal{O}$ is an order ideal stable w.r.t. to the set of empirical points $\mathbb{X}^{(\varepsilon)}$. If $|\mathcal{O}| = s$, then $\mathcal{I}(\mathbb{X})$ has a corresponding stable border basis w.r.t. $\mathbb{X}^{(\varepsilon)}$.*

Proof. Compare [34, Theorem 15]. □

Remark 4.7.25. As pointed out in [34, page 891], it is necessary to chose a starting value for γ even though $\sup_{\delta \in D_\varepsilon} \|\varrho_{2+}(\delta)\|$ may be unknown. As a heuristic Abbott et al. suggest to use a value of $\gamma \ll 1$ in case ϱ is approximated well by its homogeneous components of degree 0 and 1.

The approach of the SOI algorithm is quite different compared to the AVI/ABM type algorithms. The whole concept of stable border bases assumes that all the points which are in the input data set $\mathbb{X}$ are meaningful and that it is possible to associate a priori a maximal amount of noise with each coordinate of the points. In reality one can expect the measurement error to have a Gaussian distribution in each coordinate. However, this means that it is not easily possible to assign a maximal tolerance. The SOI algorithm therefore heavily relies on a preprocessing phase of the data points which tries to eliminate outliers and tries to cluster points which are close to each other. Some strategies for data clustering and preprocessing which are supposed to work well together with the SOI algorithm are discussed by Abbott et al. in [33]. It should be noted that preprocessing can be quite costly and depending on the algorithm which was used for clustering it can destroy some relations between the original input points. Additionally, the cost of the SOI algorithm itself is significantly higher than e.g. the cost of the ABM algorithm which is another reason why preprocessing the data is necessary before the SOI algorithm can be applied.
From a theoretical point of view is is nice that the stability of the border basis can be controlled in a much more direct way compared to the algorithms presented in this thesis. Nevertheless it may be a lot more difficult than for the AVI or ABM algorithm to determine a suitable ε for which a stable border basis actually exists.

Remark 4.7.26. The behaviour we just explained also finds its resemblance in the fact that SOI will in general return an order ideal $\mathcal{O}$ which contains about s elements. This is not true for the ABM algorithm where it is expected for practical values of ε that $|\mathcal{O}| \ll s$.

Remark 4.7.27. In case the SOI algorithm returns a set $\mathcal{O}$ such that $|\mathcal{O}| = s$ the associated stable border basis for $\mathcal{I}(\mathbb{X})$ is an exact border basis in the usual sense.

4.7.3 The Numerical Buchberger-Möller Algorithm (NBM)

Another approach, the so-called numerical Buchberger-Möller algorithm was presented by Fassino in [35]. It does not deal with approximate border bases but with approximate Gröbner bases. Before we can present the algorithm we need to introduce a few more abbreviations and definitions in addition to those from Subsection 4.7.2. Let $t = \prod_{i=1}^{n} x_i^{e_i}$ be a term and let $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$ be a tolerance. Then we let $\partial_k t = \frac{\partial t}{\partial x_k}$, $\partial_k \mathcal{O} = \{\partial_k t \,|\, t \in \mathcal{O}\}$, and $\varepsilon_M = \max\{\varepsilon_1, ..., \varepsilon_n\}$.

Definition 4.7.28. Let $\mathbb{X}^{(\varepsilon)}$ be a finite set of empirical points with $\mathbb{X} = \{p_1, ..., p_s\}$, $p_i = (c_{i,1}, ..., c_{i,n})$, and $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$. Then we denote by $\mathbb{X}_S^{(\delta)}$ the set of **scaled empirical points** with $\mathbb{X}_S = \{\bar{p}_1, ..., \bar{p}_s\}$, $\bar{p}_i = (d_1 c_{i,1}, ..., d_n c_{i,n})$, $(d_1, ..., d_n) \in \mathbb{R}^n$ and $\delta = (|d_1|\,\varepsilon_1, ..., |d_n|\,\varepsilon_n)$.

By $\mathbb{X}_T^{(\delta)}$ we denote the set of **translated empirical points** with $\mathbb{X}_T = \{\hat{p}_1, ..., \hat{p}_s\}$, $\hat{p}_i = (c_{i,1} + v_1, ..., c_{i,n} + v_n)$, $(v_1, ..., v_n) \in \mathbb{R}^n$ and $\delta = \varepsilon$.

Algorithm 29: Numerical Buchberger-Möller (NBM) Algorithm

Input: A set of distinct empirical points $\mathbb{X}^{(\varepsilon)} = \left\{p_1^{(\varepsilon)}, ..., p_s^{(\varepsilon)}\right\}$ with specified values $\mathbb{X} \subset \mathbb{R}^n$ and tolerance $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$, and a degree compatible term ordering σ on $\mathbb{T}^n$

Output: An order ideal $\mathcal{O}$ and a polynomial set G

$\mathcal{O} := [1]$, $G := [\,]$, $A := (1, ..., 1)^{\mathrm{tr}} \in \mathrm{Mat}_{s,1}(\mathbb{R})$;
$L := [t_1, ..., t_\ell] :=$ all terms of degree 1 ordered decreasingly w.r.t. σ;
while $L \neq [\,]$ **do**
  for *i := 1* **to** ℓ **do**
    $m := |\mathcal{O}|$;
    $b := \mathrm{eval}_{\mathbb{X}}(t_i)$;
    $s := (s_m, s_{m-1}, ..., s_1) :=$ solution of the least squares problem $\min_s \|As - b\|$;
    $\varrho := \|As - b\|$;
    $\tau := \|I_s - AA^+\| \sum_{k=1}^n \varepsilon_k \|\mathrm{eval}_{\mathbb{X}}(\partial_k t) - \mathrm{eval}_{\mathbb{X}}(\partial_k \mathcal{O})\, s\|$;
    if $\varrho < \tau$ **then**
      `// We assume that` $\mathcal{O} = [o_m, ..., o_1]$
      $g := t_i - s_m o_m - ... - s_1 o_1$;
      $G := \mathrm{concat}(G, [g])$;
    else
      $\mathcal{O} := \mathrm{concat}([t_i], \mathcal{O})$;
      $A := (b, A)$;
    end
  end
end
return $(G, \mathcal{O})$;

Theorem 4.7.29. *This is an algorithm which stops after a finite number of steps. The sets G and $\mathcal{O}$ returned by the algorithm have the following properties:*

1. *The NBM algorithm computes the same order ideal $\mathcal{O}$ for all the input sets $\mathbb{X}^{(\varepsilon)}$, $\mathbb{X}_S^{(\delta)}$, and $\mathbb{X}_T^{(\tau)}$, which means that the result is invariant to scaling and translation of the input data.*

2. *If g is a polynomial in G with coefficient vector c and $\tilde{\mathbb{X}}$ is an admissible perturbation of $\mathbb{X}$, then*

$$\frac{\|\mathrm{eval}_{\mathbb{X}}(g)\|}{\|c\|} < s \deg(g) \sum_{k=1}^n \varepsilon_k$$

and

$$\frac{\left\|\mathrm{eval}_{\tilde{\mathbb{X}}}(g)\right\|}{\|c\|} < 2s \deg(g) \sum_{k=1}^n \varepsilon_k + O\left(\varepsilon_M^2\right).$$

3. *If the zero set of G is an admissible perturbation $\hat{\mathbb{X}}$ such that $\operatorname{eval}_{\hat{\mathbb{X}}}(\mathcal{O})$ has full rank, then G is the σ-Gröbner basis of $\mathcal{I}(\hat{\mathbb{X}})$.*

4. *If $|\mathcal{O}| = s$, each polynomial g in G corresponds to one polynomial h in the $\mathcal{O}$-border basis of $\mathcal{I}(\mathbb{X})$. The support of g is a subset of the support of h. If c is the coefficient vector of g and d the coefficient vector of h, then*

$$\frac{\|d - [c, 0, ..., 0]\|}{\|c\|} \leq \deg(g) \,\|\operatorname{eval}_{\mathbb{X}}(\mathcal{O})\|\|\operatorname{eval}_{\mathbb{X}}(\mathcal{O})^{-1}\|\sum_{k=1}^{n}\varepsilon_k$$

holds.

Proof. Proofs for these properties can be found in [35, Sections 4 and 5]. □

Remark 4.7.30. Unlike the order ideal computed by the SOI algorithm, the order ideal computed by the NBM algorithm is in general not stable with respect to $\mathbb{X}^{(\varepsilon)}$.

Remark 4.7.31. The algorithm and its properties are built on first order error analysis of the least squares problem. It must be noted that the underlying assumptions only hold if the relative error in the data is small and thus higher order error components in $O\left(\varepsilon_m^2\right)$ can be neglected.

The most obvious difference between the SOI and the NBM algorithm is that the latter is also concerned with computing almost vanishing polynomials rather than just computing an order ideal. As stated in [35, Section 3] it is assumed again that the input dataset $\mathbb{X}$ has been preprocessed by e.g. one of the algorithms presented by Abbot et al. in [33]. Unlike the SOI algorithm, the NBM algorithm also returns a set G of almost vanishing polynomials. The norm of the evaluations of the polynomials g_i in G does depend on the degree of the polynomial, thus making the polynomials increasingly unreliable if high degree relations are present in the set $\mathbb{X}$. For example, in the ABM algorithm it is guaranteed that all polynomials returned by the algorithm vanish ε-approximately. The output of the NBM algorithm is invariant with respect to scaling and translation of the input data. This is not true for the ABM family of algorithms, where the scaling of the input data allows to assign individual weights to the coordinates of the points.

4.7.4 Numerical Comparison

We will first start with a few simple examples which were given either in this thesis or by Abbott and Fassino in [34] or [35]. Note that a comparison is not always straightforward because of the different meaning of the parameters (e.g. ε) in each of the approaches. For the example computations the versions of the algorithms which are available in ApCoCoA 1.8 were used. Later on we will evaluate the performance of the algorithms using both unstructured and structured input data.

Example 4.7.32. Let us apply the SOI, the NBM and the ABM algorithm to the input data of Example 4.3.8. This means that $P = \mathbb{R}[x_1, x_2]$ and

$$\mathbb{X} = [(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)].$$

If we let σ be the `DegRevLex` term ordering and $\varepsilon = 0.2$, then the ABM algorithm computes the sets $\mathcal{O} = [1, x_2, x_1, x_1x_2]$ and $G = [g_1, g_2, g_3, g_4]$ with

$$\begin{aligned} g_1 &\approx -0.697x_1^2 + 0.715x_1 - 0.044, \\ g_2 &\approx 0.685x_2^2 + 0.041x_1x_2 - 0.724x_2 - 0.021x_1 + 0.054, \\ g_3 &\approx -0.698x_1^2x_2 + 0.715x_1x_2 - 0.008x_2 - 0.008x_1 - 0.013, \\ g_4 &\approx -0.698x_1x_2^2 + 0.715x_1x_2 - 0.008x_2 - 0.008x_1 - 0.013. \end{aligned}$$

If we apply the SOI algorithm with a comparable value of $\varepsilon = (0.15, 0.15)$ we obtain the set $\mathcal{O} = [1, x_2, x_1, x_1x_2]$ which is identical to the result of the ABM algorithm. The SOI algorithm does not return an approximate border basis in this case. The actual implementation in CoCoA returns a set of almost vanishing relations which are

$$\begin{aligned} r_1 &\approx 0.707x_2^2 - 0.707x_2 + 0.035, \\ r_2 &\approx 0.707x_1^2 - 0.707x_1 + 0.035. \end{aligned}$$

We chose the parameter $\varepsilon = (0.249, 0.251)$ of the NBM algorithm such that the set $\mathcal{O}$ returned by the algorithm contained also four elements. We obtained $\mathcal{O} = \left[1, x_2, x_1, x_1^2\right]$ and the almost vanishing polynomials

$$\begin{aligned} r_1 &\approx 0.707x_2^2 - 0.707x_2 + 0.035, \\ r_2 &\approx 0.8x_1x_2 - 0.4x_1 - 0.4x_2 + 0.2, \\ r_3 &\approx 0.535x_1^3 - 0.802x_1^2 + 0.267x_1. \end{aligned}$$

In case we pick smaller values for ε, i.e. such that $\|\varepsilon\|$ becomes smaller compared to the previously used values, we obtain identical results for all algorithms. For $\varepsilon = 0.1$ the ABM algorithm, and for $\varepsilon = (0.1, 0.1)$ the SOI and the NBM algorithm return $\mathcal{O} = \left[1, x_2, x_1, x_2^2, x_1x_2\right]$. The set G contains an exact border basis and is identical for all algorithms.

Example 4.7.33. The following input data is taken from Example 6.4. in [35]. Let $P = \mathbb{R}[x_1, x_2]$, $\mathbb{X} = [(1,6), (2,3), (2.449, 2.449), (3,2), (6,1)]$ and let σ be the `DegRevLex` term ordering. If we apply the SOI or NBM algorithm we get virtually identical results for $\varepsilon = (0.018, 0.018)$. We obtain $\mathcal{O} = \left[1, x_2, x_1, x_2^2, x_2^3\right]$ and the almost vanishing polynomials

$$\begin{aligned} r_1 &\approx x_1x_2 - 0.001x_2 - 5.995, \\ r_2 &\approx x_1^2 + 0.991x_2^2 - 11.940x_1 - 11.885x_2 + 46.544, \\ r_3 &\approx x_2^4 - 14.477x_2^3 + 76.724x_2^2 - 14.862x_1 - 188.419x_2 + 214.344. \end{aligned}$$

The zero set of r_1 is a hyperbola and the zero set of r_2 is an ellipse which captures possible geometric relations between the input points. If we apply the ABM algorithm for $\varepsilon = 0.05$ we obtain $\mathcal{O} = \left[1, x_2, x_1, x_2^2\right]$ together with the approximate border basis given by

$$\begin{aligned} g_1 &\approx x_1x_2 - 0.001x_2 - 5.995, \\ g_2 &\approx x_1^2 + 0.999x_2^2 - 11.961x_1 - 11.961x_2 + 46.735, \\ g_3 &\approx x_2^3 - 12.043x_2^2 + 6.202x_1 + 47.503x_2 - 73.656, \\ g_4 &\approx x_1x_2^2 - 0.001x_1 - 6.003x_2 + 0.012. \end{aligned}$$

We can observe that r_1 and g_1 and r_2 and g_2 are almost identical. The difference in the coefficients can be explained by the fact that SOI and NBM solve the ordinary least squares problem and ABM solves the homogeneous least squares problem. However, this example demonstrates that similar results can be obtained with all algorithms if the parameters are chosen suitably.

All timings which we will now present were obtained on an Intel Pentium Dual-Core Processor with 2.17 GHz and 3 GB of Ram running ApCoCoA 1.8 on Windows 7. If a computation did not finish within two hours it was cancelled and no timings were obtained.

Example 4.7.34. The following comparison is based on test data consisting of random generic three dimensional points. With their help we want to evaluate the raw performance of the algorithms. As the input points are generic (generated via the function GenericPoints() in CoCoA), it is not reasonable to apply any preprocessing algorithms, as information would be lost. The parameter ε is always chosen in such a way that the output of the algorithms resembles an exact border basis up to rounding errors caused by floating point arithmetic. In Table 4.1 the runtime of the algorithms in seconds depending on the number of generic points can be found. In Figure 4.2 the performance of the NBM and ABM algorithm are visualised in a diagram. Clearly, in this scenario the ABM algorithm is faster than the SOI and the NBM algorithm by at least one order of magnitude. For datasets containing more than 100 points the performance of the NBM and especially the SOI algorithm is impractical for most applications.

	15	20	25	30	35	40	45	50
SOI	21.14	83.25	236.49	552.35	1212.62	2843.68	6010.79	-
NBM	1.14	3.23	6.34	11.31	21.48	30.64	50.74	73.74
ABM	0.09	0.12	0.15	0.18	0.21	0.25	0.30	0.38

Table 4.1: Performance of the SOI, NBM and ABM algorithm

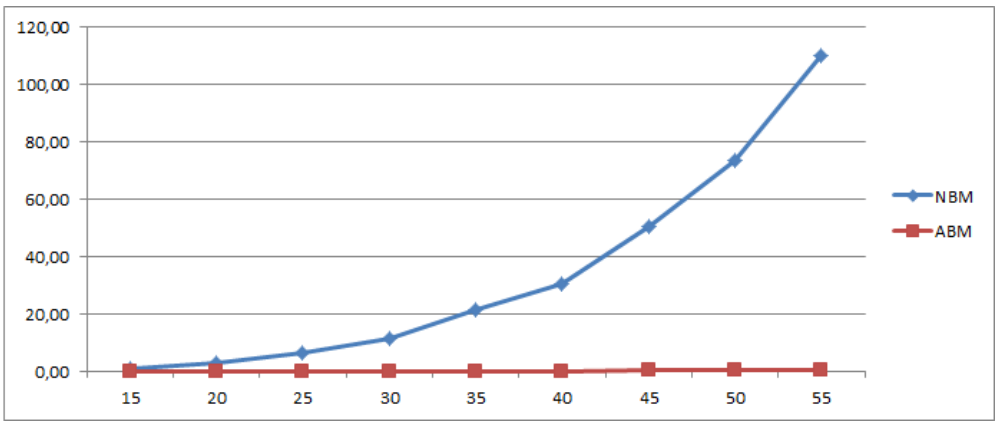

Figure 4.2: Graphical depiction of the performance of the NBM and ABM algorithm

Example 4.7.35. Finally we compare the performance of all three algorithms by applying them to s points which lie approximately on a parabola. The two dimensional points are obtained by evaluating the polynomial $f = 7.5x_1^2 - 6x_1 + 8$ on the coordinates $\left[-1 + \frac{2}{s}, ..., -1 + \frac{(s-1)2}{s}, 1\right]$.

Afterwards Gaussian noise is added. So we let

$$\mathbb{X} \approx \left[\left(-1+\frac{2}{s}, \text{eval}_{-1+\frac{2}{s}}(f)\right), ..., (1, \text{eval}_1(f))\right].$$

For all algorithms the parameter ε is chosen in such a way that the parabola is approximately recovered and either shows up in the approximate border basis (ABM) or in the almost vanishing polynomials (SOI, NBM). Furthermore, we let σ be the `DegRevLex` term ordering. In Table 4.2 we can see the runtime of the individual algorithms. Because we are only looking at two dimensional data and because the geometrical relationship that we are looking for is in fact quite simple all algorithms take less time, compared to the previous example, to return a result. The runtime of the NBM algorithm improves significantly in this situation. However, it still takes considerably more time than the ABM algorithm (compare Figure 4.3). The SOI algorithm can also in this case not be applied to datasets containing more than 110 points. The runtime of the NBM algorithm remains acceptable up to 500 points. For larger input sets preprocessing of the input data as suggested in [33] is a hard requirement. Clearly, the ABM algorithm scales better with a growing number of input points as for $s = 500$ the ABM algorithm takes 1.12 seconds to obtain an approximate border basis which also contains the parabolic relation while the NBM algorithm takes 834.20 seconds.

	15	20	25	30	35	40	45	50
SOI	11.03	81.03	218.04	501.58	1279.04	2178.06	5430.79	-
NBM	0.43	1.06	1.25	3.04	7.06	9.06	14.07	17.83
ABM	0.05	0.06	0.09	0.12	0.16	0.21	0.26	0.32

Table 4.2: Performance of the SOI, NBM and ABM algorithm

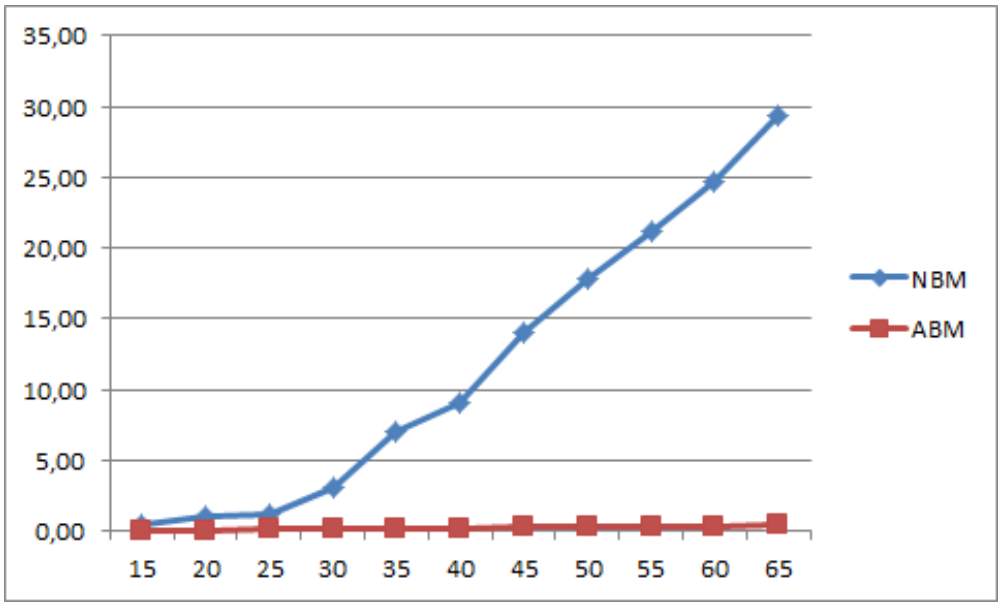

Figure 4.3: Graphical depiction of the performance of the NBM and ABM algorithm

5 The Rational Recovery Problem

Contents

So far we have seen how approximate border bases can be constructed with the help of the algorithms which we have presented in Chapter 4. Additionally, we have discussed why we are specifically interested in performing approximate interpolation and why this leads us to approximate border bases. However, a serious shortcoming of approximate border bases remained unmentioned: the lack of an extensive body of theory. For instance, it is not easily possible to transfer the concepts of ideal membership, syzygies or graded Betti numbers (compare [45] and [46] for their proper algebraic definitions) to our approximate setting. The actual complication lies in the fact the the ideal generated by an ε-approximate border bases is in general the unit ideal (see Section 4.1).

In this chapter we study a possible solution to this problem, namely to construct exact border bases which are "close" to the approximate ones with respect to the norms of the differences of the coefficient vectors of the corresponding polynomials. What we precisely mean by close will become more clear soon. When constructing these close by border bases we limit ourselves to border bases that have rational coefficients. One of the reasons is that we use numerical methods that are only capable of computing approximations of real or complex numbers. Hence, we refer to the problem of computing an exact close by border basis for a given approximate one as a **rational recovery problem**.

First of all we introduce multiplication matrices which characterise both exact and approximate border bases. Then we describe how "almost exact" multiplication matrices of an approximate $\mathcal{O}$-border basis G, as they show up when e.g. the Buchberger-Möller algorithm for border bases (18) is implemented in floating point arithmetic, can be transformed into exact multiplication matrices. Please note that already in this rather tame case the zero set $\mathcal{Z}(\langle G \rangle)$ is generally

empty and the ideal generated by G is the unit ideal (compare Section 4.1). The technique proposed originally by Auzinger and Stetter in [48] and slightly modified by Kreuzer et al. in [44] involves forming a random linear combination A_ℓ of the transposed multiplication matrices and retrieving, via the eigenvectors of A_ℓ, a set of points $\tilde{\mathbb{X}}$ which are treated as the zero set of a new exact $\mathcal{O}$-border basis $\tilde{G}$. Given $\tilde{\mathbb{X}}$ and $\mathcal{O}$ we can easily compute $\tilde{G}$. For instance we can first use the Buchberger-Möller algorithm for border bases and afterwards we can apply the Border Basis transformation algorithm (19) to its result. A detailed description of the whole computational procedure and the corresponding theory is contained in Section 5.2. Unfortunately, it will become apparent at the end of Section 5.2 that these techniques cannot be used any more whenever we are dealing with a δ-approximate border basis for which δ is significantly larger than $\varepsilon_{machine}$. This is is illustrated via several example computations. However, as a central result of this chapter we show in Section 5.3 how these problems can be solved by using a newly developed algorithm for simultaneous quasi-diagonalisation of the multiplication matrices.

Finally, we detail a rather different approach to the problem. The heuristic method which we propose forms a sum of squares expression from the polynomials in G of our given approximate $\mathcal{O}$-border basis and computes the local minima of this expression. Some of them are used to form a new set of points $\tilde{\mathbb{X}}$ for which we compute an exact $\mathcal{O}$-border basis, with the same steps that we had sketched already above. We give numerical evidence that the proposed heuristic is well-suited for practical computations.

All the algorithms which we present in this chapter have in common that they try to solve the rational recovery problem via retrieving a suitable set of points for which we compute an exact border basis with respect to $\mathcal{O}$. However, it should be noted that the problem can also be addressed by directly transforming the approximate border basis. This is one possible direction of future research that we will investigated in [44].

5.1 Multiplication Matrices for Border Bases

In this section we discuss alternative characterisations of border bases. Instead of looking directly at an ideal I it is also possible to investigate its properties by looking at the quotient algebra P/I. One way to do this is via so-called formal multiplication matrices.

Definition 5.1.1. [Formal multiplication matrix]

Let $P = K[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be its border, and let $G = \{g_1, .., g_\nu\}$ be an $\mathcal{O}$-border prebasis, such that g_j is of the form $g_j = b_j - \sum_{i=1}^{\mu} \alpha_{ij} t_j$. For every $1 \leq r \leq n$ we define the **r-th (formal) multiplication matrix** $A_r = \left(\xi_{kl}^{(r)}\right) \in \mathrm{Mat}_\mu(K)$ of G by

$$\xi_{kl}^{(r)} = \begin{cases} \delta_{ki}, & \text{if } x_r t_l = t_i \\ \alpha_{kj}, & \text{if } x_r t_l = b_j. \end{cases}$$

Here δ_{ki} denotes the Kronecker delta.

The idea behind multiplication matrices can be summarised in the following way. If we represent an element e of $\langle \mathcal{O} \rangle_K$ in terms of its coefficient vector v, $A_r v$ encodes the multiplication of e by the indeterminate x_r followed by a reduction by the elements in G. Consequently, the result will stay in $\langle \mathcal{O} \rangle_K$.

The following example will illustrate the concept of multiplication matrices.

Example 5.1.2. Let $P = \mathbb{R}[x_1, x_2]$, let $\mathcal{O} = \{1, x_1, x_2, x_1x_2, x_2^2\}$ be an order ideal, and let $G = \{g_1, ..., g_4\}$ be an $\mathcal{O}$-border prebasis with

$$\begin{aligned} g_1 &= x_1^2 - x_2^2 - x_1 + x_2, \\ g_2 &= x_1^2x_2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_3 &= x_1x_2^2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_4 &= x_2^3 - 1.5x_2^2 + 0.5x_2. \end{aligned}$$

Following the definition of formal multiplication matrices we obtain:

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.5 & -0.5 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0.5 & 0.5 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -0.5 & -0.5 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0.5 & 1.5 \end{pmatrix}.$$

If we now multiply the element $x_1 + x_1x_2$ by x_1 and reduce the result via the elements in G we obtain $x_1^2 + x_1^2x_2 \xrightarrow{G} x_1 - 1.5x_2 + x_1x_2 + 1.5x_2^2$. Now $x_1 + x_1x_2$ corresponds to the coefficient vector $(0, 1, 0, 1, 0)^{\mathrm{tr}}$ which we have to multiply by A_1. We obtain

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.5 & -0.5 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -1.5 \\ 1 \\ 1.5 \end{pmatrix}$$

which is the coefficient vector corresponding to $x_1 - 1.5x_2 + x_1x_2 + 1.5x_2^2$.

With the help of multiplication matrices it is possible to give the following alternative characterisation of border bases.

Theorem 5.1.3 (Characterisation of border bases)**.** *An $\mathcal{O}$-border prebasis G is an $\mathcal{O}$-border basis of $\langle G \rangle$ if and only if the formal multiplication matrices are pairwise commuting, i.e.*

$$A_i A_j = A_j A_i$$

for all $1 \leq i < j \leq n$.

Proof. See, for instance, [22, Proposition 16]. □

Example 5.1.4. Let us continue with Example 5.1.2. Then

$$A_1A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & -0.5 & -0.75 & -0.75 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0.5 & 0.5 & 0.75 & 0.75 \end{pmatrix} = A_2A_1.$$

This shows that G is in fact a border basis because all multiplication matrices are pairwise commuting.

Next, we collect a few well-known results about commuting matrices.

Theorem 5.1.5. *Let $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ be pairwise commuting matrices. Then the matrices preserve each others generalised eigenspaces.*

Proof. Let $V = (v_1, ..., v_o) \in \mathrm{Mat}_{m,o}(\mathbb{C})$ be a matrix containing as its columns a basis of the generalised eigenspace (compare Definition 2.3.61) associated to an eigenvalue λ of A_i, so each v_i is a (generalised) eigenvector belonging to λ. This means that, for some $k \geq 1$, we have

$$\begin{aligned} (A_i - \lambda I_m)^k V = 0_{m,o} &\iff \\ (A_i - \lambda I_m)(A_i - \lambda I_m)^{k-1} V = 0_{m,o} &\iff \\ A_i (A_i - \lambda I_m)^{k-1} V = \lambda (A_i - \lambda I_m)^{k-1} V. & \end{aligned}$$

Now we know that for an arbitrary A_j

$$\begin{aligned} A_iA_j (A_i - \lambda I_m)^{k-1} V = A_jA_i (A_i - \lambda I_m)^{k-1} V &\implies \\ A_iA_j (A_i - \lambda I_m)^{k-1} V = \lambda A_j (A_i - \lambda I_m)^{k-1} V &\implies \\ (A_i - \lambda I_m) A_j (A_i - \lambda I_m)^{k-1} V = 0_{m,o}. & \end{aligned}$$

This shows that $\mathrm{im}\left(A_j (A_i - \lambda I_m)^{k-1} V\right) \subseteq \ker(A_i - \lambda I_m) = \mathrm{im}\left((A_i - \lambda I_m)^{k-1} V\right)$ which concludes the proof. □

Theorem 5.1.6. *The following statements are equivalent:*

1. *The matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ are diagonalisable and pairwise commuting.*
2. *There exists a basis of common eigenvectors $(v_1, ..., v_m) = P \in \mathrm{Mat}_m(\mathbb{C})$ such that $P^{-1}A_iP$ is in diagonal form for all $1 \leq i \leq n$.*

Proof. First, we show that (1) implies (2). For $1 \leq i < j \leq n$, we know that $A_iA_j = A_jA_i$. Let λ be an eigenvalue of A_j. Then we denote by $V = (v_1, .., v_p) \in \mathrm{Mat}_{m,p}(\mathbb{C})$ a matrix containing a basis of the eigenvectors associated with λ as its columns, which means that $(A_j - \lambda I_m) v_q = 0_m$ for all $1 \leq q \leq p$. So we know that $\dim(\ker(A_j - \lambda I_m)) = p$. Additionally, we observe that for all $1 \leq i < j \leq n$ we have

$$A_jA_iV = A_iA_jV = \lambda A_iV. \tag{5.1}$$

Now suppose that $A_i v_q = 0_m$ for some $q \in \{1, ..., p\}$, which would imply that v_q is an eigenvector of A_i with respect to the eigenvalue 0. This would mean that v_q is a common eigenvector of A_i and A_j. So let us assume w.l.o.g. that $A_i v_q \neq 0_m$ for all $1 \leq q \leq p$. From Equation 5.1 follows that the row-vectors of $A_i V$ are eigenvectors of A_j with respect to the eigenvalue λ. Because of this we can conclude that $\operatorname{im}(A_i V) \subseteq \ker(A_j - \lambda I_m) = \operatorname{im}(V)$. As A_i is diagonalisable we also know that $\operatorname{im}(A_i V) = \operatorname{im}(V)$ and that we can choose a basis of eigenvectors for this subspace of $\operatorname{im}(A_i)$. Obviously, these are also eigenvectors of A_j. They may in fact be different from $v_1, ..., v_p$ but it suffices that they are in the span of V. If we repeat this process for every eigenvalue λ of A_j we can construct a basis transformation matrix $P = (\tilde{v}_1, ..., \tilde{v}_m) \in \operatorname{Mat}_m(\mathbb{C})$ which simultaneously diagonalises the matrices $A_1, ..., A_n$.

Now we show that statement (2) implies (1). Let $A_k = P D_k P^{-1}$ be the eigendecomposition of A_k. We thus obtain

$$\begin{aligned} A_i A_j - A_j A_i &= P D_i P^{-1} P D_j P^{-1} - P D_j P^{-1} P D_i P^{-1} \\ &= P D_i D_j P^{-1} - P D_j D_i P^{-1} \\ &= P D_i D_j P^{-1} - P D_i D_j P^{-1} \\ &= 0_{m,m} \end{aligned}$$

for all $1 \leq i < j \leq n$, which shows that the matrices $A_1, ..., A_n$ are pairwise commuting. □

Remark 5.1.7. As we can see from the proof of Theorem 5.1.6 the matrix P may not be unique even if we identify matrices which only differ in the order of the eigenvectors. For instance, let $n = 2$, let $A_1 = I_m$, and let $A_2 = 2I_m$. Clearly the matrices A_1 and A_2 are commuting and diagonalisable. In fact for every invertible matrix $P \in \mathrm{GL}_m(\mathbb{C})$, the matrix products $P^{-1} A_1 P$ and $P^{-1} A_2 P$ are diagonal.

It should be noted that in general, which means for non-diagonalisable matrices, it is not possible to find a basis transformation P such that the matrices $A_1, ..., A_n$ can be brought simultaneously into Jordan Normal Form. An example can be found in [50].

Now we explain how to compute the common eigenvectors of a set of matrices in the exact case. Later on we will present an improved method which has more favourable numerical properties and also works if the matrices are not exactly commuting.

Remark 5.1.8. Let $A_1, ..., A_n$ and P be chosen as in Theorem 5.1.6. This means that $P^{-1} A_i P = \Lambda_i \in \operatorname{Mat}_m(\mathbb{C})$ is diagonal for $1 \leq i \leq n$. Additionally let $c = (c_1, ..., c_n) \in \mathbb{R}^n$ with each $c_i \neq 0$. Then we observe that

$$\sum_{i=1}^{n} c_i A_i = \sum_{i=1}^{n} c_i \left(P \Lambda_i P^{-1} \right) = P \left(\sum_{i=1}^{n} c_i \Lambda_i \right) P^{-1}.$$

This means that we can form a random linear combination of the matrices A_i and compute its (not necessarily unique) eigendecomposition to obtain the common eigenvectors of the matrices $A_1, ..., A_n$.

Another important fact is that, for a given $\mathcal{O}$-border basis G, it is possible to obtain the roots of G via the left-hand common eigenvectors of the multiplication matrices. Compare also [49, Subsections 2.4.2 and 2.4.3].

Theorem 5.1.9. *Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, and let $G = \{g_1, ..., g_\nu\}$ be on $\mathcal{O}$-border basis for a 0-dimensional ideal $I \subseteq P$ such that $\mathcal{Z}(I) = \{p_1, ..., p_s\}$. Additionally, let $A_1, ..., A_n$ be the associated multiplication matrices and let us denote by $p_{ik} \in \mathbb{C}$ the k-th coordinate of the point p_i. Then there exist s common left-hand eigenvectors $v_1, ..., v_s$ of the multiplication matrices $A_1, ..., A_n$ such that $v_i A_k = p_{ik} v_i$ for all p_{ik} with $1 \leq i \leq s$ and $1 \leq k \leq n$.*

Proof. Let us recall that each $g_j \in G$ is of the form $g_j = b_j - \sum_{i=1}^{\mu} c_{ji} t_i$ with $c_{ji} \in \mathbb{C}$, $t_i \in \mathcal{O}$, and $b_j \in \partial\mathcal{O}$. By $a_{i,k}$ we denote the k-th column of the matrix A_i. Recall from Definition 5.1.1 that each column of a matrix A_k contains either all the coefficients c_{ji} of a polynomial g_j or it encodes a trivial relation between the elements in the order ideal $\mathcal{O}$ in form of a unit vector. Note that the coefficients c_{ji} of each polynomial $g_j \in G$ show up in at least one of the multiplication matrices. Now let $p \in \mathbb{C}^n$ be some point in $\mathbb{C}^n$ and let $v = \mathrm{eval}_p(\mathcal{O}) = \mathrm{eval}_p(t_1, ..., t_\mu)$. Then for all $g_j \in G$ the complex number $\mathrm{eval}_p(\sum_{i=1}^{\mu} c_{ji} t_i)$ is (at least) one entry of the vector-matrix products $vA_1, ..., vA_n$ as the c_{ji} belonging to the polynomial g_j are stored as columns in the multiplication matrices. Some of the entries of $vA_1, ..., vA_n$ are not associated with a polynomial $g_j \in G$. As pointed out earlier, the multiplication matrices also encode via a column unit vector all trivial relations between the elements in $\mathcal{O}$ of the form $t_k x_i - t_j = 0$ where $t_k, t_j \in \mathcal{O}$. A point p satisfies these trivial equations in terms of the multiplication matrices if the equality $v \cdot a_{i,k} = \mathrm{eval}_p(x_i)\,\mathrm{eval}_p(t_k)$ holds. Note that in this case $a_{i,k}$ is just some unit vector in $\mathbb{C}^n$. Furthermore, a point p is contained in the zero set of $\mathcal{I}$ if $v\,(c_{j1}, ..., c_{j\mu})^{\mathrm{tr}} = \mathrm{eval}_p(b_j) = \mathrm{eval}_p(x_i t_k)$ holds for all $1 \leq j \leq \nu$. If we pay attention to the structure of the multiplication matrices, we obtain once more

$$v \cdot a_{i,k} = \mathrm{eval}_p(t_1, ..., t_\mu)\, a_{i,k} = \mathrm{eval}_p(x_i)\,\mathrm{eval}_p(t_k).$$

Extending our view to all columns of all A_i simultaneously, we can observe that $vA_i = \mathrm{eval}_p(x_i)\, v$ for all $1 \leq i \leq n$ must hold. This means that p is a common zero of all polynomials in G if $v = \mathrm{eval}_p(t_1, ..., t_\mu)$ is a common left-hand eigenvector of the matrices A_i with respect to the eigenvalues $\lambda_i = \mathrm{eval}_p(x_i)$. As the eigenvectors are just evaluations of the terms in $\mathcal{O}$ with respect to a point p, it is excluded that two distinct eigenvectors can lead to the same point p, which is given by the associated eigenvalues of the multiplication matrices. In other words this means that there is no joint left eigenspace of dimension > 1. Because of the same reasoning, every point p in $\mathcal{Z}(I)$ gives rise to a simultaneous left-hand eigenvector $v = \mathrm{eval}_p(t_1, ..., t_\mu)$ of $A_1, ..., A_n$. So the number of the different points in $\mathcal{Z}(I)$, namely s, and the number of the distinct common left-hand eigenvectors is identical. □

This means that if we are able to compute the common eigenvectors for a set of matrices, Theorem 5.1.9 suggests an efficient way how to obtain the roots of a border basis. First we compute the left-hand common eigenvectors $v_1, ..., v_s$, with $1 \leq s \leq \mu$, of the corresponding

multiplication matrices $A_1, ..., A_n$. Then we construct the associated eigenvalues $v_i A_k = \lambda_{ik} v_i$ in case this has not already happened during the computation of the left-hand eigenvectors. In this way we obtain $p_i = (\lambda_{i1}, ..., \lambda_{in})$ for $1 \leq i \leq s$ such that $\mathcal{Z}(I) = \{p_1, ..., p_s\}$. Soon, we will present an algorithm which makes use of this idea.

Note that if $I = \langle G \rangle$ has at least one multiple zero, which means that $s < \mu$, the multiplication matrices will have at least one joint generalised eigenspace of dimension greater than 1. This follows readily from Theorem 5.1.9, as the number of distinct left-hand eigenvectors must match the number of distinct points in $\mathcal{Z}(I)$. Because $|\mathcal{Z}(I)| = s < \mu$ we know that the multiplication matrices must have at least one joint generalised eigenspace of dimension greater than 1. In practice it is difficult to compute the generalised eigenspaces with a numerical algorithm, essentially because the set of diagonalisable matrices is dense in $\mathrm{Mat}_m(\mathbb{C})$. A more detailed discussion of this problem can be found in [6, Subsection 7.6.5].

However, if I has only simple roots we can compute the eigenvectors in a stable way. Fortunately the assumption that I has no multiple zeros is not a true limitation for our purposes because we are working with the output of the AVI/ABM family of algorithms for which it is guaranteed that I has only simple roots.

Theorem 5.1.10. *Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, and let G be an $\mathcal{O}$-border basis for a 0-dimensional ideal $I \subseteq P$ such that I has only simple roots. Then the multiplication matrices $A_1, ..., A_n$ of G are simultaneously diagonalisable.*

Proof. Because I has only simple roots, we know that the matrices $A_1, ..., A_n$ need to posses μ distinct left-hand eigenvectors $v_1, ..., v_\mu \in \mathbb{C}^n$. Let $V = (v_1, ..., v_\mu)^{\mathrm{tr}}$ be the matrix whose rows are $v_1, ..., v_\mu$. Note that V is of full rank and thus can be inverted. So, for each A_i, we know by Theorem 5.1.9 that $VA_i = \Lambda_i V$ must hold where $\Lambda_i \in \mathrm{Mat}_\mu(\mathbb{C})$ is a diagonal matrix and contains as its k-th diagonal entry the eigenvalue λ_k of A_i that is associated with the left-hand eigenvector v_k. Finally, we obtain $VA_iV^{-1} = \Lambda_i$, which shows that the multiplication matrices can be simultaneously diagonalised. □

Example 5.1.11. Let us consider the same setup as in Example 5.1.2. We have the multiplication matrices

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0.5 & -0.5 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0.5 & 0.5 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -0.5 & -0.5 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0.5 & 1.5 \end{pmatrix}.$$

The common left-hand eigenvectors are given approximately by

$$V \approx \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0.7071 & 0.7071 & 0 & 0 & 0 \\ 0.7846 & 0.3921 & 0.3921 & 0.1961 & 0.1961 \\ 0.5774 & 0 & 0.5774 & 0 & 0.5774 \\ 0.4472 & 0.4472 & 0.4472 & 0.4472 & 0.4472 \end{pmatrix}.$$

We obtain

$$VA_1V^{-1} \approx \begin{pmatrix} 0 & & & & \\ & 1 & & & \\ & & 0.5 & & \\ & & & 0 & \\ & & & & 1 \end{pmatrix}, \; VA_2V^{-1} \approx \begin{pmatrix} 0 & & & & \\ & 0 & & & \\ & & 0.5 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}.$$

By Theorem 5.1.9 we know that $p_1 = (\lambda_{11}, \lambda_{21}) \approx (0,0)$, ..., $p_5 = (\lambda_{15}, \lambda_{25}) \approx (1,1)$. Thus we recover the points $\mathbb{X} \approx \{(0,0), (1,0), (0.5,0.5), (0,1), (1,1)\}$. This set $\mathbb{X}$ is identical to the input dataset we used to initially compute the border basis.

Lemma 5.1.12. *Let $\lambda_1, ..., \lambda_m \in \mathbb{C}$. Then the determinant of the (Vandermonde) matrix*

$$M = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{m-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{m-1} \\ 1 & \vdots & \vdots & \cdots & \vdots \\ 1 & \lambda_{m-1} & \lambda_{m-1}^2 & \cdots & \lambda_{m-1}^{m-1} \\ 1 & \lambda_m & \lambda_m^2 & \cdots & \lambda_m^{m-1} \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{C})$$

is given by $\det(M) = \prod_{1 \leq i < j \leq m} (\lambda_i - \lambda_j)$.

Proof. A proof is contained in [51, Subsection 1.4.5]. □

Proposition 5.1.13. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonalisable matrix. If A^m is the smallest power of A which can be expressed as a linear combination of the smaller powers, i.e. $A^m = \sum_{i=0}^{m-1} c_i A^i$ with $c_i \in \mathbb{C}$, then A has m distinct eigenvalues.*

Proof. Let $P \in \mathrm{Mat}_m(\mathbb{C})$ be a matrix which contains as its columns a basis of right-hand eigenvectors for A. Then the eigendecomposition of A is given by $A = P\Lambda P^{-1}$, where $\Lambda \in \mathrm{Mat}_m(\mathbb{C})$ is diagonal and $\lambda_1, ..., \lambda_m$ are the entries of Λ and consequently the eigenvalues of A. Using this special structure, we know that $A^m = P\Lambda P^{-1} \cdots P\Lambda P^{-1} = P\Lambda^m P^{-1}$. So we have reduced the problem to analysing the equation $\Lambda^m = \sum_{j=0}^{i-1} c_j \Lambda^j$. The assumption that A^m is the smallest power of A which is a linear combination of the smaller powers means that A^{m-1} and consequently Λ^{m-1} cannot be expressed as a linear combination of the smaller powers. Written in matrix form this means that

$$M = \begin{pmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{m-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{m-1} \\ 1 & \vdots & \vdots & \cdots & \vdots \\ 1 & \lambda_{m-1} & \lambda_{m-1}^2 & \cdots & \lambda_{m-1}^{m-1} \\ 1 & \lambda_m & \lambda_m^2 & \cdots & \lambda_m^{m-1} \end{pmatrix} \in \mathrm{Mat}_m(\mathbb{C})$$

only has a trivial kernel. According to Lemma 5.1.12, its determinant is given by $\det(M) = \prod_{1 \leq i < j \leq m} (\lambda_i - \lambda_j)$. As $\det(M) \neq 0$, we know that all eigenvalues of A must be distinct which concludes the proof. □

Now we shift our focus to approximate border bases and introduce all necessary definitions.

Before we will be able to characterise approximate border bases with the help of multiplication matrices, let us briefly recall the definition of an ε-approximate border basis from 4.1.4.

Let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be its border, and let $G = \{g_1, ..., g_\nu\}$ be an $\mathcal{O}$-border prebasis. This means that g_j is of the form $g_j = b_j - h_j = b_j - \sum_{i=1}^{\mu} c_{ij} t_i$ with $c_{ij} \in \mathbb{C}$. For every pair (i, j) such that b_i and b_j are neighbours in $\partial\mathcal{O}$ we compute the normal remainder $S'_{ij} = \mathrm{NR}_{\mathcal{O},G}(S_{ij})$ of the S-polynomial of g_i and g_j with respect to G. The set G is an ε-approximate border basis of the ideal $I = \langle G \rangle$ if we have $\|S'_{ij}\| \leq \varepsilon$ for all such pairs (i, j).

Definition 5.1.14. Let $\mathcal{O}$ be an order ideal, let G be an $\mathcal{O}$-border prebasis, and let $A_1, ..., A_n$ be the multiplication matrices of G. If we want to stress that the multiplication matrices belong to an (exact) border basis G we call the matrices $A_1, ..., A_n$ also **exact multiplication matrices** of G. Consequently, in case the multiplication matrices are not pairwise commuting, we call them **approximate multiplication matrices** of G.

Theorem 5.1.15 (Characterisation of approximate border bases)**.** *Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, let $G = \{g_1, .., g_\nu\}$ be an $\mathcal{O}$-border prebasis, and let $\partial\mathcal{O} = \{b_1, ..., b_\nu\}$ be the border of $\mathcal{O}$. By $A_1, ..., A_n$ we denote the associated multiplication matrices. Then the following two statements are equivalent:*

- *G is a δ-approximate $\mathcal{O}$-border basis.*
- *For all $1 \leq i < j \leq n$ and $1 \leq k \leq \mu$, the inequality $\|(A_j A_i - A_i A_j)\, e_k\| \leq \delta$ holds, where e_k is the k-th unit vector in $\mathbb{C}^\mu$.*

Proof. Before we begin with the details of the proof, we explain the necessary steps on an abstract level. First, we need to show that the coefficients of the normal remainder of all neighbouring pairs in $\partial\mathcal{O}$ and "certain" columns of the commutator of the multiplication matrices are identical. Additionally, we will observe that the commutator also contains additional columns which do not correspond to a normal remainder of a neighbouring pair. However, it will turn out that those have always norm zero.

Now we can start with the actual proof. Let $t_k \in \mathcal{O}$ be an arbitrary element of the order ideal. We now analyse what happens when t_k is multiplied by x_i and x_j with $i \neq j$ and how this translates to our multiplication matrices. In analogy to [22, Section 4] we treat all possible cases.

- **Case 1:** $x_i x_j t_k \in \mathcal{O}$. In terms of the multiplication matrices this means that $A_i A_j e_k = A_j A_i e_k$. Thus $\|A_i A_j e_k - A_j A_i e_k\| = 0$ holds for all e_k and hence for all t_k for which $x_i x_j t_k \in \mathcal{O}$.
- **Case 2:** $x_i x_j t_k \in \partial\mathcal{O}$, $x_i t_k = t_m \in \mathcal{O}$, and $x_j t_k = t_n \in \mathcal{O}$. This means that $A_j A_i e_k = A_j e_m$ and $A_i A_j e_k = A_i e_n$. We know by the construction of the (approximate) multiplication matrices that the m-th column of A_j and the n-th column of A_i are identical. So $\|A_i A_j e_k - A_j A_i e_k\| = 0$ for all e_k and hence for all t_k with the conditions mentioned in the beginning of Case 2.

- **Case 3 (next-door neighbours):** $x_i t_k = t_m \in \mathcal{O}$, and $x_j t_k = b_q \in \partial\mathcal{O}$. We can conclude that

$$A_j A_i e_k = A_j e_m = \begin{pmatrix} c_{1p} \\ c_{2p} \\ \vdots \\ c_{\mu p} \end{pmatrix} \text{ and } A_i A_j e_k = A_i \begin{pmatrix} c_{1q} \\ c_{2q} \\ \vdots \\ c_{\mu q} \end{pmatrix}.$$

  Now we let g_p, g_q be next-door-neighbours such that $g_p = x_i b_q + h_p$ and $g_q = b_q + h_q$ with $\operatorname{supp}(h_p), \operatorname{supp}(h_q) \subseteq \mathcal{O}$. If we recall that $S(g_p, g_q) = \frac{\operatorname{lcm}(b_p, b_q)}{b_p} g_p - \frac{\operatorname{lcm}(b_p, b_q)}{b_q} g_q = x_i b_q + h_p - x_i b_q - x_i h_q = h_p - x_i h_q =: S_{pq}$ the correspondence becomes evident as the c_{ip} and c_{iq} are exactly the coefficients of the polynomials h_p and h_q. So we arrive at $S'_{pq} := \operatorname{NR}_{\mathcal{O},G}(S_{pq}) = (A_j A_i e_k - A_i A_j e_k)(t_1, ..., t_\mu)$ and $\|S'_{pq}\| = \|A_j A_i e_k - A_i A_j e_k\|$.

- **Case 4 (across-the-street neighbours):** $x_i t_k = b_r \in \partial\mathcal{O}$, and $x_j t_k = b_q \in \partial\mathcal{O}$. We conclude that

$$A_j A_i e_k = A_j \begin{pmatrix} c_{1r} \\ c_{2r} \\ \vdots \\ c_{\mu r} \end{pmatrix} \text{ and } A_i A_j e_k = A_i \begin{pmatrix} c_{1q} \\ c_{2q} \\ \vdots \\ c_{\mu q} \end{pmatrix}.$$

  Now we let g_r, g_q be across-the-street neighbours such that $g_r = b_r + h_r = x_i t_k + h_r$ and $g_q = b_q + h_q = x_j t_k + h_q$ with $\operatorname{supp}(h_r), \operatorname{supp}(h_q) \subseteq \mathcal{O}$. If we recall that $S(g_r, g_q) = \frac{\operatorname{lcm}(b_r, b_q)}{b_r} g_r - \frac{\operatorname{lcm}(b_r, b_q)}{b_q} g_q = x_i x_j t_k + x_j h_r - x_i x_j t_k - x_i h_q = x_j h_r - x_i h_q =: S_{rq}$ the correspondence now also becomes clear as the c_{ir} and c_{iq} are exactly the coefficients of the polynomials h_r and h_q. So we arrive at $S'_{rq} := \operatorname{NR}_{\mathcal{O},G}(S_{rq}) = (A_j A_i e_k - A_i A_j e_k)(t_1, ..., t_\mu)$ and $\|S'_{rq}\| = \|A_j A_i e_k - A_i A_j e_k\|$.

Thus we have shown that G is a δ-approximate $\mathcal{O}$-border basis if and only if for all $1 \leq i < j \leq n$ and $1 \leq k \leq \mu$ the inequality $\|(A_j A_i - A_i A_j) e_k\| \leq \delta$ holds. □

The following definition will help us to simplify our notation with respect to approximate border bases.

Definition 5.1.16. Let e_k, with $1 \leq k \leq \mu$, be the k-th unit vector in $\mathbb{C}^\mu$, and let $A \in \operatorname{Mat}_\mu(\mathbb{C})$ be a complex matrix. We denote by

$$\|A\|_\delta = \max_{1 \leq k \leq \mu} \|A e_k\|_2$$

the maximal Euclidean norm of the columns of A.

Remark 5.1.17. Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O}$ be an order ideal, let G be an $\mathcal{O}$-border prebasis, and let $A_1, ..., A_n$ be the associated multiplication matrices. Then G is a τ-approximate border basis if $\|A_j A_i - A_i A_j\|_\delta \leq \tau$ for all $1 \leq i < j \leq n$.

5.2 The Eigenvector Algorithm

As a next step we investigate an algorithm which can be used to solve the rational recovery problem in case we are dealing with a δ-approximate border basis for which the parameter δ is roughly as large as $\varepsilon_{machine}$. As mentioned earlier, it is also important that the δ-approximate border basis is in fact close to an exact border basis of a radical ideal in order to be able to compute the solution in a stable way. For a more detailed analysis compare [44].

The original idea was brought up by Auzinger and Stetter in [48]. Here we give a version similar to the one presented in [44].

Algorithm 30: Rational Recovery via Eigenvectors

Input: An order ideal $\mathcal{O} = \{1, t_2, ..., t_\mu\}$, a δ-approximate $\mathcal{O}$-border basis G for an ideal I such that $|\mathcal{Z}(I)| = \mu$ and $\delta \geq 0$ is small

Output: An exact $\mathcal{O}$-border basis

$(A_1, ..., A_n) :=$ the multiplication matrices of G;
$\kappa := \infty$;
while $\kappa = \infty$ **do**
  $(a_1, ..., a_n) :=$ a random tuple in $\mathbb{R}^n$ with $\|(a_1, ..., a_n)\| = 1$;
  $L := \sum_{i=1}^{n} a_i A_i$;
  $M :=$ (reshape$(L^0, \mu^2, 1)$, ..., reshape$(L^{\mu-1}, \mu^2, 1)$) $\in \mathrm{Mat}_{\mu^2,\mu}(\mathbb{C})$;
  $\kappa :=$ conditionNumber(M);
end
$(v_1, ..., v_\mu) :=$ the right-hand eigenvectors of L^{tr} (e.g. via the QR algorithm (2.9.1));
// The entries of v_i are namend in the way $v_i = (v_{i1}, ..., v_{i\mu})$
for *i := 1* **to** μ **do**
  $\tilde{v}_i := \left(v_{i2}/v_{i1}, ..., v_{i\,\min(\mu,n)}/v_{i1}\right)$;
  $p_i :=$ empty n-tuple;
end
for *i := 1* **to** n **do**
  /* It is guaranteed that x_i is either in $\mathcal{O}$ or that a g_k in G containing x_i as a border term exists. */
  if x_i *is the k-th element in* $\mathcal{O}$ **then**
    for *j := 1* **to** μ **do** $p_{ji} := \tilde{v}_{jk}$;
  else
    $g := g_k \in G$ with border term x_i;
    $g := x_i - g$;
    for *j := 1* **to** μ **do** $p_{ji} := \mathrm{eval}_{\tilde{v}_j}(g)$;
  end
end
$\tilde{G} := \mathcal{O}$-border basis of the vanishing ideal of $\mathbb{X} := \{p_1, ..., p_\mu\}$ computed e.g. via Algorithm 19;
return $\tilde{G}$;

Theorem 5.2.1. *This an algorithm which takes as input a δ-approximate $\mathcal{O}$-border basis G and returns an exact $\mathcal{O}$-border basis $\tilde{G}$. If and only if $\delta \approx \varepsilon_{machine}$ the algorithm is stable and the choice of the random tuple in line 4 will have no significant influence on the result $\tilde{G}$. Furthermore if $\delta \approx \varepsilon_{machine}$ the difference of the coefficient tuples of G and the computed $\tilde{G}$ is small.*

Proof. We will not give a rigorous proof but only sketch why the individual steps are sound. Compare [44] for full details.
First, let us assume that G is a δ-approximate $\mathcal{O}$-border basis where $\delta \approx \varepsilon_{machine}$. As we have assumed that G is close to an exact border basis $\tilde{G}$ for a 0-dimensional ideal I which has only simple roots, we know by Theorem 5.1.10 that the associated multiplication matrices are simultaneously diagonalisable. The algorithm forms a random linear combination L of the matrices A_1 to A_n in line 5. In this way we reduce the problem of simultaneously diagonalising the matrices $A_1, ..., A_n$ to diagonalising the matrix L (see Remark 5.1.8). Note that this is only stable because we have assumed that $\delta \approx \varepsilon_{machine}$, as this guarantees that the individual multiplication matrices have almost identical eigenspaces. Then in line 9 the left-hand eigenvectors of L are obtained by computing the ordinary eigenvectors of L^{tr} (see Proposition 2.3.54). Now we could compute numerical approximations of the roots of G with the help of Theorem 5.1.9. So far the eigenvalues of the individual multiplication matrices have not been computed, so it would still be necessary to obtain them in order to apply Theorem 5.1.9 directly. However, if we have a closer look at the proof of Theorem 5.1.9, we observe that the common eigenvectors $v_1, ..., v_\mu$ contain scalar multiples of the evaluations of the elements in $\mathcal{O}$ on the (approximate) roots $p_1, ..., p_\mu$ of G. This means that $v_i = \alpha_i \cdot \mathrm{eval}_{p_i}(1, t_2, ..., t_\mu)$ with $\alpha_i \in \mathbb{C} \setminus \{0\}$. Normally, the α_i are given in such a way that the norm of the coefficient vectors of the v_i is one. However, in order to be able to extract the solution it is necessary to make sure that the evaluation of the term 1, the first element in the order ideal, is in fact 1. For this purpose we have to divide all entries of v_i by the first entry v_{i1} of the eigenvector (compare line 11). Note that $v_{i1} = \alpha_i \cdot \mathrm{eval}_{p_i}(1) = \alpha_i \neq 0$, so we can always divide by it. Now the coordinates for each x_i which is contained in $\mathcal{O}$ can be read off directly. This is done in line 16. All x_i which are not in $\mathcal{O}$ are computed via the relations in G in lines 18 to 20, as there must be an (almost) linear relationship between the elements in $\mathcal{O}$ and x_i it is guaranteed that for each x_i which is not in $\mathcal{O}$ there exists a $g_k \in G$ such that x_i is the border term of g_k. After the roots $\mathbb{X} = \{p_1, ..., p_\mu\}$ have been extracted, Algorithm 19 is used to construct an (exact) $\mathcal{O}$-border basis for $\mathcal{I}(\mathbb{X})$. □

Remark 5.2.2. Please note that this method relies on numerical (not exact) techniques because in line 9 the eigenvectors of L^{tr} are computed. As we know there is generally no closed form representation for the eigenvalues and -vectors (compare [5, Theorem 25.1]). So, even if the input polynomials in G have only coefficients in $\mathbb{Q}$ or in $\mathbb{Q}[i]$ and form an exact $\mathcal{O}$-border basis, the result $\tilde{G}$ may differ slightly from G in terms of the coefficients. However, for all practical purposes the accuracy which can be achieved in this way is sufficient.

Remark 5.2.3. The while loop in lines 3-8 ensures that we chose a random linear combination L of the matrices $A_1, ..., A_n$ such that the powers of L in form of the matrices $L^0, ..., L^{\mu-1}$ are linearly independent. Clearly the condition number κ of the matrix M which is computed in

line 7 would be ∞ in case the matrices $L^0, ..., L^{\mu-1}$ were linearly dependent. This is motivated by Proposition 5.1.13 and guarantees that the matrix L has μ different eigenvalues and therefore only 1-dimensional eigenspaces. The criterion in line 3 could be modified to accept online linear combinations L for which κ is smaller than a given threshold number ε. This would guarantee additional stability for the computed result. Compare also Example 5.2.7 and Figure 5.3, which illustrate that the chosen linear combination can have a significant influence on the computed solution.

Remark 5.2.4. Instead of extracting the points from the eigenvectors in lines 1-22, it would of course also be possible to use Theorem 5.1.6 directly by computing the eigenvalues for each individual multiplication matrix. This approach would, however, be more costly than the method described here, as it involves the computation of n matrix-matrix products.

Remark 5.2.5. The problem in line 23 of the algorithm to compute a border basis of $\mathcal{I}(\mathbb{X})$ for a specific order ideal $\mathcal{O}$ can be solved via Algorithm 19. Please note, that Algorithm 19 does not require an exact input border basis to work properly.

Remark 5.2.6. With respect to the stability of computing the eigenvectors in line 9 we refer to Corollary 2.9.7. In a simplified form it states that the stability of an individual eigenvector depends to some extent on the separation of its associated eigenvalue from the other eigenvalues. As other factors also play an important role, it is also advisable to read Theorem 2.9.6 for a better theoretical understanding. For Algorithm 30 this has the practical consequence that if some exact solutions are very close to each other, already small perturbations in the matrices A_i and rounding errors may change the computed solution drastically. Note that this only effects the solutions which are very close to each other, all others remain stable.

Now we investigate the workings of the algorithm when applied to an essentially exact example and to an approximate one. The latter example will guide the way how we can adapt the algorithm to the approximate case.

Example 5.2.7. Let us come back to Example 3.4.3. We consider the $\mathcal{O}$-border basis G for which we already know the zero set $\mathcal{Z}(\langle G\rangle)$ and slightly perturb it. Let again $P = \mathbb{R}[x_1, x_2]$, let $\mathcal{O} = \{1, x_1, x_2, x_1x_2, x_2^2\}$, and let $G = \{g_1, ..., g_4\}$ be an $\mathcal{O}$-border basis with

$$\begin{aligned} g_1 &= x_1^2 - x_2^2 - x_1 + x_2, \\ g_2 &= x_1^2x_2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_3 &= x_1x_2^2 - x_1x_2 - 0.5x_2^2 + 0.5x_2, \\ g_4 &= x_2^3 - 1.5x_2^2 + 0.5x_2. \end{aligned}$$

By slightly perturbing G we obtain $\bar{G} = \{\bar{g}_1, ..., \bar{g}_4\}$ with

$$\begin{aligned} \bar{g}_1 &= x_1^2 - \left(0.9999x_2^2 - 0.0001x_1x_2 - 1.0001x_2 + 0.9999x_1 + 0.00001\right), \\ \bar{g}_2 &= x_1^2x_2 - \left(0.50001x_2^2 + 0.99998x_1x_2 - 0.49999x_2 + 0.0001x_1 + 0.00001\right), \\ \bar{g}_3 &= x_1x_2^2 - \left(0.49999x_2^2 + 0.99999x_1x_2 - 0.50001x_2 - 0.0001x_1 + 0.00001\right), \\ \bar{g}_4 &= x_2^3 - \left(1.49999x_2^2 + 0.00001x_1x_2 - 0.49999x_2 - 0.0001x_1 - 0.00001\right). \end{aligned}$$

In line 1 of the algorithm we obtain the following multiplication matrices:

$$\bar{A}_1 = \begin{pmatrix} 0 & 0.00001 & 0 & 0.00001 & 0.00001 \\ 1 & 0.9999 & 0 & 0.0001 & -0.0001 \\ 0 & -1.0001 & 0 & -0.49999 & -0.50001 \\ 0 & -0.0001 & 1 & 0.99998 & 0.99999 \\ 0 & 0.9999 & 0 & 0.50001 & 0.49999 \end{pmatrix},$$

$$\bar{A}_2 = \begin{pmatrix} 0 & 0 & 0 & 0.00001 & -0.00001 \\ 0 & 0 & 0 & -0.0001 & -0.0001 \\ 1 & 0 & 0 & -0.50001 & -0.49999 \\ 0 & 1 & 0 & 0.99999 & 0.00001 \\ 0 & 0 & 1 & 0.49999 & 1.49999 \end{pmatrix}.$$

We compute the commutator between A_1 and A_2 to verify that δ is "small":

$$\bar{A}_1\bar{A}_2 - \bar{A}_2\bar{A}_1 = \begin{pmatrix} 0 & 0.00002 & 0 & 0.0000099991 & 0.000009999 \\ 0 & 0.00019998 & 0 & 0.000110009 & -0.00010999 \\ 0 & -0.00011 & 0 & 0.0001050097 & 0.0000700102 \\ 0 & 0.00017 & 0 & -0.00012999 & 0.0001100099 \\ 0 & 0.000319998 & 0 & -0.0001049901 & -0.00006999 \end{pmatrix}.$$

We observe that $\left\|\bar{A}_1\bar{A}_2 - \bar{A}_2\bar{A}_1\right\|_\delta \approx 0.0004287$ which shows that we are dealing with a δ-approximate border basis with $\delta = 0.0005$.

Now, in line 5, we have to form a random linear combination of the multiplication matrices. So let $a_1 = 0.6$ and $a_2 = 0.8$. Then we have $\|(a_1, a_2)\| = \sqrt{0.6^2 + 0.8^2} = 1$. We obtain

$$L = 0.6\bar{A}_1 + 0.8\bar{A}_2 = \begin{pmatrix} 0 & 0.000006 & 0 & 0.000014 & -0.000002 \\ 0.6 & 0.59994 & 0 & -0.00002 & -0.00014 \\ 0.8 & -0.60006 & 0 & -0.700002 & -0.699998 \\ 0 & 0.79994 & 0.6 & 1.39998 & 0.600002 \\ 0 & 0.59994 & 0.8 & 0.699998 & 1.499986 \end{pmatrix}.$$

In line 9 we compute the eigenvectors of L^{tr} and obtain

$$\begin{aligned} v_1 &\approx (0.9999, 0, 0, 0, 0), \\ v_2 &\approx (-0.4472, -0.4471, -0.4472, -0.4472, -0.4471), \\ v_3 &\approx (-0.5772, 0.0002, -0.5774, 0, -0.5773), \\ v_4 &\approx (-0.7842, -0.3912, -0.3930, -0.1962, -0.1968), \\ v_5 &\approx (-0.7062, -0.7079, 0.0013, 0.0008, 0.0003). \end{aligned}$$

In lines 10-13 we compute $\tilde{v}_1 \approx (0, 0), \tilde{v}_2 \approx (0.9997, 0.9999)$, $\tilde{v}_3 \approx (-0.0004, 1.0002)$, $\tilde{v}_4 \approx (0.4989, 0.5011)$, and $\tilde{v}_5 \approx (1.0023, -0.0019)$.

As x_1 and x_2 show up in $\mathcal{O}$ we can just let $p_1 = \tilde{v}_1, ..., p_5 = \tilde{v}_5$. In line 23 we compute the $\mathcal{O}$-border basis $\tilde{G}$ of the vanishing ideal of $\{p_1, ..., p_5\}$. We obtain the following polynomials

(rounded to 10 decimals):

$$\begin{aligned}
\tilde{g}_1 &\approx x_1^2 + 0.0007987644x_1x_2 - 1.0003977744x_2^2 - 1.0003988639x_1 + 1.0001979339x_2,\\
\tilde{g}_2 &\approx x_1^2x_2 - 1.0004031816x_1x_2 - 0.5002024061x_2^2 + 0.0009530407x_1 + 0.4999025064x_2,\\
\tilde{g}_3 &\approx x_1x_2^2 - 0.9990972612x_1x_2 - 0.4996979543x_2^2 - 0.0009526580x_1 + 0.4997979540x_2,\\
\tilde{g}_4 &\approx x_2^3 - 0.0008020420x_1x_2 - 1.5002027387x_2^2 + 0.0009519011x_1 + 0.5001027992x_2.
\end{aligned}$$

Now we look at the actual differences of the multiplication matrices of the approximate and the exact border basis. For this purpose we compute

$$\bar{A}_1 - \tilde{A}_1 = \begin{pmatrix} 0 & 0.00001 & 0 & 0.00001 & 0.00001 \\ 0 & -0.00049 & 0 & 0.00105 & -0.00105 \\ 0 & 0.00009 & 0 & -0.00008 & -0.00021 \\ 0 & 0.00069 & 0 & -0.00042 & 0.00089 \\ 0 & -0.00049 & 0 & -0.00019 & 0.00029 \end{pmatrix},$$

$$\bar{A}_2 - \tilde{A}_2 = \begin{pmatrix} 0 & 0 & 0 & 0.00001 & -0.00001 \\ 0 & 0 & 0 & -0.00105 & 0.00085 \\ 0 & 0 & 0 & -0.00021 & 0.00011 \\ 0 & 0 & 0 & 0.00089 & -0.00079 \\ 0 & 0 & 0 & 0.00029 & -0.00021 \end{pmatrix}.$$

We observe that $\|\bar{A}_1 - \tilde{A}_1\|_\delta \approx 0.00142$ and $\|\bar{A}_2 - \tilde{A}_2\|_\delta = 0.00142$. As we can see, the error is amplified but the result is still within a reasonable range of the multiplication matrices. In Figure 5.1 we see a visualisation of $\max\left(\|\bar{A}_1 - \tilde{A}_1\|_\delta, \|\bar{A}_2 - \tilde{A}_2\|_\delta\right)$ with varying a_1 and a_2. The horizontal axis depicts the value of a_1 which is incremented in steps of 0.002. Then a_2 is chosen to be $a_2 = \sqrt{1 - a_1^2}$. In the plot we can identify one peak around $a_1 \approx 0.7$. As it turns out the powers of L are almost linearly dependent for such choices of a_1 (and consequently a_2). In the next example we will investigate this behaviour more systematically.

Let us now investigate a case in which δ is significantly larger than $\varepsilon_{machine}$.

Example 5.2.8. Please note that for all results only 3 decimals after the comma are given. However, for the computations the full double accuracy was used. We start with the set of points $\mathbb{X} = \{(0,0), (0,1), (1,0), (1,1), (0.5,0.5)\}$, $P = \mathbb{R}[x_1, x_2]$, and apply ABM with $\varepsilon = 0.2$. We obtain the approximate $\mathcal{O}$-border basis $G = \{g_1, ..., g_4\}$ containing the polynomials

$$\begin{aligned}
g_1 &\approx x_2^2 - 1.026x_2 + 0.063,\\
g_2 &\approx x_1^2 + 0.060x_1x_2 - 1.056x_1 - 0.032x_2 + 0.079\\
g_3 &\approx x_1x_2^2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018\\
g_4 &\approx x_1^2x_2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018
\end{aligned}$$

with the corresponding order ideal $\mathcal{O} = \{1, x_2, x_1, x_1x_2\}$. We get the following (approximate)

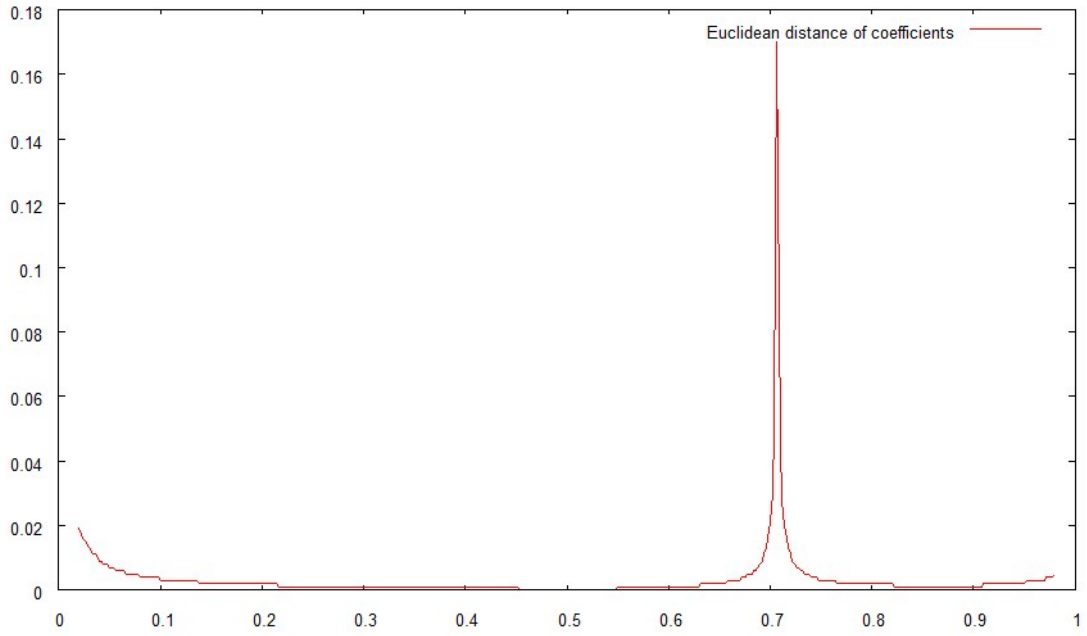

Figure 5.1: Influence of the linear combination L on $\max\left(\|\tilde{A}_1 - A_1\|_\delta, \|\tilde{A}_2 - A_2\|_\delta\right)$.

multiplication matrices:

$$A_1 = \begin{pmatrix} 0 & 0 & -0.079 & -0.018 \\ 0 & 0 & 0.032 & -0.012 \\ 1 & 0 & 1.056 & -0.012 \\ 0 & 1 & -0.060 & 1.025 \end{pmatrix}, \; A_2 = \begin{pmatrix} 0 & -0.063 & 0 & -0.018 \\ 1 & 1.026 & 0 & -0.012 \\ 0 & 0 & 0 & -0.012 \\ 0 & 0 & 1 & 1.025 \end{pmatrix}.$$

First, we observe that the matrices are still almost commuting as $\|A_1A_2 - A_2A_1\|_\delta \approx 0.054$. Thus we are dealing with a 0.06-approximate $\mathcal{O}$-border basis.

Now we form a random linear combination of the multiplication matrices such that

$$L = \frac{\sqrt{2}}{2}A_1 + \frac{\sqrt{2}}{2}A_2.$$

The eigenvectors of L^{tr} are given by

$$\begin{aligned} v_1 &\approx (0.996, 0.046, 0.061, 0.02)\,, \\ v_2 &\approx (-0.506, -0.5, -0.492, -0.5)\,, \\ v_3 &\approx (-0.691, 0.003, -0.720, -0.043)\,, \\ v_4 &\approx (0.028 - 0.692, 0.721, 0.001)\,. \end{aligned}$$

Following the steps in the algorithm we let $p_1 = (0.062, 0.469)$, $p_2 = (0.973, 0.989)$, $p_3 = (1.041, -0.004)$, and $p_4 = (25.510, -24.484)$, for which we obtain the exact $\mathcal{O}$-border basis

$\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_4\}$ with

$$\begin{aligned}
\tilde{g}_1 &= x_2^2 + 1.034x_1x_2 - 0.095x_1 - 1.998x_2 + 0.094,\\
\tilde{g}_2 &= x_1^2 + 1.032x_1x_2 - 1.145x_1 - 0.945x_2 + 0.108,\\
\tilde{g}_3 &= x_1x_2^2 + 25.451x_1x_2 - 1.158x_1 - 25.819x_2 + 1.210,\\
\tilde{g}_4 &= x_1^2x_2 - 26.441x_1x_2 + 1.106x_1 + 24.873x_2 - 1.160.
\end{aligned}$$

This leads to the multiplication matrices

$$\tilde{A}_1 = \begin{pmatrix} 0 & 0 & -0.108 & 1.160 \\ 0 & 0 & 0.945 & -24.873 \\ 1 & 0 & 1.145 & -1.106 \\ 0 & 1 & -1.032 & 26.441 \end{pmatrix}, \quad \tilde{A}_2 = \begin{pmatrix} 0 & -0.094 & 0 & -1.210 \\ 1 & 1.998 & 0 & 25.819 \\ 0 & 0.095 & 0 & 1.158 \\ 0 & -1.034 & 1 & -25.451 \end{pmatrix}.$$

If we investigate the differences between the multiplication matrices of $\tilde{G}$ and G we obtain

$$\tilde{A}_1 - A_1 = \begin{pmatrix} 0 & 0 & 0.028 & -1.178 \\ 0 & 0 & -0.913 & 24.860 \\ 0 & 0 & -0.088 & 1.093 \\ 0 & 0 & 0.971 & -25.416 \end{pmatrix}, \quad \tilde{A}_2 - A_2 = \begin{pmatrix} 0 & 0.030 & 0 & 1.191 \\ 0 & -0.972 & 0 & -25.831 \\ 0 & -0.094 & 0 & -1.171 \\ 0 & 1.034 & 0 & 26.477 \end{pmatrix}$$

with $\|\tilde{A}_1 - A_1\|_\delta \approx 35.589$ and $\|\tilde{A}_2 - A_2\|_\delta \approx 37.028$.

As we can see, there is now a large difference in the multiplication matrices. Let us look at another random linear combination with $a_1 = 0.1$ and $a_2 = 0.995$. This time we will skip the intermediate steps and only look at the exact multiplication matrices and the differences to the original ones:

$$\tilde{A}_1 = \begin{pmatrix} 0 & 0 & 0.151 & 0.032 \\ 0 & 0 & -0.537 & -0.399 \\ 1 & 0 & 0.482 & -0.066 \\ 0 & 1 & 1.070 & 1.582 \end{pmatrix}, \tilde{A}_2 = \begin{pmatrix} 0 & -0.061 & 0 & 0 \\ 1 & 1.020 & 0 & -0.001 \\ 0 & -0.005 & 0 & -0.064 \\ 0 & 0.011 & 1 & 1.033 \end{pmatrix},$$

$$\tilde{A}_1 - A_1 = \begin{pmatrix} 0 & 0 & -0.231 & -0.051 \\ 0 & 0 & 0.569 & 0.386 \\ 0 & 0 & 0.574 & 0.053 \\ 0 & 0 & -1.130 & -0.556 \end{pmatrix}, \quad \tilde{A}_2 - A_2 = \begin{pmatrix} 0 & -0.002 & 0 & -0.018 \\ 0 & 0.005 & 0 & -0.010 \\ 0 & 0.005 & 0 & 0.051 \\ 0 & -0.011 & 0 & -0.007 \end{pmatrix}.$$

Finally, we obtain $\|\tilde{A}_1 - A_1\|_\delta \approx 1.408$ and $\|\tilde{A}_2 - A_2\|_\delta \approx 0.055$.

This time the differences between the approximate and exact multiplication matrices are smaller again. What we have observed, though, is that the choice in line 4 of Algorithm 30 has a strong influence on the result of the algorithm. In Figure 5.2 we can see again a visualisation of $\max\left(\|\bar{A}_1 - \tilde{A}_1\|_\delta, \|\bar{A}_2 - \tilde{A}_2\|_\delta\right)$ with respect to the choice of a_1 and a_2. On the horizontal axis the value of a_1 is depicted and a_2 is chosen to be $a_2 = \sqrt{1 - a_1^2}$. A step width of 0.002 was chosen, in which a_1 was incremented. The strong influence of the random linear combination is also clearly visible. The result is particularly bad if the powers of L are almost linearly

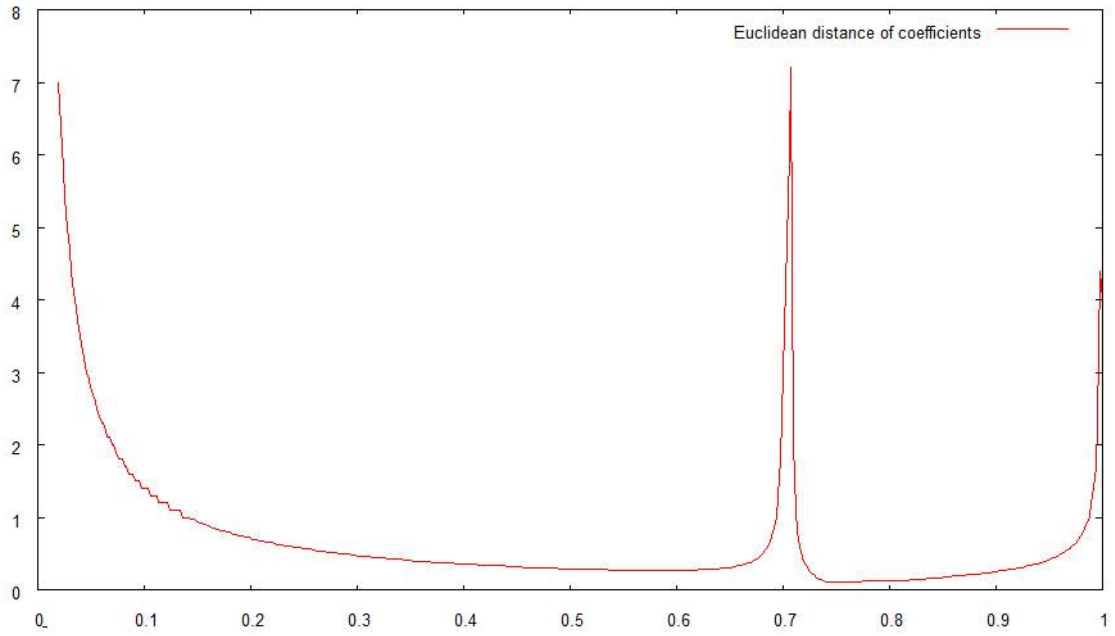

Figure 5.2: Influence of the linear combination L on $\max\left(\|\tilde{A}_1 - A_1\|_\delta, \|\tilde{A}_2 - A_2\|_\delta\right)$.

dependent. This behaviour becomes clear if we have a look at Figure 5.3. There we have plotted the condition number of the matrix M that is formed in line 6 of the algorithm. A large condition number of this matrix implies that that powers of L are almost linearly dependent. To mitigate these problems it makes sense in a practical implementation of Algorithm 30 to check that the condition number of M does not become too large. For that purpose we can modify line 3 of the algorithm to check that κ is in fact smaller than a given threshold-number ε (compare Remark 5.2.3).

We will now investigate what causes the instability in the previous example in more detail. The general problem that we have encountered is that depending on which linear combination of the multiplication matrices we choose the eigenvectors may change drastically. However, in hindsight it becomes obvious that there is a major difference between Example 5.2.7 and Example 5.2.8. The individual multiplication matrices in the almost exact example have virtually identical eigenspaces, whereas the eigenspaces of the multiplication matrices in the second example are quite different. Of course, if we form a linear combination of matrices with quite different eigenvectors we cannot expect the result to be independent of the chosen combination nor can we expect the result to be stable. As we want the multiplication matrices of the exact border basis to be reasonably small perturbations of the multiplication matrices of the approximate border basis we must first answer the question how stable the eigenvectors will behave when considering small changes in the entries of the multiplication matrices. As we have already seen in Example 2.7.13, the general problem of computing eigenvectors/eigenvalues is ill-conditioned. There is one class of matrices, though, which behaves stable under small perturbations, namely the normal matrices (compare Definition 2.3.25).

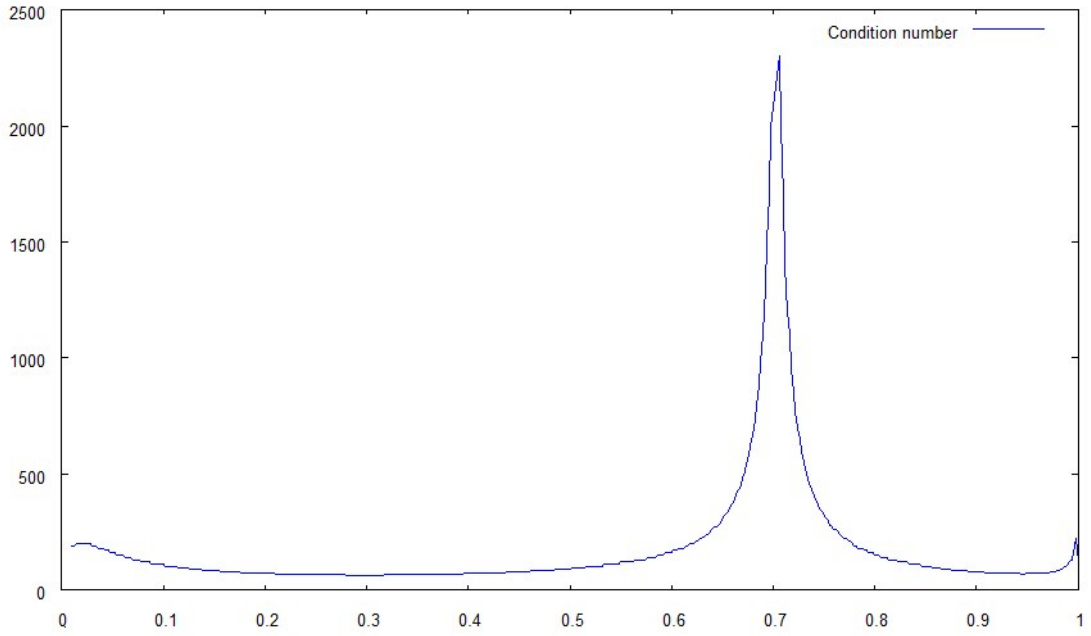

Figure 5.3: The condition number of M with varying L.

Proposition 5.2.9. *Given a normal matrix $A \in \mathrm{Mat}_m(\mathbb{C})$, the problem of computing the associated eigenvalues is well conditioned.*

Proof. First we recall Theorem 2.5.22, which states that if A is normal, it can be unitarily diagonalised. So let $A = UDU^*$, where $U \in \mathrm{Mat}_m(\mathbb{C})$ is a unitary matrix and $D \in \mathrm{Mat}_m(\mathbb{C})$ is a diagonal matrix. Then the proposition is an immediate consequence of the Bauer-Fike theorem (2.9.1) which states that if μ is an eigenvalue of $A+\delta A$, then there exists an eigenvalue $\nu \in \lambda(A)$ such that $|\lambda - \mu| \leq \|\delta A\|$. □

For a general square matrix, we are therefore interested in determining how far it deviates from a normal one. The following definition of departure from normality is motivated by the fact that normal matrices can be unitarily diagonalised.

Proposition 5.2.10. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a normal matrix and let $\Lambda \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonal matrix that contains as its diagonal entries all the eigenvalues of A. Then the equation*

$$\|A\|_F = \|\Lambda\|_F$$

holds.

Proof. By Theorem 2.5.22, we know that $A = U\Lambda U^*$ where U is unitary and Λ is diagonal containing only the eigenvalues of A. Because the Frobenius norm is invariant under multiplication with a unitary matrix (compare Proposition 2.3.34) we get

$$\|A\|_F = \|U\Lambda U^*\|_F = \|\Lambda\|_F$$

which proves the claim. □

With respect to the stability of eigenvectors, the situation is a bit more involved. Corollary 2.9.7 contains the necessary theoretical background and full details. In essence it states that the stability depends on the separation (see Definition 2.9.4) of the eigenspaces associated with the eigenvectors. This means that the closer the distance between two eigenvalues the more unstable it becomes to compute the corresponding eigenvectors.

Definition 5.2.11. [Departure from Normality]

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and let $\Lambda \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonal matrix containing on its diagonal all the eigenvalues of A. Then

$$\triangle_N(A) = \sqrt{\|A\|_F^2 - \|\Lambda\|_F^2}$$

is called the **(Euclidean) departure from normality** of A.
Similarly, for n matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ we call the sum

$$\sum_{i=1}^{n} \triangle_N^2(A_i)$$

the **common squared departure from normality** of $A_1, ..., A_n$.

Example 5.2.12. Let us consider the multiplication matrix A_1 of Example 5.2.8. We compute the departure from normality of A_1 and obtain

$$\begin{aligned}\triangle_N(A) &\approx \sqrt{\left\|\begin{pmatrix} 0 & 0 & -0.079 & -0.018 \\ 0 & 0 & 0.032 & -0.012 \\ 1 & 0 & 1.056 & -0.012 \\ 0 & 1 & -0.060 & 1.025 \end{pmatrix}\right\|_F^2 - \left\|\begin{pmatrix} 0.021 & 0 & 0 & 0 \\ 0 & 0.07 & 0 & 0 \\ 0 & 0 & 0.958 & 0 \\ 0 & 0 & 0 & 1.03 \end{pmatrix}\right\|_F^2} \\ &\approx \sqrt{4.177 - 1.984} \approx 1.481.\end{aligned}$$

So, in general, we cannot hope that the multiplication matrices are close to normal matrices and thus have their nice properties. For instance this means that also non-unitary transformations are necessary to compute a diagonalisation of them. This will become important in the next section where we develop an algorithm to simultaneously quasi-diagonalise the multiplication matrices as it shows that we need to introduce non-unitary transformations.

In order to keep the general structure of Algorithm 30 intact, we need to find a way to avoid the computation of exact eigenvectors on either of the multiplication matrices or on a combination of them. In fact, we are not interested in exact eigenvectors of the multiplication matrices. We want to find vectors which are "approximately" eigenvectors for all matrices. This idea of approximate eigenvectors and how they can be computed will be the central subject of the following section.

5.3 Simultaneous Quasi-Diagonalisation

We begin this section by introducing the concept of approximate eigenvectors and approximate eigenvalues. With the help of a small example, we learn that the approximate eigenvectors

need not lie in the vicinity of the exact eigenvectors, which is why we cannot hope to get a reasonably good approximation using the exact eigenvectors as candidates for the approximate ones. Finally, we study in detail how approximate eigenvalues and eigenvectors can be computed via simultaneous quasi-diagonalisation of the multiplication matrices.

Similarly to Bernstein in [52], we introduce the following definitions:

Definition 5.3.1. [ε-Approximate Eigenvectors and Eigenvalues]

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ and $\varepsilon \in \mathbb{R}^+$. Additionally, let us denote by $\|\cdot\|$ the Euclidean norm. If $\lambda \in \mathbb{C}$ and $z \in \mathbb{C}^m \setminus \{0_m\}$ exist such that

$$\|Az - \lambda z\| \leq \varepsilon \|z\|$$

holds, we call z an **ε-approximate eigenvector** and λ an **ε-approximate eigenvalue** of A.

Remark 5.3.2. Consequently, we say that z is an **ε-approximate common eigenvector** for n matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ if there exist $\lambda_1, ..., \lambda_n \in \mathbb{C}$ such that

$$\|A_i z - \lambda_i z\| \leq \varepsilon \|z\|$$

for all $1 \leq i \leq n$.

Intuitively, one could expect that ε-approximate eigenvectors should always be close to the exact eigenvectors of a matrix. However, the following example shows that this intuition is wrong.

Example 5.3.3. Let $\varepsilon \in \mathbb{R}^+$, let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, and let $B = \begin{pmatrix} 1 & 0 \\ 0 & 1-\varepsilon \end{pmatrix}$. For a definition of $\|\cdot\|_\sigma$ see 5.1.16. First we observe that the matrices A and B are almost commuting, as

$$\|AB - BA\|_\sigma = \left\| \begin{pmatrix} 0 & 1-\varepsilon \\ 1 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ 1-\varepsilon & 0 \end{pmatrix} \right\|_\sigma = \left\| \begin{pmatrix} 0 & -\varepsilon \\ \varepsilon & 0 \end{pmatrix} \right\|_\sigma = \varepsilon.$$

The norm one eigenvectors of A are $\frac{1}{2}\left(\sqrt{2}, \pm\sqrt{2}\right)$ and the norm one eigenvectors of B are $(1,0)$ and $(0,1)$.

Now we investigate the approximate eigenvectors of A and B. As B is only perturbed by ε from the unit matrix we know that, for every $z = (z_1, z_2) \in \mathbb{C}^2$, we have

$$\|Bz - z\| = \left\| \begin{pmatrix} z_1 \\ z_2 - z_2\varepsilon \end{pmatrix} - \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 0 \\ z_2\varepsilon \end{pmatrix} \right\| = |z_2|\,\varepsilon \leq \varepsilon \|z\|.$$

This means that every norm one eigenvector of A is also an ε-approximate norm one eigenvector of B. So the matrices A and B have the ε-approximate norm one eigenvectors $\frac{1}{2}\left(\sqrt{2}, \pm\sqrt{2}\right)$.

We have seen in this example, that we cannot hope to find ε-approximate eigenvectors by computing the exact eigenvectors of the given matrices and then to search in their vicinity. However, let us have a closer look again at the eigendecomposition of multiplication matrices and its properties.

Let $A \in \mathrm{Mat}_m(\mathbb{C})$ be a diagonalisable matrix. Then A can be written as $A = P\Lambda P^{-1}$ where $P \in \mathrm{Mat}_m(\mathbb{C})$ contains as its columns the eigenvectors of A and where $\Lambda \in \mathrm{Mat}_m(\mathbb{C})$ is a diagonal matrix that contains on its diagonal exactly the eigenvalues of A. Now let us assume we are given all n multiplication matrices $A_1, ..., A_n$ which are associated with a given $\mathcal{O}$-border basis G for a 0-dimensional ideal that has only simple roots. By Theorem 5.1.10, there exists a matrix $P \in \mathrm{Mat}_m(\mathbb{C})$ containing the common eigenvectors of the multiplication matrices such that $A_i = P\Lambda_i P^{-1}$ for $1 \leq i \leq n$, where Λ_i is a diagonal matrix. If we are dealing with multiplication matrices of an approximate border basis, such a decomposition does generally no longer exist as the matrices cannot be simultaneously diagonalised. However, a similar idea can be applied, namely to construct a matrix P such that the matrices $P^{-1}A_iP$ will be "as diagonal as possible" with respect to a cost measure which we introduce below. We can then identify the rows of P with our approximate eigenvectors and the elements on the diagonal of $P^{-1}A_iP$ with our approximate eigenvalues. We will soon clarify in greater detail what we mean precisely by this formulation. However, this is the central idea behind how we hope to solve the rational recovery problem. The process of computing this matrix P is unfortunately not completely straightforward and cannot be achieved with the standard algorithms from numerical linear algebra. Our next step will be to outline which building blocks are necessary to create such a method.

5.3.1 Building Blocks

It is well-known that no algorithm exists which could exactly compute the eigenvectors or eigenvalues of a general matrix $A \in \mathrm{Mat}_m(\mathbb{C})$ in a finite number of steps (compare [5, Theorem 25.1]). Consequently, all common algorithms construct a sequence of similarity transformations which will converge to an eigenvalue (and in some cases also an eigenvector) revealing decomposition. Compare, for instance, the basic QR-algorithm (8). In this spirit we also try to find possibly simple similarity transformations P_i which further quasi-diagonalise our matrices $A_1, ..., A_n$. To be more precise, our aim is to decompose all matrices $A_i \in \mathrm{Mat}_m(\mathbb{C})$ simultaneously such that we obtain

$$A_i = P_q \cdots P_1 (\Lambda_i + E_i) P_1^{-1} \cdots P_q^{-1}$$

for $1 \leq i \leq n$. Here $q \in \mathbb{N}$, $\Lambda_i \in \mathrm{Mat}_m(\mathbb{C})$ is diagonal, and $E_i \in \mathrm{Mat}_m(\mathbb{C})$ has only entries off its diagonal. The matrices P_i shall be chosen in such a way that $\sum_{i=1}^{n} \|E_i\|_F$ is minimised. Our approach will be based on the idea of the Jacobi algorithm, but we extend it to n matrices, similarly to what was proposed by Fu and Gao in [53]. Our work differs from that of the mentioned authors in the points that we arrive at a different parameter choice for the transformation matrices and that we give a version which also works for complex matrices and not only for real ones.

First we analyse which kind of similarity transformations are necessary to achieve our goal.

An immediate consequence of the spectral theorem (2.5.22) is that it does not suffice to limit ourselves to a sequence of unitary similarity transformations if we want to diagonalise general matrices. Therefore, Eberlein proposed in [59] to additionally apply shear similarity transformations to reduce the matrices departure from normality and then to apply unitary similarity

transformations to reduce their departure from diagonality. Please note that Eberlein only examined how to diagonalise a single matrix. Certainly, there is no intrinsic requirement which forces us to use exactly these transformation matrices. However, we stick to them as they are well understood and can be handled without using non-linear optimisation techniques to obtain a suitable choice of parameters. This is a common shortcoming of other approaches which rely on more involved transformations. In Subsection 5.3.6 we present a detailed numerical comparison that exemplifies the advantages of our method.

Proposition 5.3.4. *Let* $A \in \mathrm{Mat}_m(\mathbb{C})$. *Then*

$$\inf_{P \in \mathrm{GL}_m(\mathbb{C})} \left\| P^{-1}AP \right\|_F^2 = \|\Lambda\|_F^2$$

where $\Lambda \in \mathrm{Mat}_m(\mathbb{C})$ *is a diagonal matrix that contains on its diagonal exactly the eigenvalues of* A. *The lower bound is attained if and only if* A *is diagonalisable.*

Proof. A proof can be found in [60] after the first theorem. □

Remark 5.3.5. By Proposition 5.3.4 it becomes evident that every matrix can be "almost" diagonalised. This is the root cause why it is practically impossible to compute the Jordan Normal Form of a non-diagonalisable matrix with the help of a numerical algorithm in a stable way.

Corollary 5.3.6. *Let* $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$. *If we determine* $P \in \mathrm{GL}_m(\mathbb{C})$ *in such a way that* $\sum_{i=1}^n \left\| P^{-1}A_iP \right\|_F^2$ *is smaller than* $\sum_{i=1}^n \|A_i\|^2$, *then* $\sum_{i=1}^n \triangle_N^2 \left(P^{-1}A_iP \right)$ *is also smaller than* $\sum_{i=1}^n \triangle_N^2 (A_i)$.

Proof. For $1 \leq i \leq n$ let $\Lambda_i \in \mathrm{Mat}_m(\mathbb{C})$ be the diagonal matrix that contains on its diagonal exactly the eigenvalues of A_i. Furthermore, suppose that we have found P such that $\sum_{i=1}^n \left\| P^{-1}A_iP \right\|_F^2 < \sum_{i=1}^n \|A_i\|_F^2$. Then the relations

$$\begin{aligned}
\sum_{i=1}^n \left\| P^{-1}A_iP \right\|_F^2 &< \sum_{i=1}^n \|A_i\|_F^2 && \Longleftrightarrow \\
\sum_{i=1}^n \left(\left\| P^{-1}A_iP \right\|_F^2 - \|\Lambda_i\|_F^2 \right) &< \sum_{i=1}^n \left(\|A_i\|_F^2 - \|\Lambda_i\|_F^2 \right) && \Longleftrightarrow \\
\sum_{i=1}^n \left(\sqrt{\|P^{-1}A_iP\|_F^2 - \|\Lambda_i\|_F^2} \right)^2 &< \sum_{i=1}^n \left(\sqrt{\|A_i\|_F^2 - \|\Lambda_i\|_F^2} \right)^2 &&
\end{aligned}$$

hold. If we note that the eigenvalues of $P^{-1}A_iP$ are the same as those of A_i, then using Proposition 5.3.4 we immediately obtain

$$\sum_{i=1}^n \triangle_N^2 \left(P^{-1}A_iP \right) = \sum_{i=1}^n \left(\sqrt{\|P^{-1}A_iP\|_F^2 - \|\Lambda_i\|_F^2} \right)^2 < \sum_{i=1}^n \left(\sqrt{\|A_i\|_F^2 - \|\Lambda_i\|_F^2} \right)^2 = \sum_{i=1}^n \triangle_N^2 (A_i) .$$

□

This means that in order to find a similarity transformation that reduces the common squared departure from normality of the transformed matrices $A_1, ..., A_n$, it suffices to find a similarity transformation which reduces the sum of the squared Frobenius norms of the transformed matrices.

Definition 5.3.7. Let $A \in \mathrm{Mat}_m(\mathbb{C})$. Then we can write A as the sum of the elements on its diagonal $D \in \mathrm{Mat}_m(\mathbb{C})$ and the off diagonal entries $E \in \mathrm{Mat}_m(\mathbb{C})$ such that $A = D + E$. The number

$$\Delta_D(A) = \|E\|_F$$

is called the **departure from diagonality** of A. Similarly we call for n matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ the sum

$$\sum_{i=1}^{n} \Delta_D^2(A_i)$$

the **common squared departure from diagonality** of $A_1, ..., A_n$.

For convenience, let us denote the entries of a matrix $A \in \mathrm{Mat}_{m,n}(\mathbb{C})$ by a_{ij}. Using the notation of [58], we introduce two different kinds of matrices.

Let $k \in \mathbb{N}$ and let $1 \leq p < q \leq m$. The unitary matrices $U^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ and the matrices $S^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ are identical to the unit matrix $I_m \in \mathrm{Mat}_m(\mathbb{C})$ except for four entries.

Please recall that, for $y \in \mathbb{R}$ the hyperbolic sine and cosine functions are given by $\sinh(y) = \frac{1}{2}(e^y - e^{-y})$ and $\cosh(y) = \frac{1}{2}(e^y + e^{-y})$.

Definition 5.3.8. [Shear Rotation Matrix]

Let $\alpha_{k,p,q} \in]-\pi, \pi]$ and let $y_{k,p,q} \in \mathbb{R}$. We call a matrix $S^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ with entries identical to the unit matrix I_m except for the four entries

$$\begin{aligned} s_{pp}^{(k,p,q)} &= \cosh(y_{k,p,q}), \\ s_{pq}^{(k,p,q)} &= -ie^{i\alpha_{k,p,q}} \sinh(y_{k,p,q}), \\ s_{qp}^{(k,p,q)} &= ie^{-i\alpha_{k,p,q}} \sinh(y_{k,p,q}), \\ s_{qq}^{(k,p,q)} &= \cosh(y_{k,p,q}) \end{aligned}$$

a **shear rotation matrix** with parameters $\alpha_{k,p,q}$ and $y_{k,p,q}$.

Definition 5.3.9. [Unitary Rotation Matrix]

Let $\varphi_{k,p,q}, \theta_{k,p,q} \in]-\pi, \pi]$. We call a matrix $U^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ with entries identical to the unit matrix I_m except for the four entries

$$\begin{aligned} u_{pp}^{(k,p,q)} &= \cos(\theta_{k,p,q}), \\ u_{pq}^{(k,p,q)} &= -e^{i\varphi_{k,p,q}} \sin(\theta_{k,p,q}), \\ u_{qp}^{(k,p,q)} &= e^{-i\varphi_{k,p,q}} \sin(\theta_{k,p,q}), \\ u_{qq}^{(k,p,q)} &= \cos(\theta_{k,p,q}) \end{aligned}$$

a **unitary rotation matrix** with parameters $\varphi_{k,p,q}$ and $\theta_{k,p,q}$.

Example 5.3.10. Let $m = 4$, $k = 1$, $p = 2$, $q = 3$, and let $\alpha_{1,2,3} \in]-\pi, \pi]$ and $y_{1,2,3} \in \mathbb{R}$. Then

$$S^{(1,2,3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cosh(y_{1,2,3}) & -ie^{i\alpha_{1,2,3}} \sinh(y_{1,2,3}) & 0 \\ 0 & ie^{-i\alpha_{1,2,3}} \sinh(y_{1,2,3}) & \cosh(y_{1,2,3}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \in \mathrm{Mat}_4(\mathbb{C})$$

is a shear rotation matrix. Furthermore let $\varphi_{1,2,3}, \theta_{1,2,3} \in]-\pi, \pi]$. Then

$$U^{(1,2,3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_{1,2,3}) & -e^{i\varphi_{1,2,3}} \sin(\theta_{1,2,3}) & 0 \\ 0 & e^{-i\varphi_{1,2,3}} \sin(\theta_{1,2,3}) & \cos(\theta_{1,2,3}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \in \mathrm{Mat}_4(\mathbb{C})$$

is a unitary rotation matrix.

As we want to apply a sequence of similarity transformations to the initially given matrices $A_1, ..., A_n$, it is crucial that the inverse of the matrices $S^{(k,p,q)}$ and $U^{(k,p,q)}$ can be computed efficiently as well.

Proposition 5.3.11. *Let $S^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ be a shear rotation matrix and let $U^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{C})$ be a unitary rotation matrix. The matrix $\left(S^{(k,p,q)}\right)^{-1}$ is identical to the unit matrix I_m except for the four entries*

$$\begin{aligned} s_{pp}^{(k,p,q)} &= \cosh(y_{k,p,q}), \\ s_{pq}^{(k,p,q)} &= ie^{i\alpha_{k,p,q}} \sinh(y_{k,p,q}), \\ s_{qp}^{(k,p,q)} &= -ie^{-i\alpha_{k,p,q}} \sinh(y_{k,p,q}), \\ s_{qq}^{(k,p,q)} &= \cosh(y_{k,p,q}). \end{aligned}$$

Similarly, the matrix $\left(U^{(k,p,q)}\right)^{-1}$ is identical to the unit matrix I_m except for the four entries

$$\begin{aligned} u_{pp}^{(k,p,q)} &= \cos(\theta_{k,p,q}), \\ u_{pq}^{(k,p,q)} &= e^{i\varphi_{k,p,q}} \sin(\theta_{k,p,q}), \\ u_{qp}^{(k,p,q)} &= -e^{-i\varphi_{k,p,q}} \sin(\theta_{k,p,q}), \\ u_{qq}^{(k,p,q)} &= \cos(\theta_{k,p,q}). \end{aligned}$$

Furthermore, the matrix U is unitary.

Proof. As we only look at one specific matrix $S^{(k,p,q)}$ and $U^{(k,p,q)}$ we omit the superscript (k,p,q) and subscript k,p,q. We also omit the trivial entries, which are identical to the corresponding

entries of the unit matrix I_m, and only give the relevant entries of the matrix products:

$$\begin{aligned}
\left(SS^{-1}\right)_{pp} &= \cosh^2(y) - \sinh^2(y) = 1,\\
\left(SS^{-1}\right)_{pq} &= \cosh(y)\, ie^{i\alpha} \sinh(y) - ie^{i\alpha} \sinh(y) \cosh(y) = 0,\\
\left(SS^{-1}\right)_{qp} &= ie^{-i\alpha} \sinh(y) \cosh(y) - \cosh(y)\, ie^{-i\alpha} \sinh(y) = 0,\\
\left(SS^{-1}\right)_{qq} &= \cosh^2(y) - \sinh^2(y) = 1,
\end{aligned}$$

$$\begin{aligned}
\left(UU^{-1}\right)_{pp} &= \cos^2(\theta) + \sin^2(\theta) = 1,\\
\left(UU^{-1}\right)_{pq} &= \cos(\theta)\, e^{i\varphi} \sin(\theta) - e^{i\varphi} \sin(\theta) \cos(\theta) = 0,\\
\left(UU^{-1}\right)_{qp} &= e^{-i\varphi} \sin(\theta) \cos(\theta) - \cos(\theta)\, e^{-i\varphi} \sin(\theta) = 0,\\
\left(UU^{-1}\right)_{qq} &= \sin(\theta) \sin(\theta) + \cos(\theta) \cos(\theta) = 1.
\end{aligned}$$

We observe that $U = U^*$, which proves that U is in fact a unitary matrix. □

Next we analyse the actions of $S^{(k,p,q)}$ and $U^{(k,p,q)}$ on a single matrix $A_h^{(k)}$, i.e. we give explicit representations for the entries of the resulting matrices (compare [58, Section 3]). Those representations are crucial for our further analysis. Additionally, they can be used to create efficient implementations of the similarity transformations that do not rely on full matrix-matrix multiplications. As we first only look at one combination (p,q) in iteration k for a given matrix A_h, we omit the superscript (k,p,q) and (k) as well as the subscript h to keep our notation simple. Let

$$\begin{aligned}
A' &= S^{-1}AS\\
A'' &= U^*A'U
\end{aligned}$$

and let additionally

$$\begin{aligned}
d &= a_{pp} - a_{qq},\\
\xi &= e^{i\alpha}a_{qp} + e^{-i\alpha}a_{pq},\\
R &= \sinh^2(y)\, d + \frac{i}{2} \sinh(2y)\, \xi,\\
T &= -\frac{i}{2} \sinh(2y)\, d + \sinh^2(y)\, \xi.
\end{aligned}$$

Proposition 5.3.12. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$, let $k \in \mathbb{N}$, and let $1 \leq p < q \leq m$. Furthermore, let $S = S^{(k,p,q)}$ be a shear rotation matrix with parameters $\alpha = \alpha_{k,p,q}$ and $y = y_{k,p,q}$. Then the entries of the matrix $A' = S^{-1}AS$ are given by*

$$
\begin{aligned}
a_{ij}^{'} &= a_{ij} && (i,j \neq p,q)\\
a_{pj}^{'} &= \cosh(y)\, a_{pj} + ie^{i\alpha}\sinh(y)\, a_{qj} && (j \neq p,q)\\
a_{qj}^{'} &= -ie^{-i\alpha}\sinh(y)\, a_{pj} + \cosh(y)\, a_{qj} && (j \neq p,q)\\
a_{jp}^{\prime} &= \cosh(y)\, a_{jp} + ie^{-i\alpha}\sinh(y)\, a_{jq} && (j \neq p,q)\\
a_{jq}^{\prime} &= -ie^{i\alpha}\sinh(y)\, a_{jp} + \cosh(y)\, a_{jq} && (j \neq p,q)\\
a_{pp}^{\prime} &= a_{pp} + R\\
a_{pq}^{\prime} &= a_{pq} + e^{i\alpha}T\\
a_{qp}^{\prime} &= a_{qp} + e^{-i\alpha}T\\
a_{qq}^{\prime} &= a_{qq} - R.
\end{aligned}
$$

Proof. Through tedious but straightforward computation we obtain:

$$
\begin{aligned}
a_{ij}^{'} &= a_{ij} && (i,j \neq p,q)\\
a_{pj}^{'} &= s_{pp}^{-1}a_{pj} + s_{pq}^{-1}a_{qj} = \cosh(y)\, a_{pj} + ie^{i\alpha}\sinh(y)\, a_{qj} && (j \neq p,q)\\
a_{qj}^{'} &= s_{qp}^{-1}a_{pj} + s_{qq}^{-1}a_{qj} = -ie^{-i\alpha}\sinh(y)\, a_{pj} + \cosh(y)\, a_{qj} && (j \neq p,q)\\
a_{jp}^{\prime} &= a_{jp}s_{pp} + a_{jq}s_{qp} = \cosh(y)\, a_{jp} + ie^{-i\alpha}\sinh(y)\, a_{jq} && (j \neq p,q)\\
a_{jq}^{\prime} &= a_{jp}s_{pq} + a_{jq}s_{qq} = -ie^{i\alpha}\sinh(y)\, a_{jp} + \cosh(y)\, a_{jq} && (j \neq p,q)
\end{aligned}
$$

$$
\begin{aligned}
a_{pp}^{\prime} &= s_{pp}\left(s_{pp}^{-1}a_{pp} + s_{pq}^{-1}a_{qp}\right) + s_{qp}\left(s_{pp}^{-1}a_{pq} + s_{pq}^{-1}a_{qq}\right)\\
&= \left(\cosh(y)\, a_{pp} + ie^{i\alpha}\sinh(y)\, a_{qp}\right)\cosh(y) +\\
&\quad \left(\cosh(y)\, a_{pq} + ie^{i\alpha}\sinh(y)\, a_{qq}\right) ie^{-i\alpha}\sinh(y)\\
&= \cosh^2(y)\, a_{pp} + ie^{i\alpha}\sinh(y)\cosh(y)\, a_{qp} +\\
&\quad ie^{-i\alpha}\sinh(y)\cosh(y)\, a_{pq} - \sinh^2(y)\, a_{qq}\\
&= a_{pp} + \sinh^2(y)\,(a_{pp} - a_{qq}) + \frac{i}{2}\sinh(2y)\left(e^{i\alpha}a_{qp} + e^{-i\alpha}a_{pq}\right)\\
&= a_{pp} + R
\end{aligned}
$$

$$
\begin{aligned}
a_{pq}^{\prime} &= s_{pq}\left(s_{pp}^{-1}a_{pp} + s_{pq}^{-1}a_{qp}\right) + s_{qq}\left(s_{pp}^{-1}a_{pq} + s_{pq}^{-1}a_{qq}\right)\\
&= -\left(\cosh(y)\, a_{pp} + ie^{i\alpha}\sinh(y)\, a_{qp}\right) ie^{i\alpha}\sinh(y) +\\
&\quad \left(\cosh(y)\, a_{pq} + ie^{i\alpha}\sinh(y)\, a_{qq}\right)\cosh(y)\\
&= \cosh(y)\, a_{pq}\cosh(y) +\\
&\quad e^{i\alpha}\left(-i\cosh(y)\sinh(y)\, a_{pp} + e^{i\alpha}\sinh(y)\sinh(y)\, a_{qp} + i\sinh(y)\cosh(y)\, a_{qq}\right)\\
&= \cosh^2(y)\, a_{pq} + e^{i\alpha}\left(-i\cosh(y)\sinh(y)\,(a_{pp} - a_{qq}) + e^{i\alpha}\sinh^2(y)\, a_{qp}\right)\\
&= e^{i\alpha}\left(-i\cosh(y)\sinh(y)\, d + \sinh^2(y)\, a_{qp}e^{i\alpha} + \left(1 + \sinh^2(y)\right) a_{pq}e^{-i\alpha}\right)\\
&= a_{pq} + e^{i\alpha}\left(-i\cosh(y)\sinh(y)\, d + \sinh^2(y)\left(a_{qp}e^{i\alpha} + a_{pq}e^{-i\alpha}\right)\right)\\
&= a_{pq} + e^{i\alpha}\left(-\frac{i}{2}\sinh(2y)\, d + \sinh^2(y)\,\xi\right) = a_{pq} + e^{i\alpha}T
\end{aligned}
$$

$$\begin{aligned}
a'_{qp} &= s_{pp}\left(s_{qp}^{-1}a_{pp}+s_{qq}^{-1}a_{qp}\right)+s_{qp}\left(s_{qp}^{-1}a_{pq}+s_{qq}^{-1}a_{qq}\right)\\
&= \left(-ie^{-i\alpha}\sinh(y)\,a_{pp}+\cosh(y)\,a_{qp}\right)\cosh(y)+\\
&\quad \left(-ie^{-i\alpha}\sinh(y)\,a_{pq}+\cosh(y)\,a_{qq}\right)ie^{-i\alpha}\sinh(y)\\
&= e^{-i\alpha}\left(e^{i\alpha}\cosh^2(y)\,a_{qp}-i\sinh(y)\cosh(y)\,a_{pp}\right)+\\
&\quad e^{-i\alpha}\left(e^{-i\alpha}\sinh^2(y)\,a_{pq}+i\sinh(y)\cosh(y)\,a_{qq}\right)\\
&= e^{-i\alpha}\left(-\frac{i}{2}\sinh(2y)\,(a_{pp}-a_{qq})+e^{i\alpha}\left(1+\sinh^2(y)\right)a_{qp}+e^{-i\alpha}\sinh^2(y)\,a_{pq}\right)\\
&= a_{qp}+e^{-i\alpha}\left(-\frac{i}{2}\sinh(2y)\,d+\sinh^2(y)\,\xi\right)=a_{qp}+e^{-i\alpha}T
\end{aligned}$$

$$\begin{aligned}
a'_{qq} &= s_{pq}\left(s_{qp}^{-1}a_{pp}+s_{qq}^{-1}a_{qp}\right)+s_{qq}\left(s_{qp}^{-1}a_{pq}+s_{qq}^{-1}a_{qq}\right)\\
&= \left(-ie^{-i\alpha}\sinh(y)\,a_{pp}+\cosh(y)\,a_{qp}\right)\left(-ie^{i\alpha}\right)\sinh(y)+\\
&\quad \left(-ie^{-i\alpha}\sinh(y)\,a_{pq}+\cosh(y)\,a_{qq}\right)\cosh(y)\\
&= a_{qq}-\sinh^2(y)\,(a_{pp}-a_{qq})-\frac{i}{2}\sinh(2y)\left(e^{i\alpha}a_{qp}+e^{-i\alpha}a_{pq}\right)\\
&= a_{qq}-R.
\end{aligned}$$

□

Now we also analyse the effect of applying a similarity transformation in form of matrices $\left(U^{(k,p,q)}\right)^*$ and $U^{(k,p,q)}$ on a matrix $A^{'}$. For this purpose, let us define

$$\begin{aligned}
d' &= a'_{pp}-a'_{qq},\\
\xi' &= e^{i\varphi}a'_{qp}+e^{-i\varphi}a'_{pq},\\
P &= -\sin^2(\theta)\,d'+\frac{1}{2}\sin(2\theta)\,\xi',\\
Q &= \frac{1}{2}\sin(2\theta)\,d'+\sin^2(\theta)\,\xi'.
\end{aligned}$$

Proposition 5.3.13. *Let $A^{'}\in \mathrm{Mat}_m(\mathbb{C})$, let $k\in\mathbb{N}$, and let $1\le p<q\le m$. Furthermore, let $U=U^{(k,p,q)}$ be a unitary rotation matrix with parameters $\varphi=\varphi_{k,p,q}$ and $\theta=\theta_{k,p,q}$. Then the entries of the matrix $A''=U^*A'U$ are given by*

$$\begin{aligned}
a^{''}_{ij} &= a^{'}_{ij} && (i,j\neq p,q)\\
a^{''}_{pj} &= \cos(\theta)\,a^{'}_{pj}+e^{i\varphi}\sin(\theta)\,a^{'}_{qj} && (j\neq p,q)\\
a^{''}_{qj} &= -e^{-i\varphi}\sin(\theta)\,a'_{pj}+\cos(\theta)\,a^{'}_{qj} && (j\neq p,q)\\
a^{''}_{jp} &= \cos(\theta)\,a^{'}_{jp}+e^{-i\varphi}\sin(\theta)\,a^{'}_{jq} && (j\neq p,q)\\
a^{''}_{jq} &= -e^{i\varphi}\sin(\theta)\,a^{'}_{jp}+\cos(\theta)\,a^{'}_{jq} && (j\neq p,q)\\
a^{''}_{pp} &= a^{'}_{pp}+P\\
a^{''}_{pq} &= a^{'}_{pq}-e^{i\varphi}Q\\
a^{''}_{qp} &= a^{'}_{qp}-e^{-i\varphi}Q\\
a^{''}_{qq} &= a^{'}_{qq}-P.
\end{aligned}$$

Proof. Again through straightforward computation we obtain:

$$\begin{aligned}
a_{ij}^{''} &= a_{ij}^{'} && (i,j \neq p,q)\\
a_{pj}^{''} &= \cos(\theta)\, a_{pj}^{'} + e^{i\varphi}\sin(\theta)\, a_{qj}^{'} && (j \neq p,q)\\
a_{qj}^{''} &= -e^{-i\varphi}\sin(\theta)\, a_{pj}^{\prime} + \cos(\theta)\, a_{qj}^{'} && (j \neq p,q)\\
a_{jp}^{''} &= \cos(\theta)\, a_{jp}^{'} + e^{-i\varphi}\sin(\theta)\, a_{jq}^{'} && (j \neq p,q)\\
a_{jq}^{''} &= -e^{i\varphi}\sin(\theta)\, a_{jp}^{'} + \cos(\theta)\, a_{jq}^{'} && (j \neq p,q)
\end{aligned}$$

$$\begin{aligned}
a_{pp}^{''} &= u_{pp}\left(u_{pp}^{-1}a_{pp}^{'} + u_{pq}^{-1}a_{qp}^{'}\right) + u_{qp}\left(u_{pp}^{-1}a_{pq}^{'} + u_{pq}^{-1}a_{qq}^{'}\right)\\
&= \cos^2(\theta)\, a_{pp}^{'} + e^{i\varphi}\sin(\theta)\cos(\theta)\, a_{qp}^{'} +\\
&\quad \cos(\theta)\, e^{-i\varphi}\sin(\theta)\, a_{pq}^{'} + e^{i\varphi}\sin^2(\theta)\, e^{-i\varphi}a_{qq}^{'}\\
&= a_{pp}^{'} - \sin^2(\theta)\, a_{pp}^{'} + \sin^2(\theta)\, a_{qq}^{'} + \frac{1}{2}\sin(2\theta)\left(e^{i\varphi}a_{qp}^{'} + e^{-i\varphi}a_{pq}^{'}\right)\\
&= a_{pp}^{'} + P
\end{aligned}$$

$$\begin{aligned}
a_{pq}^{''} &= u_{pq}\left(u_{pp}^{-1}a_{pp}^{'} + u_{pq}^{-1}a_{qp}^{'}\right) + u_{qq}\left(u_{pp}^{-1}a_{pq}^{'} + u_{pq}^{-1}a_{qq}^{'}\right)\\
&= -e^{i\varphi}\cos(\theta)\sin(\theta)\, a_{pp}^{'} - e^{i\varphi}\sin(\theta)\, e^{i\varphi}\sin(\theta)\, a_{qp}^{'} +\\
&\quad \cos^2(\theta)\, a_{pq}^{'} + e^{i\varphi}\sin(\theta)\cos(\theta)\, a_{qq}^{'}\\
&= a_{pq}^{'} - e^{i\varphi}\\
&\quad \left(e^{-i\varphi}\sin^2(\theta)\, a_{pq}^{'} + \cos(\theta)\sin(\theta)\, a_{pp}^{'} + e^{i\varphi}\sin^2(\theta)\, a_{pq}^{'} - \sin(\theta)\cos(\theta)\, a_{qq}^{'}\right)\\
&= a_{pq}^{'} - e^{i\varphi}\left(\sin^2(\theta)\left(e^{-i\varphi}a_{pq}^{'} + e^{i\varphi}a_{pq}^{'}\right) + \sin(\theta)\cos(\theta)\left(a_{pp}^{'} - a_{qq}^{'}\right)\right)\\
&= a_{pq}^{'} - e^{i\varphi}\left(\sin^2(\theta)\left(e^{-i\varphi}a_{pq}^{'} + e^{i\varphi}a_{pq}^{'}\right) + \frac{1}{2}\sin(2\theta)\left(a_{pp}^{'} - a_{qq}^{'}\right)\right)\\
&= a_{pq}^{'} - e^{i\varphi}Q
\end{aligned}$$

$$\begin{aligned}
a_{qp}^{''} &= u_{pp}\left(u_{qp}^{-1}a_{pp}^{'} + u_{qq}^{-1}a_{qp}^{'}\right) + u_{qp}\left(u_{qp}^{-1}a_{pq}^{'} + u_{qq}^{-1}a_{qq}^{'}\right)\\
&= -e^{-i\varphi}\sin(\theta)\cos(\theta)\, a_{pp}^{'} + \cos^2(\theta)\, a_{qp}^{'} -\\
&\quad e^{-i\varphi}\sin^2(\theta)\, e^{-i\varphi}a_{pq}^{'} + \cos(\theta)\, e^{-i\varphi}\sin(\theta)\, a_{qq}^{'}\\
&= a_{qp}^{'} - e^{-i\varphi}\\
&\quad \left(\sin^2(\theta)\left(e^{i\varphi}a_{qp}^{'} + e^{-i\varphi}a_{pq}^{'}\right) + \sin(\theta)\cos(\theta)\, a_{pp}^{'} - \cos(\theta)\, e^{-i\varphi}\sin(\theta)\, a_{qq}^{'}\right)\\
&= a_{qp}^{'} - e^{-i\varphi}\\
&\quad \left(\sin^2(\theta)\left(e^{i\varphi}a_{qp}^{'} + e^{-i\varphi}a_{pq}^{'}\right) + \frac{1}{2}\sin(2\theta)\left(a_{pp}^{'} - a_{qq}^{'}\right)\right)\\
&= a_{qp}^{'} - e^{-i\varphi}Q
\end{aligned}$$

$$\begin{aligned}
a_{qq}^{''} &= u_{pq}\left(u_{qp}^{-1}a_{pp}^{'} + u_{qq}^{-1}a_{qp}^{'}\right) + u_{qq}\left(u_{qp}^{-1}a_{pq}^{'} + u_{qq}^{-1}a_{qq}^{'}\right)\\
&= -e^{-i\varphi}\sin^2(\theta) - e^{i\varphi}a_{pp}^{'} - \cos(\theta)\, e^{i\varphi}\sin(\theta)\, a_{qp}^{'} -
\end{aligned}$$

$$\begin{aligned} & e^{-i\varphi}\sin(\theta)\cos(\theta)\,a'_{pq} + \cos^2(\theta)\,a'_{qq} \\ = \;& a'_{qq} - \sin^2(\theta)\,a'_{qq} + \sin^2(\theta)\,a'_{pp} - \frac{1}{2}\sin(2\theta)\left(e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq}\right) \\ = \;& a'_{qq} - P. \end{aligned}$$

□

Let us briefly recall the setting in which we operate. We are given n (diagonalisable) matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ which we want to simultaneously quasi-diagonalise via a sequence of similarity transformations. As pointed out before, the algorithm that we want to design will iteratively apply a shear rotation, that minimises the common squared departure from normality, followed by a unitary rotation, that minimises the common squared departure from diagonality, until convergence has occurred.
In the first step of each iteration, meaning when we form the matrices $A'_j = S^{-1}A_jS$ for $1 \leq j \leq n$, we want to choose α and y of S in such a way that the common squared departure from normality of the matrices A'_i, i.e. $\sum_{j=1}^n \triangle_N^2\left(A'_j\right) = \sum_{j=1}^n \triangle_N^2\left(S^{-1}A_jS\right)$ is (approximately) minimised. According to Corollary 5.3.6 this task is achieved by determining S via the parameters α and y in such a way that the sum of the squared Frobenius norms of the matrices $S^{-1}A'_jS$ decreases compared to the sum of the squared Frobenius norms of the matrices A_j. As we will soon uncover, unfortunately there is no closed form solution to solve our optimisation problem. This is why we will choose a linear approximation to the solution, for which we show that it will always lead to

$$\sum_{j=1}^n \left\|S^{-1}A_jS\right\|_F^2 - \sum_{j=1}^n \|A_i\|_F^2 \leq 0.$$

Then in the second step of each iteration, when we form the matrices $A''_j = U^*A'_jU$ for $1 \leq j \leq n$, both θ and φ need to be chosen in such a way that the common squared departure from diagonality of the matrices A''_j is minimised compared to the common squared departure of the matrices A'_j, meaning $\sum_{j=1}^n \triangle_N^2\left(U^*A'_jU\right)$ is minimal with respect to θ and φ which determine U. In the following, our strategy will be to adapt the proofs that are given in [59] and [61] for the choice of parameters for the case of a single matrix to the case of n matrices.

Let us start with analysing the parameter choice for the shear transformation S.

5.3.2 Choice of Parameters in the Shear Transformation

By $\|\cdot\|$ we denote the Euclidean vector norm or the Frobenius matrix norm. Additionally, we denote by $a_{k,ij}$ the (i,j)-entry of the matrix A_k.

Given matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ and $1 \leq p < q \leq m$, our objective is to determine the parameters α and y of S such that

$$\sum_{j=1}^n \|A'_j\|^2 = \sum_{j=1}^n \|S^{-1}A_jS\|^2$$

is minimised.

Definition 5.3.14. To abbreviate our notation, let us introduce the following quantities:

$$\begin{aligned} K_h &= K_{h,pq} = \sum_{j\neq p,q} \left(a_{h,pj}\bar{a}_{h,qj} - \bar{a}_{h,jp}a_{h,jq}\right), \\ G_h &= G_{h,pq} = \sum_{j\neq p,q} \left(|a_{h,pj}|^2 + |a_{h,jp}|^2 + |a_{h,jq}|^2 + |a_{h,qj}|^2\right), \\ C_h &= A_h A_h^* - A_h^* A_h, \\ c_h &= C_{h,pq}. \end{aligned}$$

Once again whenever we talk about a single matrix h for convenience we will omit the subscript h and write K, G, C and c instead of K_h, G_h, C_h, c_h.

First we show the following lemma:

Lemma 5.3.15.

$$\Im\left(\overline{d}\xi\right) - \Im\left(e^{-i\alpha}K\right) = -\Im\left(e^{-i\alpha}c\right). \tag{5.2}$$

Proof.

$$\begin{aligned} & \Im\left(\overline{d}\xi\right) - \Im\left(e^{-i\alpha}K\right) \\ = \; & \Im\left(\left(\overline{a}_{pp} - \overline{a}_{qq}\right)\left(a_{qp}e^{i\alpha} + a_{pq}e^{-i\alpha}\right) - e^{-i\alpha}\sum_{j\neq p,q}\left(a_{pj}\overline{a}_{qj} - \overline{a}_{jp}a_{jq}\right)\right) \\ = \; & \Im\left(\overline{a}_{pp}a_{qp}e^{i\alpha} + \overline{a}_{pp}a_{pq}e^{-i\alpha} - \overline{a}_{qq}a_{qp}e^{i\alpha} - \overline{a}_{qq}a_{pq}e^{-i\alpha} - e^{-i\alpha}\sum_{j\neq p,q}\left(a_{pj}\overline{a}_{qj} - \overline{a}_{jp}a_{jq}\right)\right) \\ = \; & \frac{1}{2i}\left(\overline{a}_{pp}a_{qp}e^{i\alpha} + \overline{a}_{pp}a_{pq}e^{-i\alpha} - \overline{a}_{qq}a_{qp}e^{i\alpha} - \overline{a}_{qq}a_{pq}e^{-i\alpha} - e^{-i\alpha}\sum_{j\neq p,q}\left(a_{pj}\overline{a}_{qj} - \overline{a}_{jp}a_{jq}\right) - \right. \\ & \left.\left(a_{pp}\overline{a}_{qp}e^{-i\alpha} + a_{pp}\overline{a}_{pq}e^{i\alpha} - a_{qq}\overline{a}_{qp}e^{-i\alpha} - a_{qq}\overline{a}_{pq}e^{i\alpha} - e^{i\alpha}\sum_{j\neq p,q}\left(\overline{a}_{pj}a_{qj} - a_{jp}\overline{a}_{jq}\right)\right)\right) \\ = \; & \frac{1}{2i}e^{i\alpha}\left(\overline{a}_{pp}a_{qp} - \overline{a}_{qq}a_{qp} - a_{pp}\overline{a}_{pq} + a_{qq}\overline{a}_{pq} + \sum_{j\neq p,q}\left(\overline{a}_{pj}a_{qj} - a_{jp}\overline{a}_{jq}\right)\right) - \\ & \frac{1}{2i}e^{-i\alpha}\left(a_{pp}\overline{a}_{qp} - a_{qq}\overline{a}_{qp} - \overline{a}_{pp}a_{pq} + \overline{a}_{qq}a_{pq} + \sum_{j\neq p,q}\left(a_{pj}\overline{a}_{qj} - \overline{a}_{jp}a_{jq}\right)\right) \\ = \; & -\Im\left(e^{-i\alpha}\left(a_{pp}\overline{a}_{qp} - a_{qq}\overline{a}_{qp} - \overline{a}_{pp}a_{pq} + \overline{a}_{qq}a_{pq} + \sum_{j\neq p,q}\left(a_{pj}\overline{a}_{qj} - \overline{a}_{jp}a_{jq}\right)\right)\right) \\ = \; & -\Im\left(e^{-i\alpha}c_{pq}\right) = -\Im\left(e^{-i\alpha}c\right). \end{aligned}$$

□

Next we focus on minimizing a single matrix $A_h' = S^{-1}A_hS$. Afterwards we extend this result to the more general case of several matrices $A_1', ..., A_n'$. For that purpose we adapt the approach

of [59] to our setting. In the following part we also write A' instead of $A_h^{'}$ in order to keep the notation as simple as possible. Now we show how $\|A'\|^2$ can be expressed in terms of $\|A\|^2$ and the transformations which are applied.

Lemma 5.3.16. *Let $A \in \mathrm{Mat}_m(\mathbb{C})$, let $1 \leq p < q \leq m$, let S be a shear rotation matrix with parameters α and y, and let $A' = S^{-1}AS$. Then we have*

$$\begin{aligned} \|A'\|^2 &= \|A\|^2 + (\cosh(2y) - 1)\, G + 2\sinh(2y)\, \Im\left(e^{-i\alpha}K\right) + \\ &\quad \sinh^2(2y)\left(|d|^2 + |\xi|^2\right) - \sinh(4y)\, \Im\left(\overline{d}\xi\right). \end{aligned} \tag{5.3}$$

Proof.

$$\begin{aligned} \left\|A'\right\|^2 &= \sum_{j\neq p,q} \left|\cosh(y)\, a_{pj} + ie^{i\alpha}\sinh(y)\, a_{qj}\right|^2 + \sum_{j\neq p,q} \left|-ie^{-i\alpha}\sinh(y)\, a_{pj} + \cosh(y)\, a_{qj}\right|^2 + \\ &\quad \sum_{j\neq p,q} \left|\cosh(y)\, a_{jp} + ie^{-i\alpha}\sinh(y)\, a_{jq}\right|^2 + \sum_{j\neq p,q} \left|-ie^{i\alpha}\sinh(y)\, a_{jp} + \cosh(y)\, a_{jq}\right|^2 + \\ &\quad \left|a_{pq} + e^{i\alpha}T\right|^2 + \left|a_{qp} + e^{-i\alpha}T\right|^2 + |a_{pp} + R|^2 - |a_{qq} - R|^2 + \sum_{j,i\neq p,q} |a_{ij}|^2 \\ &= \ldots + \left(a_{pq} + e^{i\alpha}T\right)\left(\bar{a}_{pq} + e^{-i\alpha}\bar{T}\right) + \left(a_{qp} + e^{-i\alpha}T\right)\left(\bar{a}_{qp} + e^{i\alpha}\bar{T}\right) + \\ &\quad (a_{pp} + R)\left(\overline{a}_{pp} + \overline{R}\right) + (a_{qq} - R)\left(\overline{a}_{qq} - \overline{R}\right) + \sum_{j,i\neq p,q} |a_{ij}|^2 \\ &= \ldots + |a_{pq}|^2 + a_{pq}e^{-i\alpha}\bar{T} + \bar{a}_{pq}e^{i\alpha}T + T\bar{T} + |a_{qp}|^2 + a_{qp}e^{i\alpha}\bar{T} + e^{-i\alpha}T\bar{a}_{qp} + T\bar{T} + \\ &\quad |a_{pp}|^2 + a_{pp}\overline{R} + \overline{a}_{pp}R + R\overline{R} + |a_{qq}|^2 - a_{qq}\overline{R} - \overline{a}_{qq}R + R\overline{R} + \sum_{j,i\neq p,q} |a_{ij}|^2 \\ &= \sum_{j\neq p,q} \left(\cosh(y)\, a_{pj} + ie^{i\alpha}\sinh(y)\, a_{qj}\right)\left(\cosh(y)\, \overline{a}_{pj} - ie^{-i\alpha}\sinh(y)\, \overline{a}_{qj}\right) + \\ &\quad \sum_{j\neq p,q} \left(-ie^{-i\alpha}\sinh(y)\, a_{pj} + \cosh(y)\, a_{qj}\right)\left(ie^{i\alpha}\sinh(y)\, \overline{a}_{pj} + \cosh(y)\, \overline{a}_{qj}\right) + \\ &\quad \sum_{j\neq p,q} \left(\cosh(y)\, a_{jp} + ie^{-i\alpha}\sinh(y)\, a_{jq}\right)\left(\cosh(y)\, \overline{a}_{jp} - ie^{i\alpha}\sinh(y)\, \overline{a}_{jq}\right) + \\ &\quad \sum_{j\neq p,q} \left(-ie^{i\alpha}\sinh(y)\, a_{jp} + \cosh(y)\, a_{jq}\right)\left(ie^{-i\alpha}\sinh(y)\, \overline{a}_{jp} + \cosh(y)\, \overline{a}_{jq}\right) + \ldots \\ &= \sum_{j\neq p,q} \left(\cosh^2(y)\, |a_{pj}|^2 - ie^{-i\alpha}\cosh(y)\sinh(y)\, a_{pj}\overline{a}_{qj}\right) + \\ &\quad \sum_{j\neq p,q} \left(ie^{i\alpha}\sinh(y)\cosh(y)\, a_{qj}\overline{a}_{pj} + \sinh^2(y)\, |a_{qj}|^2\right) + \\ &\quad \sum_{j\neq p,q} \left(\sinh^2(y)\, |a_{pj}|^2 - ie^{-i\alpha}\sinh(y)\cosh(y)\, a_{pj}\overline{a}_{qj}\right) + \\ &\quad \sum_{j\neq p,q} \left(ie^{i\alpha}\cosh(y)\sinh(y)\, a_{qj}\overline{a}_{pj} + \cosh^2(y)\, |a_{qj}|^2\right) + \\ &\quad \sum_{j\neq p,q} \left(\cosh^2(y)\, |a_{jp}|^2 - ie^{i\alpha}\cosh(y)\sinh(y)\, a_{jp}\overline{a}_{jq}\right) + \\ &\quad \sum_{j\neq p,q} \left(ie^{-i\alpha}\sinh(y)\cosh(y)\, a_{jq}\overline{a}_{jp} + \sinh^2(y)\, |a_{jq}|^2\right) + \end{aligned}$$

$$
\begin{aligned}
&\sum_{j\neq p,q}\left(\sinh^2(y)\,|a_{jp}|^2 - ie^{i\alpha}\sinh(y)\cosh(y)\,a_{jp}\overline{a}_{jq}\right)+\\
&\sum_{j\neq p,q}\left(ie^{-i\alpha}\cosh(y)\sinh(y)\,a_{jq}\overline{a}_{jp}+\cosh^2(y)\,|a_{jq}|^2\right)+\ldots\\
=\;&\left(\cosh^2(y)+\sinh^2(y)-1\right)\sum_{j\neq p,q}\left(|a_{pj}|^2+|a_{qj}|^2+|a_{jp}|^2+|a_{jq}|^2\right)+\\
&\sum_{j\neq p,q}\left(|a_{pj}|^2+|a_{qj}|^2+|a_{jp}|^2+|a_{jq}|^2\right)+\\
&2\cosh(y)\sinh(y)\,\frac{2}{2i}\sum_{j\neq p,q}\left(e^{-i\alpha}a_{pj}\overline{a}_{qj}-e^{-i\alpha}\overline{a}_{jp}a_{jq}-e^{i\alpha}\overline{a}_{pj}a_{qj}+e^{i\alpha}a_{jp}\overline{a}_{jq}\right)+\ldots\\
=\;&(\cosh(2y)-1)\,G+\sum_{j\neq p,q}\left(|a_{pj}|^2+|a_{qj}|^2+|a_{jp}|^2+|a_{jq}|^2\right)+\\
&2\sinh(2y)\,\frac{1}{2i}\sum_{j\neq p,q}\left(e^{-i\alpha}a_{pj}\overline{a}_{qj}-e^{-i\alpha}\overline{a}_{jp}a_{jq}-e^{i\alpha}\overline{a}_{pj}a_{qj}+e^{i\alpha}a_{jp}\overline{a}_{jq}\right)+\ldots\\
=\;&(\cosh(2y)-1)\,G+\sum_{j\neq p,q}\left(|a_{pj}|^2+|a_{qj}|^2+|a_{jp}|^2+|a_{jq}|^2\right)+2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right)+\\
&|a_{pq}|^2+a_{pq}e^{-i\alpha}\bar{T}+\bar{a}_{pq}e^{i\alpha}T+T\bar{T}+|a_{qp}|^2+a_{qp}e^{i\alpha}\bar{T}+e^{-i\alpha}T\bar{a}_{qp}+T\bar{T}+\\
&|a_{pp}|^2+a_{pp}\overline{R}+\overline{a}_{pp}R+R\overline{R}+|a_{qq}|^2-a_{qq}\overline{R}-\overline{a}_{qq}R+R\overline{R}+\sum_{j,i\neq p,q}|a_{ij}|^2\\
=\;&(\cosh(2y)-1)\,G+2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right)+\|A\|_F^2+a_{pq}e^{-i\alpha}\bar{T}+\\
&\bar{a}_{pq}e^{i\alpha}T+T\bar{T}+a_{qp}e^{i\alpha}\bar{T}+e^{-i\alpha}T\bar{a}_{qp}+T\bar{T}+\\
&a_{pp}\overline{R}+\overline{a}_{pp}R+R\overline{R}-a_{qq}\overline{R}-\overline{a}_{qq}R+R\overline{R}\\
=\;&\ldots+a_{pq}e^{-i\alpha}\left(\frac{i}{2}\sinh(2y)\,\overline{d}+\sinh^2(y)\,\overline{\xi}\right)+\bar{a}_{pq}e^{i\alpha}\left(-\frac{i}{2}\sinh(2y)\,d+\sinh^2(y)\,\xi\right)+\\
&\left(-\frac{i}{2}\sinh(2y)\,d+\sinh^2(y)\,\xi\right)\left(\frac{i}{2}\sinh(2y)\,\overline{d}_{pq}+\sinh^2(y)\,\overline{\xi}\right)+\\
&a_{qp}e^{-i\alpha}\left(\frac{i}{2}\sinh(2y)\,\overline{d}+\sinh^2(y)\,\overline{\xi}\right)+\bar{a}_{qp}e^{i\alpha}\left(-\frac{i}{2}\sinh(2y)\,d+\sinh^2(y)\,\xi\right)+\\
&\left(-\frac{i}{2}\sinh(2y)\,d+\sinh^2(y)\,\xi\right)\left(\frac{i}{2}\sinh(2y)\,\overline{d}+\sinh^2(y)\,\overline{\xi}\right)+\\
&a_{pp}\left(\sinh^2(y)\,\overline{d}-\frac{i}{2}\sinh(2y)\,\overline{\xi}\right)+\overline{a}_{pp}\left(\sinh^2(y)\,d+\frac{i}{2}\sinh(2y)\,\xi\right)+\\
&\left(\sinh^2(y)\,d+\frac{i}{2}\sinh(2y)\,\xi\right)\left(\sinh^2(y)\,\overline{d}-\frac{i}{2}\sinh(2y)\,\overline{\xi}\right)+\\
&a_{qq}\left(\sinh^2(y)\,\overline{d}-\frac{i}{2}\sinh(2y)\,\overline{\xi}\right)+\overline{a}_{qq}\left(\sinh^2(y)\,d+\frac{i}{2}\sinh(2y)\,\xi\right)+\\
&\left(\sinh^2(y)\,d+\frac{i}{2}\sinh(2y)\,\xi\right)\left(\sinh^2(y)\,\overline{d}-\frac{i}{2}\sinh(2y)\,\overline{\xi}\right)\\
=\;&\ldots+\frac{i}{2}e^{-i\alpha}\sinh(2y)\,a_{pq}\left(\overline{a}_{pp}-\overline{a}_{qq}\right)+e^{-i\alpha}\sinh^2(y)\,a_{pq}\left(\overline{a}_{qp}e^{-i\alpha}+\overline{a}_{pq}e^{i\alpha}\right)-\\
&\frac{i}{2}e^{i\alpha}\sinh(2y)\,\overline{a}_{pq}\left(a_{pp}-a_{qq}\right)+e^{i\alpha}\sinh^2(y)\,\overline{a}_{pq}\left(a_{qp}e^{i\alpha}+a_{pq}e^{-i\alpha}\right)+
\end{aligned}
$$

$$
\begin{aligned}
& 2\left(\frac{1}{4}\sinh^2(2y)\,|d|^2 - \frac{i}{2}\sinh(2y)\sinh^2(y)\,d\overline{\xi}\right) + \\
& 2\left(\frac{i}{2}\sinh(2y)\sinh^2(y)\,\overline{d}\xi + \sinh^4(y)\,|\xi|^2\right) + \\
& \frac{i}{2}e^{-i\alpha}\sinh(2y)\,a_{qp}\left(\overline{a}_{pp} - \overline{a}_{qq}\right) + e^{-i\alpha}\sinh^2(y)\,a_{qp}\left(\overline{a}_{qp}e^{-i\alpha} + \overline{a}_{pq}e^{i\alpha}\right) - \\
& \frac{i}{2}e^{i\alpha}\sinh(2y)\,\overline{a}_{qp}\left(a_{pp} - a_{qq}\right) + e^{i\alpha}\sinh^2(y)\,\overline{a}_{qp}\left(a_{qp}e^{i\alpha} + a_{pq}e^{-i\alpha}\right) + \\
& \sinh^2(y)\,a_{pp}\left(\overline{a}_{pp} - \overline{a}_{qq}\right) - \frac{i}{2}\sinh(2y)\,a_{pp}\left(\overline{a}_{qp}e^{-i\alpha} + \overline{a}_{pq}e^{i\alpha}\right) + \\
& \sinh^2(y)\,\overline{a}_{pp}\left(a_{pp} - a_{qq}\right) + \frac{i}{2}\sinh(2y)\,\overline{a}_{pp}\left(a_{qp}e^{i\alpha} + a_{pq}e^{-i\alpha}\right) + \\
& \sinh^2(y)\,a_{qq}\left(\overline{a}_{pp} - \overline{a}_{qq}\right) - \frac{i}{2}\sinh(2y)\,a_{qq}\left(\overline{a}_{qp}e^{-i\alpha} + \overline{a}_{pq}e^{i\alpha}\right) + \\
& \sinh^2(y)\,\overline{a}_{qq}\left(a_{pp} - a_{qq}\right) + \frac{i}{2}\sinh(2y)\,\overline{a}_{qq}\left(a_{qp}e^{i\alpha} + a_{pq}e^{-i\alpha}\right) + \\
& 2\left(\sinh^4(y)\,|d|^2 - \frac{i}{2}\sinh^2(y)\sinh(2y)\,d\overline{\xi}\right) + \\
& 2\left(\frac{i}{2}\sinh^2(y)\sinh(2y)\,\overline{d}\xi + \frac{1}{4}\sinh^2(2y)\,|\xi|^2\right) \\
= \;& \ldots + 2\left(\sinh^4(y) + \frac{1}{4}\sinh^2(2y) + \sinh^2 y\right)\left(|d|^2 + |\xi|^2\right) + \\
& \frac{1}{2i}(4\left(-\sinh^2(y)\sinh(2y) - \frac{1}{2}\sinh(2y)\right)d\overline{\xi} - \\
& \left(\sinh^2(y)\sinh(2y) + \frac{1}{2}\sinh(2y)\right)\overline{d}\xi \\
= \;& \|A\|^2 + (\cosh(2y) - 1)\,G + 2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right) + \\
& \sinh^2(2y)\left(|d|^2 + |\xi|^2\right) - \sinh(4y)\,\Im\left(\overline{d}\xi\right).
\end{aligned}
$$

□

Now remember that we want to obtain the (global) minima of Expression 5.3 or at least approximations of them. For this purpose, we first differentiate with respect to α and y. Unfortunately it will turn out that there are no closed form solutions for the stationary points of 5.3. But with the help of linear approximations we manage to compute approximations of the stationary points of 5.3. Afterwards we show that these approximations are in fact approximations of (global) minima and that their choice guarantees convergence of the algorithm.

We begin with α:

$$
\begin{aligned}
\frac{\partial}{\partial\alpha}\left\|A'\right\|^2 &= \frac{\partial}{\partial\alpha}\left(\|A\|_F^2 + (\cosh(2y) - 1)\,G + 2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right)\right) + \\
&\quad \frac{\partial}{\partial\alpha}\left(\sinh^2(2y)\left(|d|^2 + |\xi|^2\right) - \sinh(4y)\,\Im\left(\overline{d}\xi\right)\right) \\
&= \frac{\partial}{\partial\alpha}\left(2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right) + \sinh^2(2y)\,|\xi|^2 - \sinh(4y)\,\Im\left(\overline{d}\xi\right)\right).
\end{aligned}
$$

If we use the linear approximation

$$2\sinh(2y) \approx \sinh(4y)$$

and Equality 5.2 we can further simplify and arrive at

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\left\|A'\right\|^2 &= \frac{\partial}{\partial\alpha}\left(2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right)+\sinh^2(2y)\,|\xi|^2-\sinh(4y)\,\Im\left(\bar{d}\xi\right)\right)\\
&\approx \sinh(2y)\,\frac{\partial}{\partial\alpha}\left(2\Im\left(e^{-i\alpha}K\right)-2\Im\left(\bar{d}\xi\right)+\sinh(2y)\,|\xi|^2\right)\\
&= \sinh(2y)\,\frac{\partial}{\partial\alpha}\left(2\Im\left(e^{-i\alpha}c\right)+\sinh(2y)\,|\xi|^2\right)\\
&= \sinh(2y)\,\frac{\partial}{\partial\alpha}\left(2\left(\frac{e^{-i\alpha}c-e^{i\alpha}\bar{c}}{2i}\right)+\sinh(2y)\,|\xi|^2\right)\\
&= \sinh(2y)\left(-2i\left(\frac{e^{-i\alpha}c+e^{i\alpha}\bar{c}}{2i}\right)+\sinh(2y)\,\frac{\partial}{\partial\alpha}\,|\xi|^2\right)\\
&= \sinh(2y)\left(-2\Re\left(e^{-i\alpha}c\right)+\sinh(2y)\,\frac{\partial}{\partial\alpha}\xi\bar{\xi}\right)\\
&= \sinh(2y)\left(-2\Re\left(e^{-i\alpha}c\right)+\sinh(2y)\,\frac{\partial}{\partial\alpha}\left(a_{qp}e^{i\alpha}+a_{pq}e^{-i\alpha}\right)\left(\bar{a}_{qp}e^{-i\alpha}+\bar{a}_{pq}e^{i\alpha}\right)\right)\\
&= \sinh(2y)\left(-2\Re\left(e^{-i\alpha}c\right)+\sinh(2y)\,\frac{\partial}{\partial\alpha}\left(a_{qp}\bar{a}_{pq}e^{2i\alpha}+a_{pq}\bar{a}_{qp}e^{-2i\alpha}\right)\right)\\
&= \sinh(2y)\left(-2\Re\left(e^{-i\alpha}c\right)+\sinh(2y)\,2i\left(a_{qp}\bar{a}_{pq}e^{2i\alpha}-a_{pq}\bar{a}_{qp}e^{-2i\alpha}\right)\right)\\
&= -2\sinh(2y)\left(\Re\left(e^{-i\alpha}c\right)+2\sinh(2y)\,\Im\left(a_{qp}\bar{a}_{pq}e^{2i\alpha}\right)\right). \qquad (5.4)
\end{aligned}$$

In practice is is important that an approximation of α can be computed in an efficient way. For this reason we only set the first summand of 5.4 to zero and then show later on that this simplified choice will still guarantee a decrease in the departure from normality, which is at the moment not directly obvious. We compute

$$\begin{aligned}
\Re\left(e^{-i\alpha}c\right) &= 0\\
\alpha &= \arg(c)-\frac{\pi}{2}.
\end{aligned}$$

So let us now broaden our view to the case of n matrices in which we are actually interested. We get

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\sum_{h=1}^{n}\left\|A_h'\right\|^2 &= \sum_{h=1}^{n}\frac{\partial}{\partial\alpha}\left\|A_h'\right\|^2\\
&\approx 2\sum_{h=1}^{n}\left(-\sinh(2y)\left(\Re\left(e^{-i\alpha}c_h\right)+2\sinh(2y)\,\Im\left(a_{h,qp}\bar{a}_{h,pq}e^{2i\alpha}\right)\right)\right).
\end{aligned}$$

Here we obtain the following estimate for α:

$$\begin{aligned}
\sum_{h=1}^{n}\Re\left(e^{-i\alpha}c_h\right) &= 0\\
\Re\left(e^{-i\alpha}\sum_{h=1}^{n}c_h\right) &= 0\\
\alpha &= \arg\left(\sum_{h=1}^{n}c_h\right)-\frac{\pi}{2}.
\end{aligned}$$

Now we repeat the same process for y for a single matrix:

$$\begin{aligned}\frac{\partial}{\partial y}\left\|A'\right\|^2 &= \frac{\partial}{\partial y}\left(\|A\|^2+(\cosh(2y)-1)\,G+2\sinh(2y)\,\Im\left(e^{-i\alpha}K\right)\right)+\\ &\quad \frac{\partial}{\partial y}\left(\sinh^2(2y)\left(|d|^2+|\xi|^2\right)-\sinh(4y)\,\Im\left(\overline{d}\xi\right)\right)\\ &= 2\sinh(2y)\,G+4\cosh(2y)\,\Im\left(e^{-i\alpha}K\right)+\\ &\quad 2\sinh(4y)\left(|d|^2+|\xi|^2\right)-4\cosh(4y)\,\Im\left(\overline{d}\xi\right). \qquad (5.5)\end{aligned}$$

If we use the linear approximations

$$\begin{aligned}4\sinh(y) &\approx 2\sinh(2y)\approx\sinh(4y)\\ \cosh(y) &\approx \cosh(2y)\approx\cosh(4y)\end{aligned}$$

to further simplify 5.5, we obtain:

$$\frac{\partial}{\partial y}\left\|A'\right\|^2\approx 4\sinh(y)\left(G+2\left(|d|^2+|\xi|^2\right)\right)+4\cosh(y)\left(\Im\left(e^{-i\alpha}K\right)-\Im\left(\overline{d}\xi\right)\right).$$

So the derivative is approximately zero if

$$\sinh(y)\left(G+2\left(|d|^2+|\xi|^2\right)\right) = -\cosh(y)\left(\Im\left(e^{-i\alpha}K\right)-\Im\left(\overline{d}\xi\right)\right)$$

and consequently if

$$\frac{\sinh(y)}{\cosh(y)} = \tanh(y) = \frac{\Im\left(\overline{d}\xi\right)-\Im\left(e^{-i\alpha}K\right)}{G+2\left(|d|^2+|\xi|^2\right)}$$

holds. If we use equation 5.2 again and substitute $\alpha=\arg(c)-\frac{\pi}{2}$ we obtain

$$\begin{aligned}\Im\left(\overline{d}\xi\right)-\Im\left(e^{-i\alpha}K\right) &= -\Im\left(e^{-i\alpha}c\right)\\ &= \Im\left(e^{-i\left(\arg(c)-\frac{\pi}{2}\right)}c\right)=-|c|\end{aligned}$$

and thus

$$\tanh(y) = \frac{-|c|}{G+2\left(|d|^2+|\xi|^2\right)}.$$

For the case of n matrices we obtain

$$\begin{aligned}\frac{\partial}{\partial y}\sum_{h=1}^{n}\left\|A'_h\right\|^2 &= \sum_{h=1}^{n}\frac{\partial}{\partial y}\left\|A'_h\right\|^2\\ &\approx 4\sinh(y)\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)+\\ &\quad 4\cosh(y)\sum_{h=1}^{n}\left(\Im\left(e^{-i\alpha}c_h\right)\right).\end{aligned}$$

This derivative is approximately zero if

$$\begin{aligned}
4\sinh(y)\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right) &= -4\cosh(y)\sum_{h=1}^{n}\left(\Im\left(e^{-i\alpha}c_h\right)\right)\\
\tanh(y) &= \frac{-\sum_{h=1}^{n}\left(\Im\left(e^{-i\alpha}c_h\right)\right)}{\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)}\\
\tanh(y) &= \frac{-\Im\left(e^{-i\alpha}\right)\sum_{h=1}^{n}\left(c_h\right)}{\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)}\\
\tanh(y) &= \frac{-\left|\sum_{h=1}^{n}c_h\right|}{\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)}.
\end{aligned}$$

Now that we have explained how we have obtained our values of α and y, we show that these choices of α and y will always lead to $\sum_{h=1}^{n}\|A_h'\|_F^2 \leq \sum_{h=1}^{n}\|A_h\|_F^2$. Recall from Corollary 5.3.6 that this implies a reduction in the common squared departure from normality of the matrices $A_1,...,A_n$. Additionally, we prove a (crude) lower bound for $\sum_{h=1}^{n}\|A_h\|_F^2-\sum_{h=1}^{n}\|A_h'\|_F^2$. We use similar arguments like Eberlein in [59], however, generalised to the setting of n matrices.

Theorem 5.3.17. *Let $A_1,...,A_n \in \mathrm{Mat}_m(\mathbb{C})$, let $1 \leq p < q \leq m$, let $d_h = a_{h,pp}-a_{h,qq}$, and let G_h and c_h as in Definition 5.3.14. Furthermore, let*

$$\alpha = \arg\left(\sum_{h=1}^{n}c_h\right)-\frac{\pi}{2}.$$

If $\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)=0$ then let $y=0$. If $\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\neq 0$ then let

$$\tanh(y) = \frac{-\left|\sum_{h=1}^{n}c_h\right|}{\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)}$$

with $\xi_h = e^{i\alpha}a_{h,qp}+e^{-i\alpha}a_{h,pq}$.
For theses choices of α and y, the inequality

$$\sum_{h=1}^{n}\|A_h'\|_F^2 \leq \sum_{h=1}^{n}\|A_h\|_F^2$$

holds, where $A_h' = S^{-1}A_hS$ with the definition of S given in 5.3.8.

Proof. Let us first introduce the definition $r_h = \Im\left(\overline{d}_h\xi_h\right)$ in order to keep our notation simple.

As $\alpha \in \mathbb{R}$ we can write

$$\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right) \overset{5.2}{=} \sum_{h=1}^{n}\Im\left(e^{-i\alpha}c_h\right) = \sum_{h=1}^{n}\left(\sin(\alpha)\Re(c_h)-\cos(\alpha)\Im(c_h)\right).$$

We compute

$$\begin{aligned}
\triangle_E &:= \sum_{h=1}^{n} \|A_h\|^2 - \sum_{h=1}^{n} \|A_h^{'}\|^2 \\
&= \sum_{h=1}^{n} \left(- \left(\cosh\left(2y\right) - 1 \right) G_h - 2 \sinh\left(2y\right) \Im\left(e^{-i\alpha} K_h \right) \right) + \\
&\quad \sum_{h=1}^{n} \left(- \sinh^2\left(2y\right) \left(|d_h|^2 + |\xi_h|^2 \right) + \sinh\left(4y\right) r_h \right) \\
&\geq \sinh\left(2y\right) \\
&\quad \sum_{h=1}^{n} \left(2 \cosh\left(2y\right) r_h - 2\Im\left(e^{-i\alpha} K_h \right) - \sinh\left(2y\right) \left(|d_h|^2 + |\xi_h|^2 + \frac{1}{2} G_h \right) \right)
\end{aligned}$$

because $\cosh\left(2y\right) - 1 \leq \frac{1}{4}\left(\cosh\left(4y\right) - 1\right) = \frac{1}{2}\sinh^2\left(2y\right)$. As $\sinh\left(2y\right) = 2\tanh\left(y\right)\cosh^2\left(y\right)$ holds, $\cosh^2\left(y\right) \geq 1$, and

$$\tanh\left(y\right) = \frac{\sum_{h=1}^{n} \left(r_h - \Im\left(e^{-i\alpha} K_h \right) \right)}{\sum_{h=1}^{n} \left(G_h + 2\left(|d_h|^2 + |\xi_h|^2 \right) \right)},$$

we obtain

$$\begin{aligned}
\Delta_E &\geq 2\tanh\left(y\right)\cosh^2\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(2\cosh\left(2y\right) r_h - 2\Im\left(e^{-i\alpha} K_h \right) - \tanh\left(y\right)\cosh^2\left(y\right) \left(2\left(|d_h|^2 + |\xi_h|^2 \right) + G_h \right) \right) \\
&= 2\tanh\left(y\right)\cosh^2\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(2\cosh\left(2y\right) r_h - 2\Im\left(e^{-i\alpha} K_h \right) - \left(r_h - \Im\left(e^{-i\alpha} K_h \right) \right) \cosh^2\left(y\right) \right) \\
&\geq 2\tanh\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(2\cosh\left(2y\right) r_h - 2\Im\left(e^{-i\alpha} K_h \right) - \left(r_h - \Im\left(e^{-i\alpha} K_h \right) \right) \cosh^2\left(y\right) \right) \\
&= 2\tanh\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(2r_h \cosh\left(2y\right) - 2\Im\left(e^{-i\alpha} K_h \right) - \frac{1}{2}\left(\cosh\left(2y\right) + 1 \right) \left(r_h - \Im\left(e^{-i\alpha} K_h \right) \right) \right) \\
&= 2\tanh\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(2r_h \left(\frac{1}{2} + \frac{3}{4}\left(\cosh\left(2y\right) - 1 \right) \right) - 2\Im\left(e^{-i\alpha} K_h \right) \left(\frac{1}{2} - \frac{1}{4}\left(\cosh\left(2y\right) - 1 \right) \right) \right) \\
&= 2\tanh\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(r_h + \frac{3}{2} r_h \left(\cosh\left(2y\right) - 1 \right) - \Im\left(e^{-i\alpha} K_h \right) + \frac{1}{2}\Im\left(e^{-i\alpha} K_h \right) \left(\cosh\left(2y\right) - 1 \right) \right) \\
&= 2\tanh\left(y\right) \\
&\quad \sum_{h=1}^{n} \left(r_h - \Im\left(e^{-i\alpha} K_h \right) + \frac{1}{2}\left(\cosh\left(2y\right) - 1 \right) \left(3r_h + \Im\left(e^{-i\alpha} K_h \right) \right) \right)
\end{aligned}$$

$$
\begin{aligned}
&= 2\tanh(y)\sum_{h=1}^{n}\left(\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)+\sinh^2(y)\left(3r_h+\Im\left(e^{-i\alpha}K_h\right)\right)\right)\\
&\geq 2\tanh(y)\sum_{h=1}^{n}\left(\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)-\sinh^2(y)\left|3r_h+\Im\left(e^{-i\alpha}K_h\right)\right|\right)\\
&= 2\tanh(y)\left(\left|\sum_{h=1}^{n}c_h\right|-\sinh^2(y)\sum_{h=1}^{n}\left|3r_h+\Im\left(e^{-i\alpha}K_h\right)\right|\right).
\end{aligned}
$$

We know that

$$
\begin{aligned}
\sinh^2(y) &= \frac{\tanh^2(y)}{1-\tanh^2(y)}\\
&= \frac{\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2}\Bigg/\\
&\quad\left(1-\frac{\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2}\right)\\
&= \frac{\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2}\Bigg/\\
&\quad\left(\frac{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2}\right)\\
&= \frac{\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}.
\end{aligned}
$$

Therefore, we arrive at

$$
\begin{aligned}
&\sinh^2(y)\sum_{j=1}^{n}\left|3r_j+\Im\left(e^{-i\alpha}K_j\right)\right| =\\
&= \frac{\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2\sum_{j=1}^{n}\left|3r_j+\Im\left(e^{-i\alpha}K_j\right)\right|}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2}\\
&= \left|\sum_{h=1}^{n}c_h\right|\frac{\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\sum_{j=1}^{n}\left|3r_j+\Im\left(e^{-i\alpha}K_j\right)\right|}{\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\left(\sum_{h=1}^{n}\left(r_h-\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2} \qquad (5.6)
\end{aligned}
$$

Note that

$$
|2r_h| = \left|2\Im\left(\overline{d}_h\xi_h\right)\right| = \left|i\left(d_h\overline{\xi}_h-\overline{d}_h\xi_h\right)\right| \leq |d_h|^2+|\xi_h|^2
$$

and

$$
\left|2\Im\left(e^{-i\alpha}K_h\right)\right| \leq G_h.
$$

This can easily be verified if we let $d_h = a+bi$ and $\xi_h = c+di$ with $a,b,c,d\in\mathbb{R}$. Then we

compute

$$\begin{aligned} i\left(d_h\overline{\xi}_h - \overline{d}_h\xi_h\right) &= \left((a+bi)(c-di) - (a-bi)(c+di)\right)i \\ &= (ac - adi + bci + bd - ac - adi + bci - bd)\,i \\ &= 2(ad-bc) \end{aligned}$$

$$\begin{aligned} 0 \leq (a-d)^2 + (b+c)^2 &= a^2 - 2ad + d^2 + b^2 + 2bc + c^2 = |d_h|^2 + |\xi_h|^2 - 2(ad-bc) \\ 0 \leq (a+d)^2 + (b-c)^2 &= a^2 + 2ad + d^2 + b^2 - 2bc + c^2 = |d_h|^2 + |\xi_h|^2 - 2(bc-ad) \\ 2(ad-bc) &\leq |d_h|^2 + |\xi_h|^2 \\ 2(bc-ad) &\leq |d_h|^2 + |\xi_h|^2 . \end{aligned}$$

Additionally, for the case of several matrices we observe that

$$\left|2\sum_{h=1}^{n} r_h\right| \leq \sum_{h=1}^{n} |2r_h| \leq \sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2\right)$$

and

$$\left|2\sum_{h=1}^{n} \Im\left(e^{-i\alpha}K_h\right)\right| \leq \sum_{h=1}^{n} \left|2\Im\left(e^{-i\alpha}K_h\right)\right| \leq \sum_{h=1}^{n} G_h.$$

As a consequence we have

$$\left|2\sum_{h=1}^{n} \left(r_h - \Im\left(e^{-i\alpha}K_h\right)\right)\right| \leq \sum_{h=1}^{n} \left|2r_h - 2\Im\left(e^{-i\alpha}K_h\right)\right| \leq \sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2 + G_h\right)$$

and

$$\left|\sum_{h=1}^{n} \left(6r_h + 2\Im\left(e^{-i\alpha}K_h\right)\right)\right| \leq \sum_{h=1}^{n} \left|6r_h + 2\Im\left(e^{-i\alpha}K_h\right)\right| \leq \sum_{h=1}^{n} \left(3\left(|d_h|^2 + |\xi_h|^2\right) + G_h\right).$$

For the numerator of formula 5.6 we have

$$\begin{aligned} &\sum_{h=1}^{n} \left(r_h - \Im\left(e^{-i\alpha}K_h\right)\right) \sum_{j=1}^{n} \left|3r_j + \Im\left(e^{-i\alpha}K_j\right)\right| \leq \\ \leq\ & \frac{1}{4}\sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2 + G_h\right) \sum_{j=1}^{n} \left(3\left(|d_j|^2 + |\xi_j|^2\right) + G_j\right) \\ =\ & \frac{1}{4}\left(3\left(\sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2\right)\right)^2 + 4\sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2\right) \sum_{j=1}^{n} G_j + \left(\sum_{h=1}^{n} G_h\right)^2\right) \\ \leq\ & \frac{1}{4}\left(4\left(\sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2\right)\right)^2 + 4\sum_{h=1}^{n} \left(|d_h|^2 + |\xi_h|^2\right) \sum_{j=1}^{n} G_j + \left(\sum_{h=1}^{n} G_h\right)^2\right) \\ =\ & \frac{1}{4}\left(\sum_{h=1}^{n} \left(G_h + 2\left(|d_h|^2 + |\xi_h|^2\right)\right)\right)^2 . \end{aligned}$$

Further analysis of the denominator of formula 5.6 yields

$$\begin{aligned}
& \left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\frac{1}{4}\left(\sum_{h=1}^{n}\left(2r_h-2\Im\left(e^{-i\alpha}K_h\right)\right)\right)^2 \geq \\
\geq & \left(\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2-\frac{1}{4}\left(\sum_{h=1}^{n}\left(|d_h|^2+|\xi_h|^2+G_h\right)\right)^2\right) \\
= & \frac{3}{4}\left(\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)\right)^2.
\end{aligned}$$

If we put all results together we obtain

$$\sinh^2(y)\sum_{j=1}^{n}\left|3r_j+\Im\left(e^{-i\alpha}K_j\right)\right| \geq \frac{1}{3}$$

and finally

$$\begin{aligned}
\Delta_E \geq & \tanh(y)\left(\left|\sum_{h=1}^{n}c_h\right|-\left|\sum_{h=1}^{n}c_h\right|\frac{1}{3}\right)=\frac{2}{3}\tanh(y)\left|\sum_{h=1}^{n}c_h\right| \\
= & \frac{1}{3}\frac{\left|\sum_{h=1}^{n}c_h\right|^2}{\sum_{h=1}^{n}\left(G_h+2\left(|d_h|^2+|\xi_h|^2\right)\right)} \geq \frac{1}{3}\frac{\left|\sum_{h=1}^{n}c_h\right|^2}{\sum_{h=1}^{n}\|A_h\|} \geq 0.
\end{aligned}$$

□

5.3.3 Choice of Parameters in the Unitary Transformation

Now that we have demonstrated how to choose the parameters α and y in the shear transformation we focus on the unitary transformation and its parameters θ and φ. Fortunately, we are in a more favourable situation in the sense that these can be computed in an optimal way using a closed form expression.

Before we begin with the main proof we show a lemma that relates the departure from normality of $A''=U^*A'U$, where U is a unitary transformation matrix as defined in 5.3.9, to the departure of normality of A' its entries and the parameters θ and φ of U. Note, that we use similar arguments like Goldstine and Horwitz in [61], but generalised to the setting of n matrices.

Lemma 5.3.18. *Let $A'\in\mathrm{Mat}_m(\mathbb{C})$, let $1\leq p<q\leq m$, let $U\in\mathrm{Mat}_m(\mathbb{C})$ be a unitary rotation matrix with parameters θ and φ, and let $A''=U^*A'U$. Then*

$$\Delta_D^2\left(A''\right) = \triangle_D^2\left(A'\right)+\frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-\frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right), \tag{5.7}$$

where $d'=a'_{pp}-a'_{qq}$ and $\xi'=e^{i\varphi}a'_{qp}+e^{-i\varphi}a'_{pq}$.

Proof. Let us quickly recall that $Q = \frac{1}{2}\sin(2\theta)\, d' + \sin^2(\theta)\, \xi'$.

$$
\begin{aligned}
\Delta_D^2(A'') &= \sum_{j\neq p,q} \left|\cos(\theta)\, a'_{pj} + e^{i\varphi}\sin(\theta)\, a'_{qj}\right|^2 + \sum_{j\neq p,q} \left|-e^{-i\varphi}\sin(\theta)\, a'_{pj} + \cos(\theta)\, a'_{qj}\right|^2 + \\
&\quad \sum_{j\neq p,q} \left|\cos(\theta)\, a'_{jp} + e^{-i\varphi}\sin(\theta)\, a'_{jq}\right|^2 + \sum_{j\neq p,q} \left|-e^{i\varphi}\sin(\theta)\, a'_{jp} + \cos(\theta)\, a'_{jq}\right|^2 + \\
&\quad \left|a'_{pq} - e^{i\varphi}Q\right|^2 + \left|a'_{qp} - e^{-i\varphi}Q\right|^2 + \sum_{j,i\neq p,q,j\neq i} \left|a'_{ij}\right|^2 \\
&= \ldots + \left(a'_{pq} - e^{i\varphi}Q\right)\left(\bar{a}'_{pq} - e^{-i\varphi}\overline{Q}\right) + \\
&\quad \left(a'_{qp} - e^{-i\varphi}Q\right)\left(\bar{a}'_{qp} - e^{i\varphi}\bar{Q}\right) + \sum_{j,i\neq p,q,j\neq i} \left|a'_{ij}\right|^2 \\
&= \ldots + \left|a'_{pq}\right|^2 - e^{-i\varphi}a'_{pq}\overline{Q} - e^{i\varphi}\bar{a}'_{pq}Q + |Q|^2 + \left|a'_{qp}\right|^2 - \\
&\quad e^{i\varphi}a'_{qp}\bar{Q} - e^{-i\varphi}\bar{a}'_{qp}Q + |Q|^2 + \sum_{j,i\neq p,q,j\neq i} \left|a'_{ij}\right|^2 \\
&= \sum_{j\neq p,q} \left(\cos^2(\theta)\left|a'_{pj}\right|^2 + e^{-i\varphi}\cos(\theta)\sin(\theta)\, a'_{pj}\bar{a}'_{qj}\right) + \\
&\quad \sum_{j\neq p,q} \left(e^{i\varphi}\cos(\theta)\sin(\theta)\, \bar{a}'_{pj}a'_{qj} + \sin^2(\theta)\left|a'_{qj}\right|^2\right) + \\
&\quad \sum_{j\neq p,q} \left(\sin^2(\theta)\left|a'_{pj}\right|^2 - e^{-i\varphi}\cos(\theta)\sin(\theta)\, a'_{pj}\bar{a}'_{qj}\right) + \\
&\quad \sum_{j\neq p,q} \left(-e^{i\varphi}\cos(\theta)\sin(\theta)\, \bar{a}'_{pj}a'_{qj} + \cos^2(\theta)\left|a'_{qj}\right|^2\right) + \\
&\quad \sum_{j\neq p,q} \left(\cos^2(\theta)\left|a'_{jp}\right|^2 + e^{i\varphi}\cos(\theta)\sin(\theta)\, a'_{jp}\bar{a}'_{jq}\right) + \\
&\quad \sum_{j\neq p,q} \left(e^{-i\varphi}\cos(\theta)\sin(\theta)\, a'_{jq}\bar{a}'_{jp} + \sin^2(\theta)\left|a'_{jq}\right|^2\right) + \\
&\quad \sum_{j\neq p,q} \left(\sin^2(\theta)\left|a'_{jp}\right|^2 - e^{i\varphi}\cos(\theta)\sin(\theta)\, a'_{jp}\bar{a}'_{jq}\right) + \\
&\quad \sum_{j\neq p,q} \left(-e^{-i\varphi}\cos(\theta)\sin(\theta)\, \bar{a}'_{jp}a'_{jq} + \cos^2(\theta)\left|a'_{jq}\right|^2\right) + \ldots \\
&= -e^{-i\varphi}a'_{pq}\overline{Q} - e^{i\varphi}\bar{a}'_{pq}Q - e^{i\varphi}a'_{qp}\bar{Q} - e^{-i\varphi}\bar{a}'_{qp}Q + 2|Q|^2 + \sum_{j\neq i} \left|a'_{ij}\right|^2 \\
&= -\left(e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq}\right)\overline{Q} - \left(e^{-i\varphi}\bar{a}'_{qp} + e^{i\varphi}\bar{a}'_{pq}\right)Q + 2|Q|^2 + \sum_{j\neq i} \left|a'_{ij}\right|^2 \\
&= \sum_{j\neq i} \left|a'_{ij}\right|^2 + 2|Q|^2 - 2\Re\left(\xi'\overline{Q}\right) \\
&= \sum_{j\neq i} \left|a'_{ij}\right|^2 + \left(\sin(2\theta)\sin^2(\theta)\, \bar{d}'\xi' + 2\sin^4(\theta)\left|\xi'\right|^2 + \right.
\end{aligned}
$$

$$
\begin{aligned}
& \sin(2\theta)\sin^2(\theta)\,\bar{d}'\xi' + 2\sin^4(\theta)\left|\xi'\right|^2\Big) - \\
& \left(\left(\frac{1}{2}\sin(2\theta)\,\bar{d}'\xi' + \sin^2(\theta)\left|\xi'\right|^2\right) + \left(\frac{1}{2}\sin(2\theta)\,d'\bar{\xi}' + \sin^2(\theta)\left|\xi'\right|^2\right)\right) \\
= & \sum_{j\neq i}\left|a_{ij}'\right|^2 + \frac{1}{2}\sin^2(2\theta)\left|d'\right|^2 - 2\left(\sin^2(\theta) - \sin^4(\theta)\right)\left|\xi'\right|^2 + \\
& \sin(2\theta)\left(\sin^2(\theta)\,d'\bar{\xi}' + \sin^2(\theta)\,\bar{d}'\xi' - \frac{1}{2}\bar{d}'\xi' - \frac{1}{2}d'\bar{\xi}'\right) \\
= & \sum_{j\neq i}\left|a_{ij}'\right|^2 + \frac{1}{2}\sin^2(2\theta)\left|d'\right|^2 - \frac{1}{2}\sin^2(2\theta)\left|\xi'\right|^2 + \\
& 2\sin(2\theta)\left(\left(\sin^2(\theta) - \frac{1}{2}\right)\left(\bar{d}'\xi' + d'\bar{\xi}'\right)\frac{1}{2}\right) \\
= & \sum_{j\neq i}\left|a_{ij}'\right|^2 + \frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2 - \left|\xi'\right|^2\right) - \frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right) \\
= & \triangle_D^2\left(A'\right) + \frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2 - \left|\xi'\right|^2\right) - \frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right). \qquad (5.8)
\end{aligned}
$$

□

Next we assume that φ is fixed and compute the stationary points of $\sum_{h=1}^n \Delta_D^2\left(A_h''\right)$ with respect to θ and decide with the help of the usual criteria which stationary points are in fact (global) minima.

Lemma 5.3.19. *Let $A_1', ..., A_n' \in \mathrm{Mat}_m(\mathbb{C})$, let $1 \leq p < q \leq m$, let $U \in \mathrm{Mat}_m(\mathbb{C})$ be a unitary rotation matrix with parameters θ and φ, and let $A_h'' = U^* A_h' U$. Depending on $\mathrm{sgn}\left(\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)\right)$ we can establish three cases when $\sum_{h=1}^n \Delta_D^2\left(A''\right)$ will assume a global minimum with respect to θ:*

1. *If $\mathrm{sgn}\left(\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)\right) = 0$, then let*

$$\theta = \mathrm{sgn}\left(\sum_{h=1}^n \Re\left(\bar{d}_h'\xi_h'\right)\right)\frac{\pi}{8}.$$

2. *If $\mathrm{sgn}\left(\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)\right) = 1$, then let*

$$\theta = \frac{1}{4}\arctan\left(\frac{2\sum_{h=1}^n \Re\left(\bar{d}_h'\xi_h'\right)}{\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)}\right).$$

3. *If $\mathrm{sgn}\left(\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)\right) = -1$, then let*

$$\theta = \frac{1}{4}\arctan\left(\frac{2\sum_{h=1}^n \Re\left(\bar{d}_h'\xi_h'\right)}{\sum_{h=1}^n \left(\left|d_h'\right|^2 - \left|\xi_h'\right|^2\right)}\right) + \frac{\pi}{4}.$$

Proof. Initially, we investigate the case of one matrix and then afterwards we generalise this result to n matrices. So let us start with a single matrix. For this purpose, we take the derivative of $\Delta_D^2(A'')$ (see Lemma 5.3.18) with respect to θ and obtain:

$$\begin{aligned}\frac{\partial}{\partial\theta}\Delta_D^2(A'') &= \frac{\partial}{\partial\theta}\left(\sum_{j\neq i}\left|a_{ij}'\right|^2+\frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-\frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right)\\ &= \sin(4\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-2\cos(4\theta)\,\Re\left(\bar{d}'\xi'\right).\end{aligned}$$

For a minimum to occur, one requirement is that the first derivative equals zero.

$$0 = \sin(4\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-2\cos(4\theta)\,\Re\left(\bar{d}'\xi'\right)$$

We will have to take care of essentially two cases: Let us first assume that $|d'|^2-|\xi'|^2=0$. Then it is easy to see that the values

$$\theta_{1,2}=\pm\frac{\pi}{8}$$

are potential minima. The second derivative of $\Delta_D^2(A'')$ with respect to θ is given by

$$\begin{aligned}\left(\frac{\partial}{\partial\theta}\right)^2\Delta_D^2(A'') &= \frac{\partial}{\partial\theta}\left(-2\cos(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right)\\ &= 8\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right).\end{aligned}$$

A minimum is thus obtained for $\theta=\operatorname{sgn}\left(\Re\left(\bar{d}'\xi'\right)\right)\frac{\pi}{8}$.

Now we treat the general case $|d'|^2-|\xi'|^2\neq 0$. We observe that the equation

$$\frac{\sin(4\theta)}{\cos(4\theta)}=\tan(4\theta)=\frac{2\Re\left(\bar{d}'\xi'\right)}{|d'|^2-|\xi'|^2} \tag{5.9}$$

needs to be satisfied.

Using the equality $\sin(\arctan(x))=\frac{x}{\sqrt{1+x^2}}$ we can derive a relation for $\sin(4\theta)$. We obtain

$$\begin{aligned}\sin(4\theta) &= \frac{2\Re\left(\bar{d}'\xi'\right)}{|d'|^2-|\xi'|^2}\Bigg/\sqrt{1+\left(\frac{2\Re\left(\bar{d}'\xi'\right)}{|d'|^2-|\xi'|^2}\right)^2}\\ &= \frac{2\Re\left(\bar{d}'\xi'\right)}{|d'|^2-|\xi'|^2}\Bigg/\sqrt{\frac{\left(|d'|^2-|\xi'|^2\right)^2+\left(\bar{d}'\xi'+d'\bar{\xi}'\right)^2}{\left(|d'|^2-|\xi'|^2\right)^2}}\\ &= \frac{2\Re\left(\bar{d}'\xi'\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{\sqrt{\left(|d'|^2-|\xi'|^2\right)^2+\left(\bar{d}'\xi'+d'\bar{\xi}'\right)^2}}\\ &= \frac{2\Re\left(\bar{d}'\xi'\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{\sqrt{|d'|^4+|\xi'|^4+\left|\bar{d}'\right|^2|\xi'|^2+|d'|^2\left|\bar{\xi}'\right|^2}}\\ &= \frac{2\Re\left(\bar{d}'\xi'\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{|d'^2+\xi'^2|}\end{aligned}$$

and additionally with the help of the equation $\cos(\arctan(x)) = \frac{1}{\sqrt{1+x^2}}$ we can derive a relation for $\cos(4\theta)$. We arrive at

$$\cos(4\theta) = \frac{\left(|d'|^2 - |\xi'|^2\right) \operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right)}{|d'^2 + \xi'^2|}. \tag{5.10}$$

As $\Delta_D^2(A'')$ is $\frac{\pi}{2}$ periodic in θ we only have to investigate one such interval. We obtain from 5.9

$$\theta_1 = \frac{1}{4}\arctan\left(\frac{2\Re\left(\bar{d}'\xi'\right)}{|d'|^2 - |\xi'|^2}\right) \in \left]-\frac{\pi}{8}, \frac{\pi}{8}\right[\tag{5.11}$$

as the first solution and $\theta_2 = \theta_1 + \frac{\pi}{4}$ as the second solution. With the help of the second derivative of $\Delta_D^2(A'')$ with respect to θ we can decide whether θ_1 or θ_2 is the minimum. So we compute

$$\begin{aligned}\left(\frac{\partial}{\partial\theta}\right)^2 \Delta_D^2(A'') &= \frac{\partial}{\partial\theta}\left(\sin(4\theta)\left(|d'|^2 - |\xi'|^2\right) - 2\cos(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right) \\ &= 4\left(\cos(4\theta)\left(|d'|^2 - |\xi'|^2\right) + 2\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right).\end{aligned}$$

We substitute θ_1 and obtain

$$\begin{aligned}&4\left(\cos(4\theta_1)\left(|d'|^2 - |\xi'|^2\right) + 2\sin(4\theta_1)\,\Re\left(\bar{d}'\xi'\right)\right) = \\ = \;& 4\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right)\left(\frac{\left(|d'|^2 - |\xi'|^2\right)^2}{|d'^2 + \xi'^2|} + \frac{4\Re\left(\bar{d}'\xi'\right)^2}{|d'^2 + \xi'^2|}\right).\end{aligned}$$

It is easy to see that the evaluations of the second derivative at θ_1 and θ_2 have always opposite signs as

$$\begin{aligned}&4\left(\cos(4\theta_2)\left(|d'|^2 - |\xi'|^2\right) + 2\sin(4\theta_2)\,\Re\left(\bar{d}'\xi'\right)\right) = \\ =\;& 4\left(\cos\left(4\left(\theta_1 + \frac{\pi}{4}\right)\right)\left(|d'|^2 - |\xi'|^2\right) + 2\sin\left(4\left(\theta_1 + \frac{\pi}{4}\right)\right)\Re\left(\bar{d}'\xi'\right)\right) \\ =\;& -4\left(\cos(4\theta_1)\left(|d'|^2 - |\xi'|^2\right) + 2\sin(4\theta_1)\,\Re\left(\bar{d}'\xi'\right)\right)\end{aligned}$$

holds. Thus we can conclude that θ_1 has to be the global minimum if $\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right) > 0$ and θ_2 if $\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right) < 0$.

This result can easily be generalised to the case of n matrices and we obtain

$$\frac{\partial}{\partial\theta}\sum_{h=1}^{n}\Delta_D^2\left(A_h''\right) = \sin(4\theta)\sum_{h=1}^{n}\left(|d_h'|^2 - |\xi_h'|^2\right) - 2\cos(4\theta)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right).$$

Now if $\sum_{h=1}^{n}\left(|d_h'|^2 - |\xi_h'|^2\right) = 0$ we obtain

$$0 = \cos(4\theta)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)$$

with

$$\theta = \operatorname{sgn}\left(\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right) \frac{\pi}{8}.$$

In case $\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right) \neq 0$ we compute

$$\tan(4\theta) = \frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)}{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)}.$$

Using again the equalities $\sin(\arctan(x)) = \frac{x}{\sqrt{1+x^2}}$ and $\cos(\arctan(x)) = \frac{1}{\sqrt{1+x^2}}$ we can derive relations for $\sin(4\theta)$ and $\cos(4\theta)$. We get

$$\begin{aligned}
\sin(4\theta) &= \frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)}{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)} \Bigg/ \sqrt{\left(1 + \left(\frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)}{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)}\right)^2\right)} \\
&= \frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)}{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)} \Bigg/ \sqrt{\left(\frac{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)^2 + 4\left(\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right)^2}{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)^2}\right)} \\
&= \frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right) \operatorname{sgn}\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)^2 + 4\left(\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right)^2}}
\end{aligned}$$

and

$$\cos(4\theta) = \frac{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right) \operatorname{sgn}\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)\right)^2 + 4\left(\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right)^2}}.$$

We obtain two solutions for $\frac{\partial}{\partial\theta}\sum_{h=1}^{n} \Delta_D^2\left(A''_h\right) = 0$, namely

$$\theta_1 = \frac{1}{4}\arctan\left(\frac{2\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)}{\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right)}\right) \in \left]-\frac{\pi}{8}, \frac{\pi}{8}\right[$$

and $\theta_2 = \theta_1 + \frac{\pi}{4}$. With the help of the second derivative we determine when θ_1 or θ_2 is a global minimum. Therefore, we compute

$$\begin{aligned}
\left(\frac{\partial}{\partial\theta}\right)^2 \sum_{h=1}^{n} \Delta_D^2\left(A''_h\right) &= \frac{\partial}{\partial\theta}\left(\sin(4\theta)\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right) - 2\cos(4\theta)\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right) \\
&= 4\left(\cos(4\theta)\sum_{h=1}^{n}\left(\left|d'_h\right|^2 - \left|\xi'_h\right|^2\right) + 2\sin(4\theta)\sum_{h=1}^{n} \Re\left(\bar{d}'_h \xi'_h\right)\right).
\end{aligned}$$

Next, we substitute θ_1 and obtain

$$\begin{aligned}
&4\left(\cos(4\theta_1)\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)+2\sin(4\theta_1)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)=\\
=\;&4\left(\frac{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2\operatorname{sgn}\left(\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}}\right)+\\
&+4\left(\frac{4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2\operatorname{sgn}\left(\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}}\right).
\end{aligned}$$

If we substitute θ_2 we observe that

$$\begin{aligned}
&4\left(\cos(4\theta_2)\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)+2\sin(4\theta_2)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)=\\
=\;&4\left(\cos\left(4\left(\theta_1+\frac{\pi}{4}\right)\right)\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)+2\sin\left(4\left(\theta_1+\frac{\pi}{4}\right)\right)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)\\
=\;&-4\left(\cos(4\theta_1)\sum_{h=1}^{n}\left(\left|d_h'\right|^2-\left|\xi_h'\right|^2\right)+2\sin(4\theta_1)\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right).
\end{aligned}$$

So we have shown that θ_1 is a global minimum if $\operatorname{sgn}\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)=1$ and θ_2 if $\operatorname{sgn}\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)=-1$.

□

Remark 5.3.20. For angles near 0 and π the numerical calculation of the arccos and for angles near $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ the calculation of arcsin becomes ill-conditioned. Consequently, arctan, or in fact atan2 (see [63, page 42]) which is available in many programming languages, should be used in an actual computer implementation.

We now repeat the same procedure for φ to obtain its optimal value. I.e. we compute the stationary points of $\sum_{h=1}^{n}\Delta_D^2(A_h'')$ with respect to φ and determine which stationary points are in fact global minima.

Lemma 5.3.21. *Let $A_1',\ldots,A_n'\in\mathrm{Mat}_m(\mathbb{C})$, let $1\le p<q\le m$, let $U\in\mathrm{Mat}_m(\mathbb{C})$ be a unitary rotation matrix with parameters θ and φ, and let $A_h''=U^*A_h'U$. Furthermore, let the parameter θ of U be chosen as proposed in Lemma 5.3.19. We can establish two cases when $\sum_{h=1}^{n}\Delta_D^2(A'')$ will assume a global minimum with respect to φ:*

1. *If $\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}'+a_{h,qp}'\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}\right)}\right)=0$, then*

$$\varphi=\operatorname{sgn}\left(\Im\left(\sum_{h=1}^{n}\left(a_{h,pq}'-a_{h,qp}'\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}\right)}\right)\right)\frac{\pi}{2}.$$

2. If $\Re\left(\sum_{h=1}^{n}\left(a'_{h,pq}+a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d'^2_h+4a'_{h,pq}a'_{h,qp}}\right)}\right)\neq 0$, *then*

$$\varphi=\arctan\left(\frac{\Im\left(\sum_{h=1}^{n}\left(a'_{h,pq}-a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d'^2_h+4a'_{h,pq}a'_{h,qp}}\right)}\right)}{\Re\left(\sum_{h=1}^{n}\left(a'_{h,pq}+a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d'^2_h+4a'_{h,pq}a'_{h,qp}}\right)}\right)}\right).$$

Proof. Once again, we first analyse the single matrix case. Later on we extend this result to n matrices. We start by computing the derivative of $\Delta_D^2\left(A''\right)$ (see 5.3.18) with respect to φ. Recall that $d'=a'_{pp}-a'_{qq}$ and $\xi'=e^{i\varphi}a'_{qp}+e^{-i\varphi}a'_{pq}$.

$$\begin{aligned}\frac{\partial}{\partial\varphi}\Delta_D^2\left(A''\right) &= \frac{\partial}{\partial\varphi}\left(\sum_{j\neq i}\left|a'_{ij}\right|^2+\frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-\frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right)\\ &= \frac{\partial}{\partial\varphi}\left(\frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2-\left|\xi'\right|^2\right)-\frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right).\end{aligned}$$

We once again distinguish a few different cases.
First of all we consider the case $d'=0$. We observe that $\operatorname{sgn}\left(\left|d'\right|^2-\left|\xi'\right|^2\right)\leq 0$. According to Lemma 5.3.19 we have to substitute $\theta=0$ if $|\xi'|=0$ and $\theta=\frac{1}{4}\arctan(0)+\frac{\pi}{4}=\frac{\pi}{4}$ if $|\xi'|\neq 0$ (see also 5.11). As an intermediate step we will show that $\Delta_D^2\left(A''\right)$ is differentiable with respect to φ. Let us now assume that $|\xi'|=0$, which implies that

$$\begin{aligned}0 &= \xi'=e^{i\varphi}a'_{qp}+e^{-i\varphi}a'_{pq}\\ &= (\cos(\varphi)+i\sin(\varphi))\,a'_{qp}+(\cos(\varphi)-i\sin(\varphi))\,a'_{pq}\\ &= \sin(\varphi)\left(a'_{qp}-a'_{pq}\right)i+\cos(\varphi)\left(a'_{qp}+a'_{pq}\right)\end{aligned}$$

We further obtain

$$\frac{\sin(\varphi)}{\cos(\varphi)} = \frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}\iff\varphi=\tan^{-1}\left(\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}\right). \tag{5.12}$$

Since φ is real, $\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}$ must also be a real number. This implies

$$\begin{aligned}0 &= -\Im\left(\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}\right)=-\frac{1}{2i}\left(\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}-\frac{\left(\bar{a}'_{qp}+\bar{a}'_{pq}\right)}{-i\left(\bar{a}'_{qp}-\bar{a}'_{pq}\right)}\right)\\ &= \frac{\left(\bar{a}'_{qp}-\bar{a}'_{pq}\right)\left(a'_{qp}+a'_{pq}\right)+\left(a'_{qp}-a'_{pq}\right)\left(\bar{a}'_{qp}+\bar{a}'_{pq}\right)}{2\left(a'_{qp}-a'_{pq}\right)\left(\bar{a}'_{qp}-\bar{a}'_{pq}\right)}\\ &= \frac{\Re\left(\left(\bar{a}'_{qp}-\bar{a}'_{pq}\right)\left(a'_{qp}+a'_{pq}\right)\right)}{\left|a'_{qp}-a'_{pq}\right|^2}\\ &= \frac{\Re\left(\left|a'_{qp}\right|^2-\left|a'_{pq}\right|^2+\bar{a}'_{qp}a'_{pq}-a'_{qp}\bar{a}'_{pq}\right)}{\left|a'_{qp}-a'_{pq}\right|^2}\\ &= \frac{\left|a'_{qp}\right|^2-\left|a'_{pq}\right|^2+\Re\left(\Im\left(\bar{a}'_{qp}a'_{pq}\right)2i\right)}{\left|a'_{qp}-a'_{pq}\right|^2}=\frac{\left|a'_{qp}\right|^2-\left|a'_{pq}\right|^2}{\left|a'_{qp}-a'_{pq}\right|^2}.\end{aligned}$$

So only if $\left|a'_{qp}\right| = \left|a'_{pq}\right|$, the expression $\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}$ is real and it is possible to choose φ as in 5.12 such that $|\xi'| = 0$. Let us further assume that $\left|a'_{qp}\right| = \left|a'_{pq}\right|$. If we now let $\varphi = \arctan\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)} := \varphi_0$ then $|\xi'|^2 = 0$, $\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right) = 0$ and according to Lemma 5.3.19 $\theta = 0 := \theta_1$. Whenever $\varphi \neq \arctan\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}$ then $\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right) < 0$ and we have to choose $\theta = \frac{1}{4}\arctan(0) + \frac{\pi}{4} = \frac{\pi}{4} := \theta_2$. We thus have to verify that $\Delta_D^2(A'')$ is differentiable at $\varphi = \arctan\frac{\left(a'_{qp}+a'_{pq}\right)}{i\left(a'_{qp}-a'_{pq}\right)}$. Note that $\Delta_D^2(A'')(\theta_1)(\varphi_0) = \Delta_D^2(A'')(\theta_2)(\varphi_0) = 0$. So since

$$\begin{aligned}
&\frac{\partial}{\partial\varphi}\left(\Delta_D^2\left(A''\right)(\theta_2)\right)(\varphi_0)\\
&= \lim_{\varphi\to\varphi_0}\frac{\Delta_D^2\left(A''_h\right)(\theta_2)(\varphi) - \Delta_D^2\left(A''_h\right)(\theta_2)(\varphi_0)}{\varphi-\varphi_0}\\
&= \lim_{\varphi\to\varphi_0}\frac{\Delta_D^2\left(A''_h\right)(\theta_2)(\varphi) - \Delta_D^2\left(A''_h\right)(\theta_1)(\varphi_0)}{\varphi-\varphi_0}
\end{aligned}$$

we note that for $d' = 0$, $\Delta_D^2(A'')$ is differentiable for all $\varphi_0 \in]-\pi,\pi]$. Since the limits agree, we need not distinguish between $|\xi'| = 0$ and $|\xi'| \neq 0$.
Now let $\theta = \theta_2 = \frac{\pi}{4}$ and let us further assume that no special relations exists between $\left|a'_{qp}\right|$ and $\left|a'_{pq}\right|$. We obtain $\sin^2(2\theta) = \sin^2\left(2\frac{\pi}{4}\right) = 1$ and consequently we get

$$\begin{aligned}
\frac{\partial}{\partial\varphi}\Delta_D^2\left(A''\right) &= \frac{\partial}{\partial\varphi}\left(\frac{1}{2}\sin^2(2\theta)\left(\left|d'\right|^2 - \left|\xi'\right|^2\right) - \frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right)\right)\\
&= -\frac{1}{2}\frac{\partial}{\partial\varphi}\left|\xi'\right|^2\\
&= -\frac{1}{2}\frac{\partial}{\partial\varphi}\left(\left(e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq}\right)\left(e^{-i\varphi}\bar{a}'_{qp} + e^{i\varphi}\bar{a}'_{pq}\right)\right)\\
&= -\frac{1}{2}\frac{\partial}{\partial\varphi}\left(\left|a'_{qp}\right|^2 + e^{2i\varphi}a'_{qp}\bar{a}'_{pq} + e^{-2i\varphi}\bar{a}'_{qp}a'_{pq} + \left|a'_{pq}\right|^2\right)\\
&= -i\left(e^{2i\varphi}a'_{qp}\bar{a}'_{pq} - e^{-2i\varphi}\bar{a}'_{qp}a'_{pq}\right)\\
&= -i\left(\left(\cos(2\varphi) + i\sin(2\varphi)\right)a'_{qp}\bar{a}'_{pq} - \left(\cos(2\varphi) - i\sin(2\varphi)\right)\bar{a}'_{qp}a'_{pq}\right).
\end{aligned}$$

Additionally, we compute the second derivative and obtain

$$\begin{aligned}
\left(\frac{\partial}{\partial\varphi}\right)^2\Delta_D^2\left(A''\right) &= 2\left(e^{2i\varphi}a'_{qp}\bar{a}'_{pq} + e^{-2i\varphi}\bar{a}'_{qp}a'_{pq}\right)\\
&= 2\left(\left(\cos(2\varphi) + i\sin(2\varphi)\right)a'_{qp}\bar{a}'_{pq} + \left(\cos(2\varphi) - i\sin(2\varphi)\right)\bar{a}'_{qp}a'_{pq}\right).
\end{aligned}$$

Next we want to determine the zero set of $\frac{\partial}{\partial\varphi}\Delta_D^2(A'')$ and then decide with the help of $\left(\frac{\partial}{\partial\varphi}\right)^2\Delta_D^2(A'')$ which are in fact local minima. Therefore, we compute

$$\begin{aligned}
0 &= -i\left(e^{2i\varphi}a'_{qp}\bar{a}'_{pq} - e^{-2i\varphi}\bar{a}'_{qp}a'_{pq}\right)\\
&= e^{2i\varphi}a'_{qp}\bar{a}'_{pq} - e^{-2i\varphi}\bar{a}'_{qp}a'_{pq}\\
&= \left(\left(\cos(2\varphi) + i\sin(2\varphi)\right)a'_{qp}\bar{a}'_{pq} - \left(\cos(2\varphi) - i\sin(2\varphi)\right)\bar{a}'_{qp}a'_{pq}\right)\\
i\sin(2\varphi)\left(\bar{a}'_{qp}a'_{pq} + a'_{qp}\bar{a}'_{pq}\right) &= \cos(2\varphi)\left(\bar{a}'_{qp}a'_{pq} - a'_{qp}\bar{a}'_{pq}\right). \qquad (5.13)
\end{aligned}$$

If $\Re\left(\bar{a}'_{qp}a'_{pq}\right) = 0$, then we have $0 = \cos(2\varphi)\left(\bar{a}'_{qp}a'_{pq} - a'_{qp}\bar{a}'_{pq}\right)$ so $\varphi_{1,2} = \pm\frac{\pi}{4}$ are potential minima. We evaluate $\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'')$ on φ_1 and obtain

$$\begin{aligned}\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'') &= 2\left(\left(\cos(2\varphi_1) + i\sin(2\varphi_1)\right)a'_{qp}\bar{a}'_{pq} + \left(\cos(2\varphi_1) - i\sin(2\varphi_1)\right)\bar{a}'_{qp}a'_{pq}\right)\\ &= -2i\left(a'_{qp}\bar{a}'_{pq} - \bar{a}'_{qp}a'_{pq}\right)\\ &= \frac{4}{2i}\left(a'_{qp}\bar{a}'_{pq} - \bar{a}'_{qp}a'_{pq}\right) = 4\Im\left(a'_{qp}\bar{a}'_{pq}\right).\end{aligned}$$

Similarly, we obtain for φ_2 the equations

$$\begin{aligned}\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'') &= 2\left(\left(\cos(2\varphi_2) + i\sin(2\varphi_2)\right)a'_{qp}\bar{a}'_{pq} + \left(\cos(2\varphi_2) - i\sin(2\varphi_2)\right)\bar{a}'_{qp}a'_{pq}\right)\\ &= 2i\left(a'_{qp}\bar{a}'_{pq} - \bar{a}'_{qp}a'_{pq}\right)\\ &= -\frac{4}{2i}\left(a'_{qp}\bar{a}'_{pq} - \bar{a}'_{qp}a'_{pq}\right) = -4\Im\left(a'_{qp}\bar{a}'_{pq}\right).\end{aligned}$$

This means that $\Delta_D^2(A'')$ has a minimum for $\varphi = -\operatorname{sgn}\left(\Im\left(a'_{qp}\bar{a}'_{pq}\right)\right)\frac{\pi}{4}$.

In the case $\Re\left(\bar{a}'_{qp}a'_{pq}\right) \neq 0$ we further obtain through equation 5.13 that

$$\tan(2\varphi) = \frac{\Im\left(\bar{a}'_{qp}a'_{pq}\right)}{\Re\left(\bar{a}'_{qp}a'_{pq}\right)}.$$

We observe that $\Delta_D^2(A'')$ is 2π periodic in φ so we need to investigate the full interval $]-\pi,\pi]$. We therefore obtain

$$\varphi_1 = \frac{1}{2}\arctan\left(\frac{\Im\left(\bar{a}'_{qp}a'_{pq}\right)}{\Re\left(\bar{a}'_{qp}a'_{pq}\right)}\right) \in \left]-\frac{\pi}{4},\frac{\pi}{4}\right[$$

and the further solutions $\varphi_2 = \varphi_1 - \frac{\pi}{2}$ and $\varphi_3 = \varphi_1 + \frac{\pi}{2}$. Once again we evaluate $\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'')$ on the individual φ_i in order to decide which one is in fact a minimum.

$$\begin{aligned}\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'') &= 2\left(\left(\cos(2\varphi_1) + i\sin(2\varphi_1)\right)a'_{qp}\bar{a}'_{pq} + \left(\cos(2\varphi_1) - i\sin(2\varphi_1)\right)\bar{a}'_{qp}a'_{pq}\right)\\ &= \frac{4\left|\bar{a}'_{qp}\right|^2\left|a'_{pq}\right|^2}{\sqrt{\Im\left(\bar{a}'_{qp}a'_{pq}\right)^2 + \Re\left(\bar{a}'_{qp}a'_{pq}\right)^2}} > 0.\end{aligned}$$

For $\varphi_{2,3}$ we get

$$\begin{aligned}\left(\frac{\partial}{\partial\varphi}\right)^2 \Delta_D^2(A'') &= 2\left(\cos(2\varphi_{2,3}) + i\sin(2\varphi_{2,3})\right)a'_{qp}\bar{a}'_{pq} + 2\left(\cos(2\varphi_{2,3}) - i\sin(2\varphi_{2,3})\right)\bar{a}'_{qp}a'_{pq}\\ &= 2\left(-\left(\cos(2\varphi_1) - i\sin(2\varphi_1)\right)a'_{qp}\bar{a}'_{pq}\right) + 2\left(-\cos(2\varphi_1) + i\sin(2\varphi_1)\right)\bar{a}'_{qp}a'_{pq}\\ &= \frac{-4\left|\bar{a}'_{qp}\right|^2\left|a'_{pq}\right|^2}{\sqrt{\Im\left(\bar{a}'_{qp}a'_{pq}\right)^2 + \Re\left(\bar{a}'_{qp}a'_{pq}\right)^2}} < 0.\end{aligned}$$

So here φ_1 is the only minimum.

Next we treat the case $d' \neq 0$.

First we note that $\sin^2\left(\frac{1}{2}\arccos(x)\right) = \frac{1-x}{2}$ and $\sin^2\left(\frac{1}{2}\arccos(x) + \frac{\pi}{2}\right) = \frac{1+x}{2}$. Again we

substitute for θ depending on $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)$. Please note that $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)\neq 0$ also implies that $|d'^2+\xi'^2|\neq 0$. We obtain, using relation 5.10 for θ, the equations

$$\begin{aligned}\sin^2(2\theta_1) &= \sin^2\left(\frac{1}{2}\arccos\left(\frac{\left(|d'|^2-|\xi'|^2\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{|d'^2+\xi'^2|}\right)\right)\\ &= \frac{1}{2}\left(1-\frac{|d'|^2-|\xi'|^2}{|d'^2+\xi'^2|}\right)\end{aligned}$$

if $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=1$ and

$$\begin{aligned}\sin^2(2\theta_2) &= \sin^2\left(\frac{1}{2}\arccos\left(\frac{\left(|d'|^2-|\xi'|^2\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{|d'^2+\xi'^2|}\right)+\frac{\pi}{2}\right)\\ &= \frac{1}{2}\left(1-\frac{|d'|^2-|\xi'|^2}{|d'^2+\xi'^2|}\right)\end{aligned}$$

if $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=-1$. Additionally,

$$\sin(4\theta_1)=\frac{2\Re\left(\bar{d}'\xi'\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{|d'^2+\xi'^2|}=\frac{2\Re\left(\bar{d}'\xi'\right)}{|d'^2+\xi'^2|}$$

if $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=1$,

$$\begin{aligned}\sin(4\theta_2) &= \sin\left(4\left(\theta_1+\frac{\pi}{4}\right)\right)=-\sin(4\theta_1)\\ &= -\frac{2\Re\left(\bar{d}'\xi'\right)\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)}{|d'^2+\xi'^2|}=\frac{2\Re\left(\bar{d}'\xi'\right)}{|d'^2+\xi'^2|}\end{aligned}$$

if $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=-1$, and

$$\begin{aligned}\sin(4\theta_1) &= 1\\ \sin(4\theta_2) &= -1\end{aligned}$$

if $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=0$. By these observations we can conclude that $\sin^2(2\theta)$, $\sin(4\theta)$ and therefore $\Delta_D^2(A'')$ do not depend on $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=\pm 1$. Suppose that $\operatorname{sgn}\left(|d'|^2-|\xi'|^2\right)=0$, then by using equation 5.8 we obtain

$$\Delta_D^2(A'')-\Delta_D^2(A') = -\frac{1}{2}\left|\Re\left(\bar{d}'\xi'\right)\right|=-\frac{1}{4}\left|\bar{d}'\xi'+d'\bar{\xi}'\right|. \tag{5.14}$$

Starting from equation 5.8 and assuming that $|d'|^2 - |\xi'|^2 \neq 0$ we compute

$$\begin{aligned}
&\Delta_D^2(A'') - \Delta_D^2(A') = \\
= \;& \frac{1}{2}\sin^2(2\theta)\left(|d'|^2 - |\xi'|^2\right) - \frac{1}{2}\sin(4\theta)\,\Re\left(\bar{d}'\xi'\right) \\
= \;& \frac{1}{4}\left(1 - \frac{\left(|d'|^2 - |\xi'|^2\right)}{|d'^2 + \xi'^2|}\right)\left(|d'|^2 - |\xi'|^2\right) - \frac{\left(\Re\left(\bar{d}'\xi'\right)\right)^2}{|d'^2 + \xi'^2|} \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \frac{\left(|d'|^2 - |\xi'|^2\right)^2}{|d'^2 + \xi'^2|}\right) - \frac{\left(\bar{d}'\xi' + d'\bar{\xi}'\right)^2}{4\,|d'^2 + \xi'^2|} \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \frac{\left(|d'|^2 - |\xi'|^2\right)^2 + \left(\bar{d}'\xi' + d'\bar{\xi}'\right)^2}{|d'^2 + \xi'^2|}\right) \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \frac{|d'|^4 + |\xi'|^4 - 2\,|d'|^2\,|\xi'|^2 + \bar{d}'^2\xi'^2 + d'^2\bar{\xi}'^2 + 2\,|d'|^2\,|\xi'|^2}{|d'^2 + \xi'^2|}\right) \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \frac{\left(d'^2 + \xi'^2\right)\left(\bar{d}'^2 + \bar{\xi}'^2\right)}{|d'^2 + \xi'^2|}\right) \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \frac{\left|d'^2 + \xi'^2\right|^2}{|d'^2 + \xi'^2|}\right) = \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - |d'^2 + \xi'^2|\right) \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \sqrt{|d'|^4 + |\xi'|^4 - 2\,|d'|^2\,|\xi'|^2 + 2\,|d'|^2\,|\xi'|^2 + \bar{d}'^2\xi'^2 + d'^2\bar{\xi}'^2}\right) \\
= \;& \frac{1}{4}\left(|d'|^2 - |\xi'|^2 - \sqrt{\left(|d'|^2 - |\xi'|^2\right)^2 + \left(\bar{d}'\xi' + d'\bar{\xi}'\right)^2}\right). \qquad (5.15)
\end{aligned}$$

Please note that if we let $|d'|^2 - |\xi'|^2 = 0$ in equation 5.15 we obtain equation 5.14. So the case of $|d'|^2 - |\xi'|^2 = 0$ is already covered via 5.15. Since $\Delta_D^2(A'') - \Delta_D^2(A')$ does not depend on $\operatorname{sgn}\left(|d'|^2 - |\xi'|^2\right)$ we may differentiate with respect to φ without having to distinguish different cases. Similarly like Goldstine and Horwitz in [61] we do not investigate formula 5.15 directly, but the product

$$\frac{4}{|d'|^2}\left(\Delta_D^2(A'') - \Delta_D^2(A')\right)$$

instead. Multiplying by $\frac{4}{|d'|^2}$ will not alter the location of the minima and maxima but it will make the formula more easy to handle. We obtain:

$$\begin{aligned}
\frac{4}{|d'|^2}\left(\Delta_D^2(A'') - \Delta_D^2(A')\right) &= \frac{4}{|d'|^2}\frac{1}{4}\left(|d'|^2 - |\xi'|^2 - |d'^2 + \xi'^2|\right) \\
&= 1 - \frac{|\xi'|^2}{|d'|^2} - \sqrt{\frac{1}{|d'|^4}\left(d'^2 + \xi'^2\right)\left(\bar{d}'^2 + \bar{\xi}'^2\right)} \\
&= 1 - \frac{|\xi'|^2}{|d'|^2} - \sqrt{\frac{1}{d'^2}\left(d'^2 + \xi'^2\right)\frac{1}{\bar{d}'^2}\left(\bar{d}'^2 + \bar{\xi}'^2\right)} \\
&= 1 - \frac{|\xi'|^2}{|d'|^2} - \sqrt{\left(1 + \frac{\xi'^2}{d'^2}\right)\left(1 + \frac{\bar{\xi}'^2}{\bar{d}'^2}\right)}. \qquad (5.16)
\end{aligned}$$

To simplify our notation let us introduce the abbreviations $x = -\frac{\xi'}{d'}$ and

$$\begin{aligned} y &= i\frac{\partial}{\partial\varphi}(x) = -\frac{i}{d'}\frac{\partial}{\partial\varphi}(\xi') = -\frac{i}{d'}\frac{\partial}{\partial\varphi}\left(e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq}\right) \\ &= -\frac{i}{d'}\left(ie^{i\varphi}a'_{qp} - ie^{-i\varphi}a'_{pq}\right) = \frac{1}{d'}\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right). \end{aligned}$$

Note that $\frac{\partial}{\partial\varphi}(x) = -iy$ and $\frac{\partial}{\partial\varphi}(\bar{x}) = i\bar{y}$.

After these preparations we compute the derivative of $\frac{4}{|d'|^2}\left(\Delta_D^2(A'') - \Delta_D^2(A')\right)$ with respect to φ:

$$\begin{aligned} &\frac{\partial}{\partial\varphi}\left(\frac{4}{|d'|^2}\left(\Delta_D^2(A'') - \Delta_D^2(A')\right)\right) = \frac{\partial}{\partial\varphi}\left(1 - x\bar{x} - \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{0.5}\right) \\ = \;& i(y\bar{x} - \bar{y}x) - \frac{1}{2}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(-i2xy\left(1+\bar{x}^2\right) + i2\bar{x}\bar{y}\left(1+x^2\right)\right) \\ = \;& i\left(y\bar{x} - \bar{y}x + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\right). \end{aligned} \tag{5.17}$$

A necessary requirement for $\frac{4}{|d'|^2}\left(\Delta_D^2(A'') - \Delta_D^2(A')\right)$ to have a minimum with respect to φ is again that its derivative needs to vanish on φ. So we determine the zero set of 5.17.

$$\begin{aligned} 0 &= i\left(y\bar{x} - \bar{y}x + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\right) \\ 0 &= y\bar{x} - \bar{y}x + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right) \\ \bar{y}x - y\bar{x} &= \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right) \end{aligned} \tag{5.18}$$

In order to simplify the task at hand we square both sides of the equation. However, later on we must take care of possible additional solutions and check whether they satisfy the original equation. We obtain

$$\frac{\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+\bar{x}^2\right)} = \left(\bar{y}x - y\bar{x}\right)^2.$$

We start to simplify this equation

$$\begin{aligned} \left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)^2 &= \left(1+x^2\right)\left(1+\bar{x}^2\right)\left(\bar{y}x - y\bar{x}\right)^2 \\ x^2y^2\left(1+\bar{x}^2\right)^2 + \bar{x}^2\bar{y}^2\left(1+x^2\right)^2 - & \\ 2xy\bar{x}\bar{y}\left(1+\bar{x}^2\right)\left(1+x^2\right) &= \ldots\cdot\left(\bar{y}^2x^2 - 2\bar{y}xy\bar{x} + y^2\bar{x}^2\right) \\ x^2y^2\left(1+\bar{x}^2\right)^2 + \bar{x}^2\bar{y}^2\left(1+x^2\right)^2 &= \ldots\cdot\left(\bar{y}^2x^2 + y^2\bar{x}^2\right) \\ x^2y^2\left(1+2\bar{x}^2+\bar{x}^4\right) + \bar{x}^2\bar{y}^2\left(1+2x^2+x^4\right) &= \left(1+\bar{x}^2+x^2+x^2\bar{x}^2\right)\left(\bar{y}^2x^2 + y^2\bar{x}^2\right) \\ x^2y^2 + 2x^2y^2\bar{x}^2 + x^2y^2\bar{x}^4 + & \\ \bar{x}^2\bar{y}^2 + 2x^2\bar{x}^2\bar{y}^2 + \bar{x}^2\bar{y}^2x^4 &= \bar{y}^2x^2 + \bar{y}^2x^2\bar{x}^2 + \bar{y}^2x^2x^2 + \bar{y}^2x^2x^2\bar{x}^2 + \\ & \quad y^2\bar{x}^2 + y^2\bar{x}^2\bar{x}^2 + y^2\bar{x}^2x^2 + y^2\bar{x}^2x^2\bar{x}^2 \\ x^2y^2 + x^2y^2\bar{x}^2 + \bar{x}^2\bar{y}^2 + x^2\bar{x}^2\bar{y}^2 &= \bar{y}^2x^2 + \bar{y}^2x^2x^2 + y^2\bar{x}^2 + y^2\bar{x}^2\bar{x}^2 \\ x^2y^2 + x^2y^2\bar{x}^2 + x^2\bar{x}^2\bar{y}^2 - \bar{y}^2x^2 - \bar{y}^2x^2x^2 &= y^2\bar{x}^2 - \bar{x}^2\bar{y}^2 + y^2\bar{x}^2\bar{x}^2 \\ x^2\left(y^2 + y^2\bar{x}^2 + \bar{x}^2\bar{y}^2 - \bar{y}^2 - \bar{y}^2x^2\right) &= \bar{x}^2\left(y^2 - \bar{y}^2 + y^2\bar{x}^2\right) \\ x^2\left(y^2 - \bar{y}^2 + y^2\bar{x}^2\right) + x^2\bar{y}^2\left(\bar{x}^2 - x^2\right) &= \bar{x}^2\left(y^2 - \bar{y}^2 + y^2\bar{x}^2\right) \end{aligned}$$

$$\begin{aligned}\left(x^2-\bar{x}^2\right)\left(y^2+y^2\bar{x}^2-\bar{y}^2\right)+x^2\bar{y}^2\left(\bar{x}^2-x^2\right) &= 0\\ \left(\bar{x}^2-x^2\right)\left(y^2+y^2\bar{x}^2-\bar{y}^2-x^2\bar{y}^2\right) &= 0\\ \left(\bar{x}-x\right)\left(\bar{x}+x\right)\left(y^2+y^2\bar{x}^2-\bar{y}^2-x^2\bar{y}^2\right) &= 0\end{aligned}$$

Further factorisation of the third factor yields

$$\begin{aligned}\left(y^2+y^2\bar{x}^2-\bar{y}^2-x^2\bar{y}^2\right) &= 0\\ y^2\left(1+\bar{x}^2\right) &= \bar{y}^2\left(1+x^2\right).\end{aligned}$$

Thus we can establish three distinct cases, which we have to investigate:

1. $\bar{x}=x$,
2. $\bar{x}=-x$, and
3. $y^2=k\left(1+x^2\right)$ with $k\in\mathbb{R}$.

First we derive the additional requirements under which the original (not squared) equation 5.18 holds as well.

Let us start with the first case $\bar{x}=x$, which means that x must be a real number.

$$\begin{aligned}0 &= \frac{xy\left(1+x^2\right)-x\bar{y}\left(1+x^2\right)}{\sqrt{\left(1+x^2\right)\left(1+x^2\right)}}+yx-\bar{y}x\\ &= xy-x\bar{y}+yx-\bar{y}x\\ &= 2x\left(y-\bar{y}\right)\end{aligned}$$

So either $x=0$ or $y=\bar{y}$ must hold additionally.

Now let us consider the case $x=-\bar{x}$, in which x is purely imaginary.

$$\begin{aligned}0 &= \frac{xy\left(1+\bar{x}^2\right)-\bar{x}\bar{y}\left(1+x^2\right)}{\sqrt{\left(1+x^2\right)\left(1+\bar{x}^2\right)}}+y\bar{x}-\bar{y}x\\ &= \frac{xy\left(1+x^2\right)+x\bar{y}\left(1+x^2\right)}{\left|\left(1+x^2\right)\right|}-yx-\bar{y}x\end{aligned}$$

As x is purely imaginary it can be written as $x=i\left|x\right|\operatorname{sgn}\left(\Im\left(x\right)\right)$.

$$\begin{aligned}1+x^2>0\Leftrightarrow\left(i\left|x\right|\operatorname{sgn}\left(\Im\left(x\right)\right)\right)^2 &> -1\\ -\left|x\right| &> -1\\ \left|x\right| &< 1\end{aligned}$$

and equivalently $1+x^2<0$ if $|x|>1$. In the case of $|x|<1$ the above equation holds without any further requirements. On the other hand if $|x|>1$ holds we obtain

$$-2x\left(y+\bar{y}\right) = 0.$$

Thus the additional requirement is that $y=-\bar{y}$ needs to hold as well.

It remains to investigate the third case in which we have $y^2 = k\left(1+x^2\right)$ and $\bar{y}^2 = k\left(1+\bar{x}^2\right)$ with $k \in \mathbb{R}$. If $k = 0$ then $y = 0$ and the original equation holds. Let us now assume that $k \neq 0$. By substituting $1 + x^2 = \frac{y^2}{k}$ and $1 + \bar{x}^2 = \frac{\bar{y}^2}{k}$ in 5.18, we derive

$$\begin{aligned} 0 &= \left(xy\tfrac{\bar{y}^2}{k} - \bar{x}\bar{y}\tfrac{y^2}{k}\right)/\sqrt{\tfrac{y^2}{k}\tfrac{\bar{y}^2}{k}} + y\bar{x} - \bar{y}x \\ &= \left(x|y^2|\tfrac{\bar{y}}{k} - \bar{x}|y^2|\tfrac{y}{k}\right)/\tfrac{|y^2|}{|k|} + y\bar{x} - \bar{y}x \\ &= y\bar{x} - \bar{y}x - (y\bar{x} - \bar{y}x)\frac{|k|}{k} \\ &= (y\bar{x} - \bar{y}x)\left(1 - \frac{|k|}{k}\right). \end{aligned}$$

Now if $k > 0$ the equation will always hold. If $k < 0$ we must additionally require that $y\bar{x} = \bar{y}x$. So we have finally derived the zero set of equation 5.18 which is summarised in the following table:

1	$x = 0$
2	$\bar{x} = x$ and $y = \bar{y}$
3	$x = -\bar{x}$ with $\lvert x\rvert < 1$
4	$x = -\bar{x}$ and $y = -\bar{y}$ with $\lvert x\rvert > 1$
5	$y^2 = k\left(1+x^2\right)$ with $k > 0$
6	$y^2 = k\left(1+x^2\right)$ with $k = 0$
7	$y^2 = k\left(1+x^2\right)$ and $y\bar{x} = \bar{y}x$ with $k < 0$

It remains to be seen which of these solutions are maxima/minima of the original equation. For this purpose we will look at the second derivative of $\frac{4}{|d'|^2}\left(\Delta_D^2\left(A''\right) - \Delta_D^2\left(A'\right)\right)$ given in 5.16.

$$\begin{aligned} & \frac{\partial^2}{\partial^2\varphi}\left(\frac{4}{|d'|^2}\left(\Delta_D^2\left(A''\right) - \Delta_D^2\left(A'\right)\right)\right) \\ = \; & \frac{\partial}{\partial\varphi}\left(\frac{\partial}{\partial\varphi}\left(\frac{4}{|d'|^2}\left(\Delta_D^2\left(A''\right) - \Delta_D^2\left(A'\right)\right)\right)\right) \\ \overset{5.17}{=} \; & \frac{\partial}{\partial\varphi} i\left((y\bar{x} - \bar{y}x) + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\right) \\ = \; & 2\left(|x|^2 - |y|^2\right) + \frac{\partial}{\partial\varphi} i\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right) \\ = \; & 2\left(|x|^2 - |y|^2\right) + i\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\ & \left(-iyy\left(1+\bar{x}^2\right) + x\left(-ix\left(1+\bar{x}^2\right) + 2\bar{x}iy\bar{y}\right) - \left(i\bar{y}\bar{y}\left(1+x^2\right) + \bar{x}\left(i\bar{x}\left(1+x^2\right) - i2x\bar{y}y\right)\right)\right) + \\ & i\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\frac{\partial}{\partial\varphi}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\ = \; & 2\left(|x|^2 - |y|^2\right) + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\ & \left(y^2\left(1+\bar{x}^2\right) + x\left(x\left(1+\bar{x}^2\right) - 2\bar{x}y\bar{y}\right) - \left(-\bar{y}^2\left(1+x^2\right) + \bar{x}\left(-\bar{x}\left(1+x^2\right) + 2x\bar{y}y\right)\right)\right) + \\ & i\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\frac{\partial}{\partial\varphi}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \end{aligned}$$

$$
\begin{aligned}
= \quad & ... + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
& \left(y^2\left(1+\bar{x}^2\right) + x^2\left(1+\bar{x}^2\right) - 2x\bar{x}y\bar{y} + \bar{y}^2\left(1+x^2\right) + \bar{x}^2\left(1+x^2\right) - 2x\bar{x}\bar{y}y\right) + \\
& i\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\frac{\partial}{\partial\varphi}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
= \quad & ... + \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(\left(y^2+x^2\right)\left(1+\bar{x}^2\right) - 4\left|x\right|^2\left|y\right|^2 + \left(\bar{x}^2+\bar{y}^2\right)\left(1+x^2\right)\right) + \\
& i\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)\frac{\partial}{\partial\varphi}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
= \quad & ... + i\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right) \\
& \left(-\frac{1}{2}\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-1.5}\left(-2ixy\left(1+\bar{x}^2\right) + \left(1+x^2\right)2i\bar{x}\bar{y}\right)\right) \\
= \quad & ... - \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-1.5}\left(xy\left(1+\bar{x}^2\right) - \left(1+x^2\right)\bar{x}\bar{y}\right)^2 \\
= \quad & 2\left(\left|x\right|^2 - \left|y\right|^2\right) - \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
& \left(\frac{\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+\bar{x}^2\right)} + 4\left|x\right|^2\left|y\right|^2 - \left(y^2+x^2\right)\left(1+\bar{x}^2\right) - \left(\bar{x}^2+\bar{y}^2\right)\left(1+x^2\right)\right)
\end{aligned}
$$

In the first case with $x = 0$ the second derivative evaluates to

$$
\begin{aligned}
& 2\left(\left|x\right|^2 - \left|y\right|^2\right) - \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
& \left(\frac{\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+\bar{x}^2\right)} + 4\left|x\right|^2\left|y\right|^2 - \left(y^2+x^2\right)\left(1+\bar{x}^2\right) - \left(\bar{x}^2+\bar{y}^2\right)\left(1+x^2\right)\right) \\
= \quad & -2\left|y\right|^2 + y^2 + \bar{y}^2 \\
= \quad & \left(y-\bar{y}\right)^2 = \left(2i\Im\left(y\right)\right)^2 = -4\Im\left(y\right)^2.
\end{aligned}
$$

In the second case where both x and y are real numbers we obtain

$$
\begin{aligned}
& 2\left(x^2-y^2\right) - \frac{1}{\left|\left(1+x^2\right)\right|} \\
& \left(\frac{\left(xy\left(1+x^2\right) - xy\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+x^2\right)} + 4x^2y^2 - \left(y^2+x^2\right)\left(1+x^2\right) - \left(x^2+y^2\right)\left(1+x^2\right)\right) \\
= \quad & 2\left(x^2-y^2\right) - \frac{1}{1+x^2}\left(4x^2y^2 - 2\left(x^2+y^2\right)\left(1+x^2\right)\right) \\
= \quad & 2\left(x^2-y^2\right) - \frac{4x^2y^2}{1+x^2} + 2\left(x^2+y^2\right) = 4x^2 - \frac{4x^2y^2}{1+x^2} = 4x^2\left(1 - \frac{y^2}{1+x^2}\right).
\end{aligned}
$$

In the third case we let $x = -\bar{x}$ and $\left|x\right| < 1$. Note that $x^2 = -\left|x\right|^2$ as x is purely imaginary. Here we derive:

$$
\begin{aligned}
& 2\left(\left|x\right|^2 - \left|y\right|^2\right) - \left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5} \\
& \left(\frac{\left(xy\left(1+\bar{x}^2\right) - \bar{x}\bar{y}\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+\bar{x}^2\right)} + 4\left|x\right|^2\left|y\right|^2 - \left(y^2+x^2\right)\left(1+\bar{x}^2\right) - \left(\bar{x}^2+\bar{y}^2\right)\left(1+x^2\right)\right)
\end{aligned}
$$

$$
\begin{aligned}
&= 2\left(|x|^2-|y|^2\right)-\frac{1}{|1+x^2|}\\
&\quad\left(\frac{\left(xy\left(1+x^2\right)+x\bar{y}\left(1+x^2\right)\right)^2}{\left(1+x^2\right)^2}+4|x|^2|y|^2-\left(y^2+x^2\right)\left(1+x^2\right)-\left(x^2+\bar{y}^2\right)\left(1+x^2\right)\right)\\
&= 2\left(|x|^2-|y|^2\right)-\frac{1}{1+x^2}\\
&\quad\left(x^2(y+\bar{y})^2+4|x|^2|y|^2-\left(y^2+x^2\right)\left(1+x^2\right)-\left(x^2+\bar{y}^2\right)\left(1+x^2\right)\right)\\
&= 2\left(|x|^2-|y|^2\right)-\frac{1}{1+x^2}\left(x^2(y+\bar{y})^2+4|x|^2|y|^2-\left(1+x^2\right)\left(y^2+2x^2+\bar{y}^2\right)\right)\\
&= 2\left(|x|^2-|y|^2\right)-\frac{1}{1+x^2}\\
&\quad\left(x^2y^2+2x^2y\bar{y}+x^2\bar{y}^2+4|x|^2|y|^2-y^2-2x^2-\bar{y}^2-x^2y^2-2x^4-x^2\bar{y}^2\right)\\
&= 2\left(|x|^2-|y|^2\right)-\frac{1}{1+x^2}\left(2x^2y\bar{y}+4|x|^2|y|^2-y^2-2x^2-\bar{y}^2-2x^4\right)\\
&= \frac{1}{1+x^2}\left(2\left(|x|^2-|y|^2\right)\left(1+x^2\right)-\left(2x^2y\bar{y}+4|x|^2|y|^2-y^2-2x^2-\bar{y}^2-2x^4\right)\right)\\
&= \frac{1}{1+x^2}\left(-2x^2-2|y|^2-2x^4-2|y|^2x^2-2|y|^2x^2+4|y|^2x^2+y^2+2x^2+\bar{y}^2+2x^4\right)\\
&= \frac{1}{1+x^2}\left(-2|y|^2+y^2+\bar{y}^2\right)=\frac{(y-\bar{y})^2}{1+x^2}=\frac{(y-\bar{y})^2}{1-|x|^2}.
\end{aligned}
$$

We move along to the forth case $x=-\bar{x}$ and $y=-\bar{y}$ with $|x|>1$ and obtain the following value for the second derivative:

$$
\begin{aligned}
&\quad 2\left(y^2-x^2\right)+\frac{1}{1+x^2}\\
&\quad\left(\frac{\left(xy\left(1+x^2\right)-xy\left(1+x^2\right)\right)^2}{\left(1+x^2\right)\left(1+x^2\right)}+4x^2y^2-\left(y^2+x^2\right)\left(1+x^2\right)-\left(x^2+y^2\right)\left(1+x^2\right)\right)\\
&= 2\left(y^2-x^2\right)+\frac{1}{1+x^2}\left(4x^2y^2-2\left(y^2+x^2\right)\left(1+x^2\right)\right)\\
&= \frac{1}{1+x^2}\left(2y^2\left(1+x^2\right)-2x^2\left(1+x^2\right)+4x^2y^2-2\left(y^2+y^2x^2+x^2+x^4\right)\right)\\
&= \frac{1}{1+x^2}\left(2y^2+2x^2y^2-2x^2-2x^4+4x^2y^2-2y^2-2y^2x^2-2x^2-2x^4\right)\\
&= \frac{1}{1+x^2}\left(-4x^2-4x^4+4x^2y^2\right)=\frac{4|x|^2}{1+x^2}\left(1+x^2-y^2\right)\\
&= 4|x|^2\left(1-\frac{y^2}{1+x^2}\right).
\end{aligned}
$$

In the fifth case $y^2=k\left(1+x^2\right)$ with $k>0$ using $\left(1+\bar{x}^2\right)=\frac{\bar{y}^2}{k}$ we obtain:

$$
\begin{aligned}
&\quad 2\left(|x|^2-|y|^2\right)-\left(\frac{|y|^4}{k^2}\right)^{-0.5}\\
&\quad\left(\left(xy\frac{\bar{y}^2}{k}-\bar{x}\bar{y}\frac{y^2}{k}\right)^2/\frac{|y|^4}{k^2}+4|x|^2|y|^2-\left(y^2+x^2\right)\frac{\bar{y}^2}{k}-\left(\bar{x}^2+\bar{y}^2\right)\frac{y^2}{k}\right)\\
&= 2\left(|x|^2-|y|^2\right)-\frac{k}{|y|^2}\left(\frac{\left(xy\bar{y}^2-\bar{x}\bar{y}y^2\right)^2}{|y|^4}+4|x|^2|y|^2-\left(y^2+x^2\right)\frac{\bar{y}^2}{k}-\left(\bar{x}^2+\bar{y}^2\right)\frac{y^2}{k}\right)
\end{aligned}
$$

$$
\begin{aligned}
&= 2\left(|x|^2-|y|^2\right)-\frac{k}{|y|^2}\left((x\bar{y}-\bar{x}y)^2+4|x|^2|y|^2-\left(y^2+x^2\right)\frac{\bar{y}^2}{k}-\left(\bar{x}^2+\bar{y}^2\right)\frac{y^2}{k}\right)\\
&= \frac{1}{|y|^2}\left(2|x|^2|y|^2-2|y|^4-k(x\bar{y}-\bar{x}y)^2-4k|x|^2|y|^2+\left(y^2+x^2\right)\bar{y}^2+\left(\bar{x}^2+\bar{y}^2\right)y^2\right)\\
&= \frac{1}{|y|^2}\left(2|x|^2|y|^2-2|y|^4-k(x\bar{y}-\bar{x}y)^2-4k|x|^2|y|^2+|y|^4+x^2\bar{y}^2+\bar{x}^2y^2+|y|^4\right)\\
&= \frac{1}{|y|^2}\left(2|x|^2|y|^2-k(x\bar{y}-\bar{x}y)^2-4k|x|^2|y|^2+x^2\bar{y}^2+\bar{x}^2y^2\right)\\
&= \frac{1}{|y|^2}\left((x\bar{y}+\bar{x}y)^2-k(x\bar{y}-\bar{x}y)^2-4k|x|^2|y|^2\right)\\
&= \frac{1}{|y|^2}\left((x\bar{y}+\bar{x}y)^2-kx^2\bar{y}^2-k\bar{x}^2y^2+2k|x|^2|y|^2-4k|x|^2|y|^2\right)\\
&= \frac{1}{|y|^2}\left((x\bar{y}+\bar{x}y)^2-k\left(x^2\bar{y}^2+\bar{x}^2y^2+2|x|^2|y|^2\right)\right)\\
&= \frac{1}{|y|^2}\left((x\bar{y}+\bar{x}y)^2-k(x\bar{y}+\bar{x}y)^2\right)=\frac{(x\bar{y}+\bar{x}y)^2}{|y|^2}(1-k).
\end{aligned}
$$

Next we investigate $k=0$ which implies $y=0$. In this case we obtain:

$$
\begin{aligned}
&2|x|^2-\left(\left(1+x^2\right)\left(1+\bar{x}^2\right)\right)^{-0.5}\left(-x^2\left(1+\bar{x}^2\right)-\bar{x}^2\left(1+x^2\right)\right)\\
&= 2|x|^2+\frac{x^2\left(1+\bar{x}^2\right)+\bar{x}^2\left(1+x^2\right)}{|1+x^2|}=\frac{2|x|^2\left|1+x^2\right|+x^2\left(1+\bar{x}^2\right)+\bar{x}^2\left(1+x^2\right)}{|1+x^2|}\\
&= \frac{2\Re\left(|x|^2\left(\left|1+x^2\right|\right)+x^2\left(1+\bar{x}^2\right)\right)}{|1+x^2|}\\
&= \frac{2\Re\left(x\bar{x}\left(1+x^2\right)^{0.5}\left(1+\bar{x}^2\right)^{0.5}+xx\left(1+\bar{x}^2\right)^{0.5}\left(1+\bar{x}^2\right)^{0.5}\right)}{|1+x^2|}\\
&= \frac{2\Re\left(x\left(1+\bar{x}^2\right)^{0.5}\left(\bar{x}\left(1+x^2\right)^{0.5}+x\left(1+\bar{x}^2\right)^{0.5}\right)\right)}{|1+x^2|}\\
&= \frac{2\Re\left(x\left(1+\bar{x}^2\right)^{0.5}2\Re\left(x\left(1+\bar{x}^2\right)^{0.5}\right)\right)}{|1+x^2|}=\frac{4\left(\Re\left(x\left(1+\bar{x}^2\right)^{0.5}\right)^2\right)}{|1+x^2|}.
\end{aligned}
$$

The last case we have to investigate is $y^2=k\left(1+x^2\right)$ and $y\bar{x}-\bar{y}x=0$ with $k<0$. Using these two equations, we observe that

$$
\begin{aligned}
\frac{x}{\bar{x}} &= \frac{y}{\bar{y}}\\
\frac{x^2}{\bar{x}^2} &= \frac{y^2}{\bar{y}^2}\\
\frac{x^2}{\bar{x}^2} &= \frac{k\left(1+x^2\right)}{k\left(1+\bar{x}^2\right)}\\
\frac{x^2}{\bar{x}^2} &= \frac{1+x^2}{1+\bar{x}^2}\\
\bar{x}^2+|x|^4 &= x^2+|x|^4\\
\bar{x}^2 &= x^2.
\end{aligned}
$$

And consequently

$$y^2 = k\left(1+x^2\right) = k\left(1+\bar{x}^2\right) = \bar{y}^2.$$

From this follows that either $x = \bar{x}$ and $y = \bar{y}$ or $x = -\bar{x}$ and $y = -\bar{y}$. Because $k < 0$ we can exclude the first case. So x and y have to be purely imaginary numbers. As $y^2/k > 0$ holds, also $1 + x^2 > 0$ has to be satisfied. We know that $x^2 > -1$ if and only if $|x| < 1$. Thus we observe that the seventh case is identical to the third case, which we have already analysed before.

Now let us collect again the results that we have obtained thus far:

1	$x = 0$	$-4\Im\left(y\right)^2$
2	$\bar{x} = x$ and $y = \bar{y}$	$4x^2\left(1 - \frac{y^2}{1+x^2}\right)$
3	$x = -\bar{x}$ with $\|x\| < 1$	$\frac{\left(y-\bar{y}\right)^2}{1-\|x\|^2}$
4	$x = -\bar{x}$ and $y = -\bar{y}$ with $\|x\| > 1$	$4\|x\|^2\left(1 - \frac{y^2}{1+x^2}\right)$
5	$y^2 = k\left(1+x^2\right)$ with $k > 0$	$\frac{\left(x\bar{y}+\bar{x}y\right)^2}{\|y\|^2}\left(1-k\right)$
6	$y^2 = k\left(1+x^2\right)$ with $k = 0$	$\frac{4\left(\Re\left(x\sqrt{1+\bar{x}^2}\right)\right)^2}{\|1+x^2\|}$
7	$y^2 = k\left(1+x^2\right)$ and $y\bar{x} = \bar{y}x$ with $k < 0$	$\frac{\left(y-\bar{y}\right)^2}{1-\|x\|^2}$

In case 1 the second derivative is negative. We have thus found a maximum and need not investigate this case any further.

The second derivative in cases 3 and 7 is of the form $\frac{\left(y-\bar{y}\right)^2}{1-|x|^2}$. Note that $|x| < 1$, so $\frac{1}{1-|x|^2} > 0$. Additionally $y = -\bar{y}$, so y is purely imaginary unless $y = 0$. If $y = 0$ we are in the first case again and the same arguments apply. So if $y \neq 0$ we obtain $\frac{\left(y-\bar{y}\right)^2}{1-|x|^2} < 0$ and no minimum will be assumed here.

In cases 2, 4, 5, and 6 the second derivative is of the form $l^2\left(1-k\right)$ with $l \in \mathbb{R}$ and $k \geq 0$. Now if we assure that $k < 1$, then $l^2\left(1-k\right)$ will be positive and we have found a minimum. Naturally case 6 can be embedded in case 5 if we allow $k \geq 0$ and take the derivative from 6 if $k = y = 0$.

Additionally, we observe that $\bar{y}^2 = k\left(1+\bar{x}^2\right)$ holds in case 5. Together with the equation $y^2 = k\left(1+x^2\right)$ we can thus eliminate k and write $\frac{y^2}{\bar{y}^2} = \frac{\left(1+x^2\right)}{\left(1+\bar{x}^2\right)}$. Case 2 and 4 satisfy this equation, so we only need to treat the most general case 5.

In order to determine the actual value of φ we need to recall the definitions of x and y and then solve for φ.

$$\begin{aligned} x &= -\frac{\xi'}{d'} = -\frac{e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq}}{d'} \\ y &= i\frac{\partial}{\partial\varphi}x = \frac{e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}}{d'} \end{aligned}$$

By substituting the above expression for y in $y^2 = k\left(1+x^2\right)$ we obtain

$$\begin{aligned} y^2 = \frac{\left(e^{i\varphi}a_{qp}^{'} - e^{-i\varphi}a_{pq}^{'}\right)^2}{d^{'2}} &= k\left(1+\frac{\xi^{'2}}{d^{'2}}\right) = k\left(1+x^2\right) \\ \left(e^{i\varphi}a_{qp}^{'} - e^{-i\varphi}a_{pq}^{'}\right)^2 &= k\left(d^{'2}+\xi^{'2}\right) \\ e^{2i\varphi}a_{qp}^{'2} - 2a_{qp}^{'}a_{pq}^{'} + e^{-2i\varphi}a_{pq}^{'2} &= kd^{'2} + k\left(e^{2i\varphi}a_{qp}^{'2} + 2a_{qp}^{'}a_{pq}^{'} + e^{-2i\varphi}a_{pq}^{'2}\right). \end{aligned} \tag{5.19}$$

Now we start to simplify this equation further

$$\begin{aligned} (1-k)\,e^{2i\varphi}a_{qp}^{'2} - 2\,(1+k)\,a_{qp}^{'}a_{pq}^{'} + (1-k)\,e^{-2i\varphi}a_{pq}^{'2} &= kd^{'2} \\ (1-k)\,e^{2i\varphi}a_{qp}^{'2} - 2\,(1-k)\,a_{qp}^{'}a_{pq}^{'} - 4ka_{qp}^{'}a_{pq}^{'} + (1-k)\,e^{-2i\varphi}a_{pq}^{'2} &= kd^{'2} \\ (1-k)\left(e^{2i\varphi}a_{qp}^{'2} - 2a_{qp}^{'}a_{pq}^{'} + e^{-2i\varphi}a_{pq}^{'2}\right) &= kd^{'2} + 4ka_{qp}^{'}a_{pq}^{'} \\ \left(e^{i\varphi}a_{qp}^{'} - e^{-i\varphi}a_{pq}^{'}\right)^2 &= \frac{k}{1-k}\cdot \\ & \quad \left(d^{'2} + 4a_{qp}^{'}a_{pq}^{'}\right). \end{aligned} \tag{5.20}$$

In order to be able to eliminate k, we look at the imaginary and real part of equation 5.19 independently

$$k = \frac{\Im\left(\left(e^{i\varphi}a_{qp}^{'} - e^{-i\varphi}a_{pq}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2}\right)}.$$

We thus compute

$$\begin{aligned} \frac{k}{1-k} &= \frac{\Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2}\right)} \Bigg/ \left(1 - \frac{\Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2}\right)}\right) \\ &= \frac{\Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2}\right)} \Bigg/ \left(\frac{\Im\left(d^{'2}+\xi^{'2}\right) - \Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2}\right)}\right) \\ &= \frac{\Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2} - \left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}. \end{aligned}$$

So if we substitute this expression in equation 5.20 we obtain

$$\left(e^{i\varphi}a_{qp}^{'} - e^{-i\varphi}a_{pq}^{'}\right)^2 = \frac{\Im\left(\left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}{\Im\left(d^{'2}+\xi^{'2} - \left(e^{-i\varphi}a_{pq}^{'} - e^{i\varphi}a_{qp}^{'}\right)^2\right)}\left(d^{'2} + 4a_{qp}^{'}a_{pq}^{'}\right).$$

First we simplify $\xi'^2 - \left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2$:

$$\begin{aligned}\xi'^2 - \left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2 &= \left(e^{-i\varphi}a'_{pq} + e^{i\varphi}a'_{qp}\right)^2 - \left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2 \\ &= \left(e^{-i\varphi}a'_{pq} + e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq} + e^{i\varphi}a'_{qp}\right)\cdot \\ &\quad \left(e^{-i\varphi}a'_{pq} + e^{i\varphi}a'_{qp} + e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right) \\ &= 2\left(e^{i\varphi}a'_{qp}\right)2\left(e^{-i\varphi}a'_{pq}\right) = 4a'_{pq}a'_{qp}.\end{aligned}$$

Next we simplify the original equation:

$$\begin{aligned}\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2 &= \frac{\Im\left(\left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2\right)}{\Im\left(d'^2 + 4a'_{qp}a'_{pq}\right)}\left(d'^2 + 4a'_{qp}a'_{pq}\right) \\ \Re\left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2\right) &= \frac{\Im\left(\left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2\right)}{\Im\left(d'^2 + 4a'_{pq}a'_{qp}\right)}\Re\left(d'^2 + 4a'_{qp}a'_{pq}\right)\end{aligned}$$

$$\begin{aligned}0 &= \Re\left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2\right)\Im\left(d'^2 + 4a'_{qp}a'_{pq}\right) - \\ &\quad \Im\left(\left(e^{-i\varphi}a'_{pq} - e^{i\varphi}a'_{qp}\right)^2\right)\Re\left(d'^2 + 4a'_{qp}a'_{pq}\right) \\ &= \frac{1}{4i}\left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2 + \left(e^{-i\varphi}\bar{a}'_{qp} - e^{i\varphi}\bar{a}'_{pq}\right)^2\right)\left(d'^2 + 4a'_{qp}a'_{pq} - \vec{d}'^2 - 4\bar{a}'_{pq}\bar{a}'_{qp}\right) \\ &\quad -\frac{1}{4i}\left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2 - \left(e^{-i\varphi}\bar{a}'_{qp} - e^{i\varphi}\bar{a}'_{pq}\right)^2\right)\left(d'^2 + 4a'_{qp}a'_{pq} + \vec{d}'^2 + 4\bar{a}'_{pq}\bar{a}'_{qp}\right) \\ 0 &= \left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2 + \left(e^{-i\varphi}\bar{a}'_{qp} - e^{i\varphi}\bar{a}'_{pq}\right)^2\right)\left(d'^2 + 4a'_{qp}a'_{pq} - \vec{d}'^2 - 4\bar{a}'_{pq}\bar{a}'_{qp}\right) \\ &\quad -\left(\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2 - \left(e^{-i\varphi}\bar{a}'_{qp} - e^{i\varphi}\bar{a}'_{pq}\right)^2\right)\left(d'^2 + 4a'_{qp}a'_{pq} + \vec{d}'^2 + 4\bar{a}'_{pq}\bar{a}'_{qp}\right) \\ &= 2\left(e^{-i\varphi}\bar{a}'_{qp} - e^{i\varphi}\bar{a}'_{pq}\right)^2\left(d'^2 + 4a'_{qp}a'_{pq}\right) - 2\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2\left(\vec{d}'^2 + 4\bar{a}'_{pq}\bar{a}'_{qp}\right).\end{aligned}$$

So we arrive at

$$\begin{aligned}\left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)^2\overline{\left(d'^2 + 4a'_{qp}a'_{pq}\right)} &= \left(e^{i\varphi}\bar{a}'_{pq} - e^{-i\varphi}\bar{a}'_{qp}\right)^2\left(d'^2 + 4a'_{qp}a'_{pq}\right) \\ \left(e^{i\varphi}a'_{qp} - e^{-i\varphi}a'_{pq}\right)\sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} &= \pm\left(e^{i\varphi}\bar{a}'_{pq} - e^{-i\varphi}\bar{a}'_{qp}\right)\sqrt{d'^2 + 4a'_{qp}a'_{pq}}\end{aligned}$$

$$\begin{aligned}&\left(\cos(\varphi)\,a'_{qp} + i\sin(\varphi)\,a'_{qp} - \left(\cos(\varphi)\,a'_{pq} - i\sin(\varphi)\,a'_{pq}\right)\right)\sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \\ = \;&\pm\left(\cos(\varphi)\,\bar{a}'_{pq} + i\sin(\varphi)\,\bar{a}'_{pq} - \left(\cos(\varphi)\,\bar{a}'_{qp} - i\sin(\varphi)\,\bar{a}'_{qp}\right)\right)\sqrt{d'^2 + 4a'_{qp}a'_{pq}}\end{aligned}$$

$$\begin{aligned}&\left(\cos(\varphi)\left(a'_{pq} - a'_{qp}\right) - \sin(\varphi)\,i\left(a'_{pq} + a'_{qp}\right)\right)\sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \\ = \;&\pm\left(\cos(\varphi)\left(\bar{a}'_{pq} - \bar{a}'_{qp}\right) + \sin(\varphi)\,i\left(\bar{a}'_{pq} + \bar{a}'_{qp}\right)\right)\sqrt{d'^2 + 4a'_{qp}a'_{pq}}\end{aligned}$$

$$\begin{aligned} & \sin(\varphi)\, i \left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \pm \left(\bar{a}'_{pq} + \bar{a}'_{qp} \right) \sqrt{d'^2 + 4a'_{qp}a'_{pq}} \right) \\ = \; & \cos(\varphi) \left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \mp \left(\bar{a}'_{pq} - \bar{a}'_{qp} \right) \sqrt{d'^2 + 4a'_{qp}a'_{pq}} \right) \end{aligned}$$

Finally, we can solve for φ

$$\frac{\sin(\varphi)}{\cos(\varphi)} = \frac{2\left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \mp \left(\bar{a}'_{pq} - \bar{a}'_{qp} \right) \sqrt{d'^2 + 4a'_{qp}a'_{pq}} \right)}{2i\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \pm \left(\bar{a}'_{pq} + \bar{a}'_{qp} \right) \sqrt{d'^2 + 4a'_{qp}a'_{pq}} \right)}.$$

Thus

$$\tan(\varphi_1) = \frac{\Im\left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)}{\Re\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)}$$

if $\Re\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) \neq 0$ and $\cos(\varphi_1) = 0$ if $\Re\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) = 0$. And as a second solution we obtain

$$\tan(\varphi_2) = -\frac{\Re\left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)}{\Im\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)}$$

if $\Im\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) \neq 0$ and $\cos(\varphi_2) = 0$ if $\Im\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) = 0$. What remains to be done is to figure out which of the solutions is in fact a minimum. Therefore, we recall that $0 \leq k < 1$ needs to hold. If we substitute φ_1 in the equation

$$\frac{k}{1-k} = \frac{\left(e^{-i\varphi} a'_{pq} - e^{i\varphi} a'_{qp} \right)^2}{d'^2 + 4a'_{qp}a'_{pq}}$$

(compare 5.20), we obtain by a straightforward computation

$$\frac{k}{1-k} = \frac{\left(\left| a'_{qp} \right|^2 - \left| a'_{pq} \right|^2 \right)^2}{\left(\Re\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) \right)^2 + \left(\Im\left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right) \right)^2} := c.$$

Since $c \geq 0$ and $k = \frac{c}{1+c}$ we know that $0 \leq k < 1$. We have thus found a minimum in the case of a single matrix if we let

$$\varphi = \arctan\left(\frac{\Im\left(\left(a'_{pq} - a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)}{\Re\left(\left(a'_{pq} + a'_{qp} \right) \sqrt{\overline{d'^2 + 4a'_{qp}a'_{pq}}} \right)} \right)$$

in case $\Re\left(\left(a'_{pq}+a'_{qp}\right)\sqrt{\overline{d'^2+4a'_{qp}a'_{pq}}}\right)\neq 0$ and $\varphi=\operatorname{sgn}\left(\Im\left(\left(a'_{pq}-a'_{qp}\right)\sqrt{\overline{d'^2+4a'_{qp}a'_{pq}}}\right)\right)\frac{\pi}{2}$ in case $\Re\left(\left(a'_{pq}+a'_{qp}\right)\sqrt{\overline{d'^2+4a'_{qp}a'_{pq}}}\right)=0$.

It is, fortunately, straightforward to carry over our proof to the case of n matrices as we will see from the following arguments. It is clear that

$$\frac{\partial}{\partial\varphi}\sum_{h=1}^{n}\Delta_D^2\left(A''_h\right) \;=\; \sum_{h=1}^{n}\frac{\partial}{\partial\varphi}\Delta_D^2\left(A''_h\right).$$

So we can directly translate our results and obtain for $\sum_{h=1}^{n}\left|d'_h\right|^2=0$ that the equation

$$i\sin\left(2\varphi\right)\sum_{h=1}^{n}\left(\bar{a}'_{h,qp}a'_{h,pq}+a'_{h,qp}\bar{a}'_{h,pq}\right) \;=\; \cos\left(2\varphi\right)\sum_{h=1}^{n}\left(\bar{a}'_{h,qp}a'_{h,pq}-a'_{h,qp}\bar{a}'_{h,pq}\right)$$

needs to hold for φ to be a stationary point of $\sum_{h=1}^{n}\Delta_D^2\left(A''_h\right)$.

Similarly like in the case of a single matrix we obtain for $\sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)=0$ the solutions $\varphi=-\operatorname{sgn}\left(\sum_{h=1}^{n}\Im\left(a'_{h,qp}\bar{a}'_{h,pq}\right)\right)\frac{\pi}{4}$. If $\sum_{h=1}^{n}\left|d'_h\right|^2=0$ and $\sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\neq 0$ we get $\varphi=\frac{1}{2}\arctan\left(\frac{\sum_{h=1}^{n}\Im\left(\bar{a}'_{h,qp}a'_{h,pq}\right)}{\sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)}\right)$. In the setting of n matrices we recall that

$$\sin^2\left(2\theta\right)=\sin^2\left(\frac{1}{2}\arccos\left(x\right)\right)=\frac{1-x}{2}$$

with $x=\frac{\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\operatorname{sgn}\left(\sum_h\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\right)^2}}$ holds. The same is true for

$$\sin\left(4\theta\right)=\frac{2\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\operatorname{sgn}\left(\sum_h\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\right)^2}}.$$

Note that $\sin^2\left(2\theta\right)$ and $\sin\left(4\theta\right)$ do not depend on $\operatorname{sgn}\left(\sum_h\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)=\pm 1$, because of the same periodicity considerations as when dealing with only a single matrix. We compute

$$\begin{aligned}
&\sum_{h=1}^{n}\left(\Delta_D^2\left(A''_h\right)-\Delta_D^2\left(A'_h\right)\right)= \\
=\;& \frac{1}{2}\sin^2\left(2\theta\right)\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)-\frac{1}{2}\sin\left(4\theta\right)\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right) \\
=\;& \frac{1}{4}\left(1-\frac{\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\right)^2}}\right)\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)- \\
& \frac{\left(\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\right)^2}{\sqrt{\left(\sum_{h=1}^{n}\left(\left|d'_h\right|^2-\left|\xi'_h\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}'_h\xi'_h\right)\right)^2}}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{4}\left(\sum_{h=1}^{n}\left(\left|d_h^{'}\right|^2-\left|\xi_h^{'}\right|^2\right)-\frac{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2}{\sqrt{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}}\right)- \\
&\quad \frac{\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}{\sqrt{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}} \\
&= \frac{1}{4}\left(\sum_{h=1}^{n}\left(\left|d_h^{'}\right|^2-\left|\xi_h^{'}\right|^2\right)-\frac{\left(\sum_{h=1}^{n}\left(\left|d_h^{'}\right|^2-\left|\xi_h^{'}\right|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}{\sqrt{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}}\right) \\
&= \frac{1}{4}\left(\sum_{h=1}^{n}\left(\left|d_h^{'}\right|^2-\left|\xi_h^{'}\right|^2\right)-\sqrt{\left(\sum_{h=1}^{n}\left(|d_h'|^2-|\xi_h'|^2\right)\right)^2+4\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h'\xi_h'\right)\right)^2}\right)
\end{aligned}
$$

So it follows that $\sum_{h=1}^{n}\left(\Delta_D^2\left(A_h^{''}\right)-\Delta_D^2\left(A_h^{'}\right)\right)$ maintains the same structure as in the case of only one matrix (compare equation 5.15). Consequently, all following steps of the proof can be carried out in the same fashion. So finally, we obtain

$$
\varphi=\arctan\left(\frac{\Im\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'}-a_{h,qp}^{'}\right)\sqrt{\sum_{h=1}^{n}\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}}\right)}{\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'}+a_{h,qp}^{'}\right)\sqrt{\sum_{h=1}^{n}\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}}\right)}\right)
$$

if $\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'}+a_{h,qp}^{'}\right)\sqrt{\sum_{h=1}^{n}\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}}\right)\neq 0$ and
$\varphi=\operatorname{sgn}\left(\Im\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'}-a_{h,qp}^{'}\right)\sqrt{\sum_{h=1}^{n}\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}}\right)\right)\frac{\pi}{2}$
if $\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'}+a_{h,qp}^{'}\right)\sqrt{\sum_{h=1}^{n}\overline{d_h'^2+4a_{h,pq}'a_{h,qp}'}}\right)=0$.

As a last step we will show that the case $\sum_{h=1}^{n}|d_h'|^2=0$ can also be handled with the last expression and does not need special treatment. Suppose that $\sum_{h=1}^{n}|d_h'|^2=0$, then we get $\varphi=\frac{1}{2}\arctan\left(\frac{\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}{\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}\right)$. If we use the equality $\tan\left(\frac{1}{2}\arctan(x)\right)=\frac{\sqrt{x^2+1}-1}{x}$ we further obtain

$$
\begin{aligned}
\tan(\varphi) &= \tan\left(\frac{1}{2}\arctan\left(\frac{\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}{\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}\right)\right) \\
&= \left(\sqrt{\left(\frac{\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}{\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}\right)^2+1}-1\right)\Big/\frac{\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}{\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)} \\
&= \frac{\sqrt{\left(\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)\right)^2+\left(\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)\right)^2}-\sum_{h=1}^{n}\Re\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}{\sum_{h=1}^{n}\Im\left(\bar{a}_{h,qp}'a_{h,pq}'\right)}
\end{aligned}
$$

$$= \frac{\left|\sum_{h=1}^{n}\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\right| - \sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)}{\sum_{h=1}^{n}\Im\left(\bar{a}'_{h,qp}a'_{h,pq}\right)}.$$

Similarly, we obtain

$$\begin{aligned}
\tan(\varphi) &= \frac{\Im\left(\sum_{h=1}^{n}\left(a'_{h,pq}-a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d'^2_h}+4a'_{h,pq}a'_{h,qp}\right)}\right)}{\Re\left(\sum_{h=1}^{n}\left(a'_{h,pq}+a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\left(\overline{d'^2_h}+4a'_{h,pq}a'_{h,qp}\right)}\right)} \\
&= \frac{\left(\Im\left(\sum_{h=1}^{n}\left(a'_{h,pq}-a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\overline{a'_{h,pq}a'_{h,qp}}}\right)\right)^2}{\Re\left(\sum_{h=1}^{n}\left(a'_{h,pq}+a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\overline{a'_{h,pq}a'_{h,qp}}}\right)\Im\left(\sum_{h=1}^{n}\left(a'_{h,pq}+a'_{h,qp}\right)\sqrt{\sum_{h=1}^{n}\overline{a'_{h,pq}a'_{h,qp}}}\right)} \\
&= \frac{\left(\left|\sum_{h=1}^{n}\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\right| - \sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\right)\left(\left|\sum_{h=1}^{n}a'_{h,qp}\right|^2 - \left|\sum_{h=1}^{n}a'_{h,pq}\right|^2\right)}{\sum_{h=1}^{n}\Im\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\left(\left|\sum_{h=1}^{n}a'_{h,qp}\right|^2 - \left|\sum_{h=1}^{n}a'_{h,pq}\right|^2\right)} \\
&= \frac{\left(\left|\sum_{h=1}^{n}\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\right| - \sum_{h=1}^{n}\Re\left(\bar{a}'_{h,qp}a'_{h,pq}\right)\right)}{\sum_{h=1}^{n}\Im\left(\bar{a}'_{h,qp}a'_{h,pq}\right)}.
\end{aligned}$$

We have thus shown that our most general case also covers $\sum_{h=1}^{n}|d'_h|^2 = 0$.

□

Now we can combine the results that we have obtained thus far in order to show how to choose the parameters of the unitary transformation in an optimal way.

Theorem 5.3.22. *Let* $A_1^{'}, ..., A_n^{'} \in \mathrm{Mat}_m(\mathbb{C})$, *let* $1 \leq p < q \leq m$, *let* $d_h^{'} = a_{h,pp}^{'} - a_{h,qq}^{'}$ *and* $\xi_h^{'} = e^{i\varphi} a_{h,qp}^{'} + e^{-i\varphi} a_{h,pq}^{'}$. *Now let*

$$\tan(\varphi) = \frac{\Im\left(\sum_{h=1}^{n}\left(a_{h,pq}^{\prime} - a_{h,qp}^{'}\right)\varrho\right)}{\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{\prime} + a_{h,qp}^{'}\right)\varrho\right)}$$

with $\varrho = \sqrt{\sum_{h=1}^{n}\left(\overline{d_h^{\prime 2} + 4a_{h,pq}^{'} a_{h,qp}^{'}}\right)}$ *if* $\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'} + a_{h,qp}^{'}\right)\sum_{h=1}^{n}\varrho\right) \neq 0$, *and let*

$$\varphi = \mathrm{sgn}\left(\Im\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'} - a_{h,qp}^{'}\right)\varrho\right)\right)\frac{\pi}{2}$$

if $\Re\left(\sum_{h=1}^{n}\left(a_{h,pq}^{'} + a_{h,qp}^{'}\right)\sum_{h=1}^{n}\varrho\right) = 0$. *Furthermore let*

$$\tan(4\theta) = \frac{2\sum_{h=1}^{n}\Re\left(\bar{d}_h^{\prime}\xi_h^{\prime}\right)}{\sum_{h=1}^{n}\left(\left|d_h^{\prime}\right|^2 - \left|\xi_h^{\prime}\right|^2\right)},$$

if $\sum_{h=1}^{n}\left(\left|d_h^{\prime}\right|^2 - \left|\xi_h^{\prime}\right|^2\right) \neq 0$ *and let*

$$\theta = \mathrm{sgn}\left(\sum_{h=1}^{n}\Re\left(\bar{d}_h^{\prime}\xi_h^{\prime}\right)\right)\frac{\pi}{8}$$

if $\sum_{h=1}^{n}\left(\left|d_h^{\prime}\right|^2 - \left|\xi_h^{\prime}\right|^2\right) = 0$.

This choice for φ *and* θ *leads to*

$$\sum_{h=1}^{n}\Delta_D^2\left(A_h^{\prime\prime}\right) \leq \sum_{h=1}^{n}\Delta_D^2\left(A_h^{\prime\prime}\right)$$

with $A_h^{\prime\prime} = U^* A_h^{\prime} U$, *where the definition of* U *is given in 5.3.9.*

Proof. The proof follows directly from Lemma 5.3.19 and Lemma 5.3.21. □

5.3.4 The SIMQDIAG Algorithm

With the help of the definitions and results from the preceding subsection we can formulate the simultaneous quasi-diagonalisation algorithm.

Algorithm 31: Simultaneous Quasi-Diagonalisation Algorithm

Input: A set of diagonalisable matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$, $\varepsilon \in \mathbb{R}^+$

Output: An approximate diagonalisation of $A_1, ..., A_n$, m approximate common eigenvectors of $A_1, ..., A_n$

$P := I_m$, $k := 1$, $d := \infty$;
$A_1^{(1)}, ..., A_n^{(1)} := A_1, ..., A_n$;
while $d > \varepsilon$ **do**
  for $q := 1$ **to** n **do**
    for $p := 1$ **to** q **do**
      Determine the parameters $\alpha_{k,p,q}$, and $y_{k,p,q}$ of $S^{(k,p,q)}$ such that $\sum_{i=1}^{n} \left\| S^{(k,p,q)-1} A_i^{(k)} S^{(k,p,q)} \right\|_E^2$ is approximately minimised via Theorem 5.3.17;
      for $i := 1$ **to** n **do**
        $A_i^{(k)'} := S^{(k,p,q)-1} A_i^{(k)} S^{(k,p,q)}$;
      end
      Determine the parameters $\varphi_{k,p,q}$, and $\theta_{k,p,q}$ of $U^{(k,p,q)}$ such that $\sum_{i=1}^{n} \Delta_D^2 \left(U^{(k,p,q)*} A_i^{(k)'} U^{(k,p,q)} \right)$ is minimised via Theorem 5.3.22;
      for $i := 1$ **to** n **do**
        $A_i^{(k+1)} := U^{(k,p,q)*} A_i^{(k)'} U^{(k,p,q)}$;
      end
      $P := P S^{(k,p,q)} U^{(k,p,q)}$;
      $d := \sum_{i=1}^{n} \Delta_D^2 \left(A_i^{(k+1)} - A_i^{(k)} \right)$;
      $k := k + 1$;
    end
  end
end
return $\left(P, A_1^{(k+1)}, ..., A_n^{(k+1)} \right)$;

Theorem 5.3.23. *This is an algorithm which computes in a finite number of steps m approximate common eigenvectors $P = (v_1, ..., v_m)$ for $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{C})$ such that*

$$\sum_{i=1}^{n} \Delta_D^2 \left(P^{-1} A_i P \right)$$

can no longer be improved by the applied transformations.

Proof. We have shown in Theorem 5.3.22 and Theorem 5.3.17 how to choose the four parameters $\alpha_{k,p,q}$, $y_{k,p,q}$, $\varphi_{k,p,q}$, and $\theta_{k,p,q}$ for each combination k, p, q such that convergence of the

algorithm can be assured. The process becomes stationary either when the matrices are completely diagonalised or no further improvement can be achieved via the applied transformations. This is made sure in line 3 of the algorithm. □

Remark 5.3.24. Even though it seems likely, it is currently unknown and subject of future research if the algorithm will return a local minimum with respect to the cost functions defined by the departure from normality Δ_N and diagonality Δ_D in case the matrices cannot be exactly simultaneously diagonalised. A pivot strategy like in the original Jacobi algorithm might allow for a simple proof.

Remark 5.3.25. It is possible to implement the algorithm in an efficient way by not performing full matrix-matrix multiplications but by only updating the entries of the matrices which change after applying S and U in each iteration (see Propositions 5.3.12 and 5.3.13). Thus we can make use of the fact that the similarity transformations will only alter two rows and columns at a time.

Remark 5.3.26. The efficiency of the algorithm can be further improved if we implement a decoupling strategy. For this purpose see Proposition 2.9.27.

Remark 5.3.27. Another way to improve the performance of the algorithm is to skip an optimization step (i, j) if the (i, j)-th and (j, i)-th entries of all matrices are below are specified threshold value. This approach is inspired by the threshold Jacobi method, which is discussed by Wilkinson in [64, page 277 f.].

5.3.5 Parameter Choice for Real Input Data

In case the input matrices are only real valued, the choice of parameters in the Simultaneous Quasi-Diagonalisation algorithm (31) can be simplified. This permits a more efficient and robust implementation of the algorithm. Let $A_1 ..., A_n \in \mathrm{Mat}_m(\mathbb{R})$ be real matrices which we want to simultaneously quasi-diagonalise. Just like in [53], the real shear and unitary transformations take the following forms. Note that the matrices are again identical to the unit matrix I_m except for four entries.

Definition 5.3.28. [Shear Rotation Matrix]

Let $y_{k,p,q} \in \mathbb{R}$. We call a matrix $S^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{R})$ with entries identical to the unit matrix I_m except for the four entries

$$\begin{aligned} s_{pp}^{(k,p,q)} &= \cosh(y_{k,p,q}), \\ s_{pq}^{(k,p,q)} &= -\sinh(y_{k,p,q}), \\ s_{qp}^{(k,p,q)} &= -\sinh(y_{k,p,q}), \\ s_{qq}^{(k,p,q)} &= \cosh(y_{k,p,q}) \end{aligned}$$

a **real shear rotation matrix** with parameter $y_{k,p,q}$.

Definition 5.3.29. [Unitary Rotation Matrix]

Let $\theta_{k,p,q} \in]-\pi, \pi]$. We call a matrix $U^{(k,p,q)} \in \mathrm{Mat}_m(\mathbb{R})$ with entries identical to the unit matrix I_m except for the four entries

$$\begin{aligned} u_{pp}^{(k,p,q)} &= \cos(\theta_{k,p,q}), \\ u_{pq}^{(k,p,q)} &= -\sin(\theta_{k,p,q}), \\ u_{qp}^{(k,p,q)} &= \sin(\theta_{k,p,q}), \\ u_{qq}^{(k,p,q)} &= \cos(\theta_{k,p,q}) \end{aligned}$$

a **real unitary rotation matrix** with parameter $\theta_{k,p,q}$.

We first state the real version of the Simultaneous Quasi-Diagonalisation Algorithm. Afterwards we explain how the parameters y and θ need to be chosen.

Algorithm 32: Real Simultaneous Quasi-Diagonalisation Algorithm

Input: A set of diagonalisable matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{R})$, $\varepsilon \in \mathbb{R}^+$

Output: An approximate diagonalisation of $A_1, ..., A_n$, m approximate common eigenvectors of $A_1, ..., A_n$

$P := I_m$, $k := 1$, $d := \infty$;
$A_1^{(1)}, ..., A_n^{(1)} := A_1, ..., A_n$;
while $d > \varepsilon$ **do**
  for $q := 1$ **to** n **do**
    for $p := 1$ **to** q **do**
      Determine the parameter $y_{k,p,q}$ of $S^{(k,p,q)}$ such that $\sum_{i=1}^n \left\| S^{(k,p,q)-1} A_i^{(k)} S^{(k,p,q)} \right\|_E^2$ is approximately minimised via Corollary 5.3.30;
      for $i := 1$ **to** n **do**
        $A_i^{(k)'} := S^{(k,p,q)-1} A_i^{(k)} S^{(k,p,q)}$;
      end
      Determine the parameter $\theta_{k,p,q}$ of $U^{(k,p,q)}$ such that $\sum_{i=1}^n \Delta_D^2 \left(U^{(k,p,q)\,\mathrm{tr}} A_i^{(k)'} U^{(k,p,q)} \right)$ is minimised via Corollary 5.3.31;
      for $i := 1$ **to** n **do**
        $A_i^{(k+1)} = U^{(k,p,q)\,\mathrm{tr}} A_i^{(k)'} U^{(k,p,q)}$;
      end
      $P := P S^{(k,p,q)} U^{(k,p,q)}$;
      $d := \sum_{i=1}^n \Delta_D^2 \left(A_i^{(k+1)} - A_i^{(k)} \right)$;
      $k := k + 1$;
    end
  end
end
return $\left(P, A_1^{(k+1)}, ..., A_n^{(k+1)} \right)$;

Just as in the complex case, we introduce the following abbreviations:

$$\begin{aligned} d_h &= a_{h,pp} - a_{h,qq}, \\ \xi_h &= -ia_{h,qp} + ia_{h,pq}, \\ d'_h &= a'_{h,pp} - a'_{h,qq}, \\ \xi'_h &= a'_{h,qp} + a'_{h,pq}, \\ G_h &= G_{h,pq} = \sum_{j \neq p,q} \left(a^2_{h,pj} + a^2_{h,jp} + a^2_{h,jq} + a^2_{h,qj} \right), \\ C_h &= A_h A_h^{\mathrm{tr}} - A_h^{\mathrm{tr}} A_h, \\ c_h &= C_{h,pq}. \end{aligned}$$

Corollary 5.3.30. *Let $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{R})$ and let $1 \leq p < q \leq m$. Furthermore, let*

$$\tanh(y) = \frac{-\left|\sum_{h=1}^n c_h\right|}{\sum_{h=1}^n \left(G_h + 2\left(d_h^2 + |\xi_h|^2 \right) \right)}.$$

For this choice of y, the inequality

$$\sum_{h=1}^n \left\| A'_h \right\|_E^2 \leq \sum_{h=1}^n \|A_h\|_E^2$$

holds, where $A'_h = S^{-1} A_h S$ with the definition of S given in 5.3.28.

Proof. The claim follows directly from the definitions and from Theorem 5.3.17 if we let $\alpha_{k,p,q} = -\frac{\pi}{2}$. □

Corollary 5.3.31. *Let $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{R})$ and let $1 \leq p < q \leq m$. Furthermore, let*

$$\tan(4\theta) = \frac{2\sum_{h=1}^n d'_h \xi'_h}{\sum_{h=1}^n \left(d'^2_h - \xi'^2_h \right)}.$$

This choice leads to

$$\sum_{h=1}^n \Delta_D^2 \left(A''_h \right) \leq \sum_{h=1}^n \Delta_D^2 \left(A'_h \right),$$

with $A''_h = U^* A'_h U$, where the definition of U is given in 5.3.29.

Proof. The claim follows directly from the definitions and from Theorem 5.3.22 if we let $\varphi_{k,p,q} = 0$. □

Now we come back to Example 5.2.8. This time we use the Simultaneous Quasi-Diagonalisation Algorithm (31), as implemented in the ApCoCoA library, to solve the rational recovery problem.

Example 5.3.32. Let $P = \mathbb{R}[x_1, x_2]$, $\mathcal{O} = \{1, x_2, x_1, x_1x_2\}$, and let $G = \{g_1, ..., g_4\}$ be an approximate $\mathcal{O}$-border basis with

$$\begin{aligned} g_1 &\approx x_2^2 - 1.026x_2 + 0.063, \\ g_2 &\approx x_1^2 + 0.060x_1x_2 - 1.056x_1 - 0.032x_2 + 0.079, \\ g_3 &\approx x_1x_2^2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018, \\ g_4 &\approx x_1^2x_2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018. \end{aligned}$$

The corresponding multiplication matrices have the following structure:

$$A_1 = \begin{pmatrix} 0 & 0 & -0.079 & -0.018 \\ 0 & 0 & 0.032 & -0.012 \\ 1 & 0 & 1.056 & -0.012 \\ 0 & 1 & -0.060 & 1.025 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & -0.063 & 0 & -0.018 \\ 1 & 1.026 & 0 & -0.012 \\ 0 & 0 & 0 & -0.012 \\ 0 & 0 & 1 & 1.025 \end{pmatrix}.$$

As $\|A_1A_2 - A_2A_1\|_\delta \approx 0.054$, we are dealing with a 0.06-approximate border basis.

Now we apply the Simultaneous Quasi-Diagonalisation algorithm (31) and get the approximate eigenvectors

$$V = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} \approx \begin{pmatrix} 1.170 & 0.073 & 0.89 & 0.021 \\ 1.010 & 0.981 & 0.053 & 0.032 \\ 1.015 & 0.060 & 0.993 & 0.031 \\ 1.140 & 1.090 & 1.091 & 1.067 \end{pmatrix} \in \mathrm{Mat}_4(\mathbb{R}).$$

for the transposed multiplication matrices. Next, we compute

$$VA_1V^{-1} \approx \begin{pmatrix} 0.061 & 0.021 & 0 & -0.003 \\ 0.016 & 0.031 & 0.003 & 0 \\ 0 & 0.003 & 1.008 & -0.029 \\ 0.01 & 0 & -0.038 & 0.980 \end{pmatrix}$$

and

$$VA_2V^{-1} \approx \begin{pmatrix} 0.048 & 0 & 0.020 & -0.003 \\ 0 & 0.992 & 0.010 & -0.029 \\ 0.030 & -0.004 & 0.030 & -0.001 \\ -0.003 & -0.021 & 0 & 0.980 \end{pmatrix}.$$

Via the approximate eigenvalues (compare Theorem 5.1.9), we obtain the points $p_1 = (0.061, 0.048)$, $p_2 = (0.031, 0.992)$, $p_3 = (1.008, 0.03)$, and $p_4 = (0.98, 0.98)$. Using the Buchberger-Möller algorithm for border bases (18) together with the basis transformation algorithm (19) we obtain an exact $\mathcal{O}$-border basis $\tilde{G}$ which has the following multiplication matrices (we only give 3 digits after the comma):

$$\tilde{A}_1 \approx \begin{pmatrix} 0 & 0 & -0.064 & -0.001 \\ 0 & 0 & 0.033 & -0.029 \\ 1 & 0 & 1.072 & 0.002 \\ 0 & 1 & -0.060 & 1.009 \end{pmatrix} \quad \tilde{A}_2 \approx \begin{pmatrix} 0 & -0.048 & 0 & -0.001 \\ 1 & 1.042 & 0 & 0.001 \\ 0 & 0.018 & 0 & -0.029 \\ 0 & -0.031 & 1 & 1.009 \end{pmatrix}.$$

We observe that $\|A_1 - \tilde{A}_1\|_\delta \approx 0.032$ and $\|A_2 - \tilde{A}_2\|_\delta \approx 0.042$, which is reasonably close to 0.054.

Finally, we pick up Example 9.7 from Stetter in [49], where he tries to iteratively refine a given almost exact border basis (i.e. a border basis that was computed in floating point arithmetic) in such a way that it is even closer to an exact border basis. We show that the Simultaneous Quasi-Diagonalisation algorithm (31) is superior to the method of Stetter in the sense that the computed solution is both exact and closer to the original approximate border basis.

Example 5.3.33. Just like Stetter in [49, Example 9.7] we let $P = \mathbb{R}[x_1, x_2]$, we let $\mathcal{O} = \{1, x_2, x_1, x_2^2, x_1x_2, x_1^2\}$, and we let $G = \{g_1, ..., g_4\}$ be an approximate $\mathcal{O}$-border basis with

$$\begin{aligned}
g_1 &= x_2^3 - 0.18428x_1^2 + 0.27064x_1x_2 + 0.17425x_2^2 + 0.39572x_1 - 1.31238x_2 + 0.17877,\\
g_2 &= x_1x_2^2 + 0.38056x_1^2 - 0.43946x_1x_2 - 0.54850x_2^2 - 1.33649x_1 + 0.20184x_2 + 0.57210,\\
g_3 &= x_1x_2^2 - 0.34518x_1^2 - 0.74861x_1x_2 - 5.31753x_2^2 - 1.08966x_1 - 2.44432x_2 + 6.20218,\\
g_4 &= x_1^3 - 1.22257x_1^2 - 0.34229x_1x_2 + 2.95924x_2^2 - 3.59317x_1 + 1.32856x_2 - 1.26231.
\end{aligned}$$

We observe that G is a $9.3 \cdot 10^{-5}$-approximate border basis. Stetter computes the following iteratively refined approximate border basis (6 digits after the comma)

$$\begin{aligned}
\bar{g}_1 &\approx x_2^3 - 0.184281x_1^2 + 0.270637x_1x_2 + 0.174249x_2^2 + 0.395719x_1 - 1.312377x_2 + 0.178772,\\
\bar{g}_2 &\approx x_1x_2^2 + 0.380565x_1^2 - 0.439471x_1x_2 - 0.54876x_2^2 - 1.336477x_1 + 0.201853x_2 + 0.572074,\\
\bar{g}_3 &\approx x_1^2x_2 - 0.345205x_1^2 - 0.748649x_1x_2 - 5.317555x_2^2 - 1.089621x_1 - 2.444281x_2 + 6.202176,\\
\bar{g}_4 &\approx x_1^3 - 1.222582x_1^2 - 0.342329x_1x_2 + 2.959279x_2^2 - 3.593209x_1 + 1.328599x_2 - 1.262349.
\end{aligned}$$

With the help of algorithms 31, 18, and 19 we compute (6 digits after the comma)

$$\begin{aligned}
\tilde{g}_1 &\approx x_2^3 - 0.184281x_1^2 + 0.270640x_1x_2 + 0.174257x_2^2 + 0.395722x_1 - 1.312375x_2 + 0.178763,\\
\tilde{g}_2 &\approx x_1x_2^2 + 0.380562x_1^2 - 0.439458x_1x_2 - 0.548485x_2^2 - 1.336482x_1 + 0.201848x_2 + 0.572081,\\
\tilde{g}_3 &\approx x_1^2x_2 - 0.345183x_1^2 - 0.748620x_1x_2 - 5.317537x_2^2 - 1.089656x_1 - 2.444319x_2 + 6.202178,\\
\tilde{g}_4 &\approx x_1^3 - 1.222572x_1^2 - 0.342286x_1x_2 + 2.959206x_2^2 - 3.593193x_1 + 1.328540x_2 - 1.262263.
\end{aligned}$$

In terms of the associated multiplication matrices, which are defined in the usual way, we obtain $\|A_1 - \bar{A}_1\|_\delta \approx 8.8 \cdot 10^{-5}$, $\|A_2 - \bar{A}_2\|_\delta \approx 7.6 \cdot 10^{-5}$, $\|A_1 - \tilde{A}_1\|_\delta \approx 6.6 \cdot 10^{-5}$, and $\|A_2 - \tilde{A}_2\|_\delta \approx 2.7 \cdot 10^{-5}$. This shows that the result computed by the Simultaneous Quasi-Diagonalisation algorithm is significantly closer to the given approximate border basis, than the result of Stetter.

5.3.6 Comparison with other Approaches

In this subsection we compare the numerical performance of the Simultaneous Quasi-Diagonalisation (SimQDiag) algorithm (31) and the shear rotation algorithm of Fu and Gao that was proposed in [53]. For this purpose the author has implemented the algorithm from [53] as there is no publicly available version at the time of writing.

Example 5.3.34. Let

$$A_1 = \begin{pmatrix} 11.33343 & -4.9998 & -6.33323 & 4.99999 \\ 26.66676 & -11.0002 & -11.66661 & 9.00001 \\ 48.33323 & -22.9998 & -16.33328 & 14.00001 \\ 61.66676 & -26.9998 & -19.66676 & 15.99999 \end{pmatrix},$$

$$A_2 = \begin{pmatrix} 6.55565 & 0.0001 & -3.55545 & 1.99999 \\ 8.33343 & 3.9998 & -5.33328 & 2.00001 \\ 15.55545 & 0.0002 & -3.55550 & 2.00001 \\ 18.33343 & 3.0002 & -6.33343 & 2.99999 \end{pmatrix},$$

and let

$$A_3 = \begin{pmatrix} 21.0001 & -19.9998 & -3.9999 & 8.0001 \\ 56.66676 & -53.0002 & -14.66661 & 24.00001 \\ 99.9999 & -92.0002 & -28.99998 & 44.00001 \\ 136.66676 & -125.9998 & -40.66676 & 60.99999 \end{pmatrix}.$$

We apply both Algorithm 31 and the shear rotation algorithm of Fu and Gao with a varying number of iterations. Both algorithms compute approximate eigenvectors $(v_1, ..., v_4) = V$ such that the matrix product $V^{-1}A_iV$ is almost diagonal for $1 \leq i \leq 3$. After each iteration k we measure the common squared departure from diagonality (see Definition 5.3.7), i.e. we compute $\sum_{i=1}^{3} \Delta_D^2 \left(V^{-1}A_iV\right)$.
The numbers starting with $k = 5$ are presented in Table 5.1. We observe that the process becomes essentially stationary after 11 iterations for both algorithms. Notably the SimQDiag algorithm has better convergence properties than the shear rotation algorithm. For instance, after 5 iterations the total squared departure from diagonality is only 154.1 compared to the 195.2 when using the shear rotation algorithm. Finally we note, that the result to which the Simultaneous Quasi-Diagonalisation algorithm has converged is smaller than the result computed by the shear rotation algorithm.

	Iterations							
	5	6	7	8	9	10	11	12
ShearRotation	195.2	76.0	22.4	4.3	0.4	0.013	0.003	0.003
SimQDiag	154.1	54.6	14.5	2.4	0.2	0.003	0.001	0.001

Table 5.1: Total squared departure from diagonality after k iterations. See Example 5.3.34.

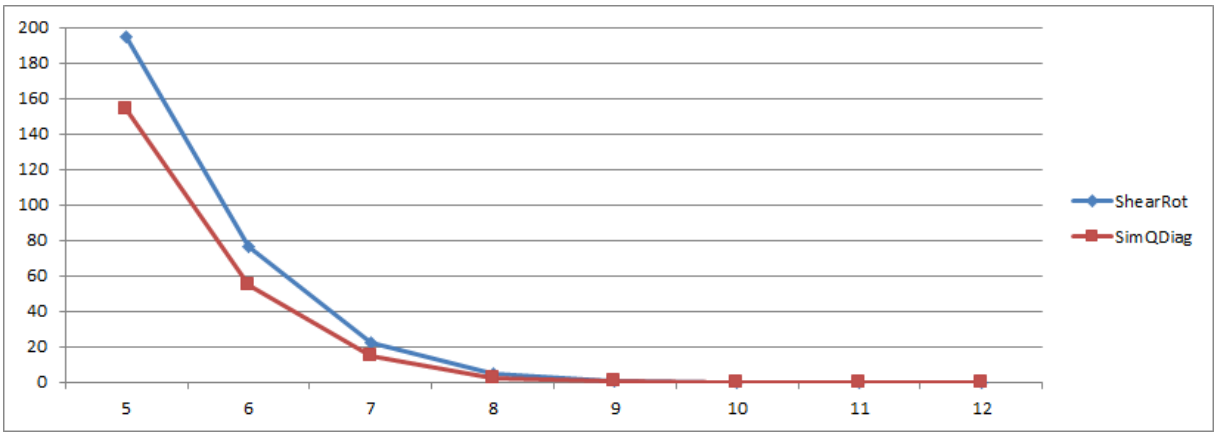

Figure 5.4: Graph of the total squared departure from diagonality after k iterations. See Example 5.3.34.

Finally, we consider the performance of the algorithms on a set of random simultaneously diagonalisable matrices to which we add some Gaussian noise.

Example 5.3.35. First we describe how we generate the input data for the example computation. Let

$$V = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 9 \\ 10 & 11 & 12 & 13 \\ 15 & 15 & 16 & 18 \end{pmatrix} \in \mathrm{Mat}_4(\mathbb{R}).$$

Then we create 5 diagonal matrices $\Lambda_1, ..., \Lambda_5$ that have uniformly random diagonal entries in the discrete set $\{-100, -99.9, -99.8..., 99.9, 100\}$. Now we form the matrices $A_i = V\Lambda_1 V^{-1}$ for $1 \leq i \leq 5$ and add Gaussian noise. We denote the resulting matrices by $\tilde{A}_i$. Afterwards, we apply both the shear rotation and the simultaneous diagonalisation algorithm to the matrices $\tilde{A}_1, ..., \tilde{A}_5$ with a varying number of iterations. In each case we obtain an invertible matrix $\tilde{V}$ and its inverse $\tilde{V}^{-1}$. Then we compute the total squared departure from diagonality for the matrices $\tilde{\Lambda}_i = \tilde{V}^{-1}\tilde{A}_1\tilde{V}$, i.e. we compute $\sum_{i=1}^{5} \Delta_D^2(\tilde{\Lambda}_i)$. This procedure of taking random diagonal matrices $\Lambda_1, ..., \Lambda_5$ together with the following steps is repeated 30 times and the average values for the total departure from diagonality are computed for each iteration k.
The results can be found in Table 5.2. Again we observe that the Simultaneous Quasi-Diagonalisation Algorithm converges faster than the shear rotation algorithm. For instance after 9 iterations the average value of the total squared departure from diagonality is about 282 for the Simultaneous Quasi-Diagonalisation Algorithm versus 11167 for the shear rotation algorithm. Additionally, after convergence the average total squared departure from diagonality was smaller for the Simultaneous Quasi-Diagonalisation Algorithm.

	6	7	8	9	10	11	12	13
ShearRotation	581541	217319	64161	11167	739	6.3	0.29	0.29
SimQDiag	244072	45754	5261	282	5.6	0.3	0.21	0.21

Table 5.2: Total squared departure from diagonality after k iterations. See Example 5.3.35.

Remark 5.3.36. Fu and Gao have pointed out in [53, Section 4] that their shear rotation algorithm converges faster than the simultaneous Schur decomposition algorithm (SSD) which was proposed in [55] and the simultaneous QR decomposition algorithm which was introduced in [56]. As we have shown that our algorithm outperforms the shear rotation algorithm at least in the given examples, we can expect to outperform the other mentioned algorithms as well.

5.4 A Sum of Squares Heuristic for the Rational Recovery Problem

In this section we present an alternative route to the solution of the rational recovery problem. The idea is to transform the problem of finding a suitable set of points to finding the minima of

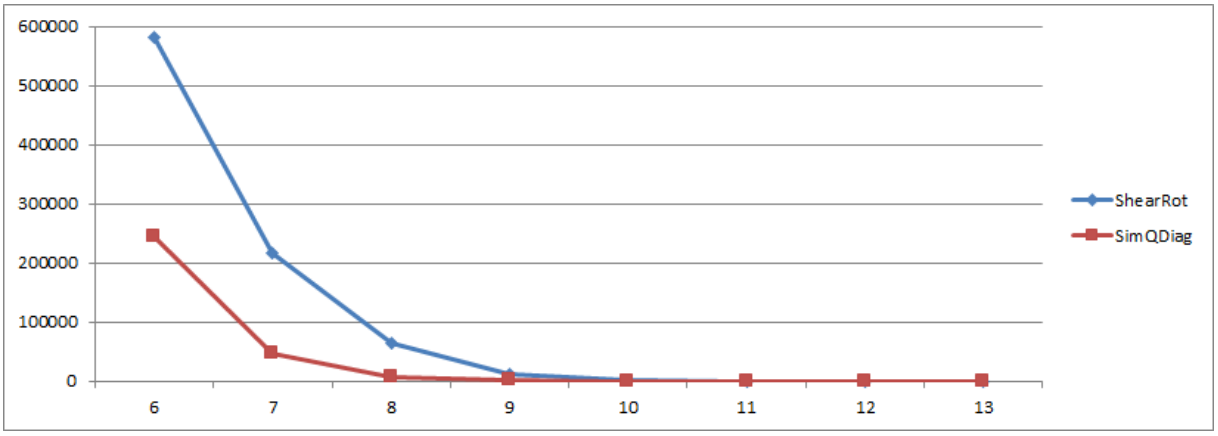

Figure 5.5: Graph of the total squared departure from diagonality after k iterations. See Example 5.3.35.

a sums of squares expression, which we obtain from G. No rigorous proof will be given, but we motivate the usefulness of the heuristic and present some numerical evidence for its adequacy and efficiency.

Before we start let us briefly recall the rational recovery problem. Let $P = K[x_1, ..., x_n]$ with $K = \mathbb{C}$ or $K = \mathbb{R}$, let $\mathcal{O}$ be an order ideal, and let $G = \{g_1, ..., g_\nu\} \subset P$ be an ε-approximate $\mathcal{O}$-border basis. We are interested in finding an exact $\mathcal{O}$-border basis $\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_\nu\}$ which is close to G with respect to the Euclidean distance of all coefficient vectors of the polynomials g_j and $\tilde{g}_j$. The method discussed in the previous sections used the approximate common eigenvalues and eigenvectors of the multiplication matrices A_i as candidates for the common zeros of an exact border basis. After these have been computed, the Buchberger-Möller algorithm (18) together with the Border Basis transformation algorithm (19) can be used to recover an exact $\mathcal{O}$-border basis which is close to the original one.

The following approach also leads to a suitable set of points which serves as input for the Buchberger-Möller algorithm. However, instead of relying on the multiplication matrices, we focus on the polynomials in G itself. Our aim is to look for points which are as close as possible to the common zeros of G. In other words, we try to find points which minimise the total residual of all polynomials when evaluated on one of these points. For this purpose, we consider the sum of the squares of all polynomials, which we will denote by S, in G and try to find the local minima of this single equation.

First we describe a method, which uses the common zeros of the derivatives of S with respect to $x_1, ..., x_n$. The obvious disadvantage of this approach is, that generally we have to deal with the NP-hard problem of solving a multivariate quadratic system which has apparently no special structure that could be exploited. Only small systems of equations can be treated in this manner.

The second method that we propose can be applied if G is obtained via a variant of the Buchberger-Möller algorithm (for example, the ABM algorithm (22) or AVI algorithm (21)). Then we already know a set of points $\mathbb{X}$, such that $\|\text{eval}_{\mathbb{X}}(g_i)\|$ is small for all $g_i \in G$. Using each point in $\mathbb{X}$ as a starting value we search in its vicinity for a local minimum, with the help

of numerical local optimisation techniques. In this setting we can even handle industrial size datasets with several thousand points. A disadvantage, though, is that we no longer have a guarantee to find all local minima of S.

In both cases we do not give exact error bounds for the algorithms, but we provide a set of examples that illustrate the effectiveness and adequacy of the methods.

Before we start to outline the algorithm we explain how to find the minima of a complex polynomial function which contains both x_i and its complex conjugate $\bar{x}_i$. A detailed overview of the technique can be found in [68, Theorem 2], where also some theoretical justification is provided.

Remark 5.4.1. We want to find the local minima of a real valued polynomial function $S : K^n \to \mathbb{R}$ by computing its derivatives with respect to $x_1, ..., x_n$. If $K = \mathbb{R}$ we observe that $x_i = \bar{x}_i$. Thus we can just compute the derivatives of S with respect to $x_1, ..., x_n$ and determine for each point in $\mathcal{Z}\left(\left\langle \frac{\partial S}{\partial x_1}, ..., \frac{\partial S}{\partial x_n} \right\rangle\right)$ if it is a local minimum via the usual criteria. However, if $K = \mathbb{C}$ we have to be more careful since $x_i \to \overline{x}_i$ is not an analytic function in x_i and therefore one cannot calculate derivatives in a straightforward way. Fortunately, it is still possible to compute the minima in a similar way. Following [68, Section 1] there are basically two solution strategies to this problem. First it would be possible to treat every complex variable as two real variables, but this would be quite tedious. The second and more elegant approach is to treat x_i and $\bar{x}_i$, for every i, as if they were independent variables and calculate the derivatives for x_i and $\bar{x}_i$ independently. As S is real valued, it even suffices to compute the derivatives either for x_i or $\bar{x}_i$ (see [68, Theorem 2]). For convenience we choose for x_i and pay no attention to $\bar{x}_i$. Then we determine again for each point in $\mathcal{Z}\left(\left\langle \frac{\partial S}{\partial x_1}, ..., \frac{\partial S}{\partial x_n} \right\rangle\right)$ which points are in fact local minima of S.

Algorithm 33: Sum of Squares Minimisation Algorithm

Input: An ε-approximate $\mathcal{O}$-border basis $G = \{g_1, ..., g_\nu\} \subset K[x_1, ..., x_n]$ with $K = \mathbb{R}$ or $K = \mathbb{C}$ for an ideal I such that $|\mathcal{Z}(I)| = \mu$, number of significant digits after the comma $d \in \mathbb{N}_0$, and the number of elements in the order ideal $\mu = |\mathcal{O}|$

Output: A list of μ points $\mathbb{X}$, such that all polynomials in G vanish approximately on $\mathbb{X}$ or an error message if less than μ local minima of S exist

$S := \sum_{i=1}^{\nu} \bar{g}_i g_i$;
$D := \left\{ \frac{\partial S}{\partial x_1}, ..., \frac{\partial S}{\partial x_n} \right\}$ via Remark 5.4.1;
$Z := \{z_1, ..., z_k\}$ approximations of the zeros of D with d significant digits after the comma, compted e.g. via Bertini [65];
$M := \{p_1, ..., p_\gamma\} \subseteq Z$ the points in Z that are local minima of S;
if $|M| < \mu$ **then**
    Return "No solution could be computed.";
else
    $\tilde{M} := [\tilde{p}_1, ..., \tilde{p}_\gamma]$ the points in M ordered ascendingly with respect to $\|\text{eval}_{\tilde{p}_i}(S)\|$;
    $\mathbb{X} := [\tilde{p}_1, ..., \tilde{p}_\mu]$ the first μ points from $\tilde{M}$;
end
return $\mathbb{X}$;

Remark 5.4.2. Unfortunately, it is currently unknown whether enough local minima always have to exist such that we can pick μ points. Because of this reason the algorithm may terminate prematurely in line 6. It is one possible direction for future research to construct an example where the algorithm terminates without returning μ points or to prove that it will always find μ minima.

To motivate why the individual steps of the algorithm and the algorithm as a whole make sense, we will that it will always return a correct result (up to rounding errors) if G is an exact border basis for a 0-dimensional ideal I that has only simple roots. For practical purposes it is not recommended to use Algorithm 33 if G is in fact an exact border basis, as there are more efficient techniques available, like notably the eigenvalue algorithm (30).

Proposition 5.4.3. *If G is an exact $\mathcal{O}$-border basis for a 0-dimensional ideal I that has only simple roots, Algorithm 33 computes and returns a numerical approximation of $\mathcal{Z}(\langle G \rangle)$.*

Proof. Let G be an exact $\mathcal{O}$-border basis. Then the evaluation of

$$S = \sum_{i=1}^{\nu} g_i \overline{g}_i = \sum_{i=1}^{\nu} |g_i|^2$$

is equal to zero if and only if the evaluation of each individual polynomial g_i is equal to zero for any point $p \in \mathbb{C}^n$. This is true because the evaluation of $|g_i|^2$ on any point in $\mathbb{C}^n$ is obviously greater than or equal to zero. As $|\mathcal{O}| = \mu$, the set G has exactly μ distinct zeros which are also zeros of S. As S cannot have any more zeros, the zero set of S and the zero set of G must agree. The polynomial S has exactly μ local minima $p_1, ..., p_\mu$ such that $\text{eval}_{p_i}(S) = 0$. Therefore the points returned by the algorithm will be the zeros of G. □

Remark 5.4.4. It is possible to extend Algorithm 33 to 0-dimensional ideals that have zeros of multiplicity greater than 1. For this purpose the multiplicity of each common zero p_i of $\langle D \rangle$ has to be computed as well in line 3 of the algorithm. Then the multiplicity of p_i in $\langle D \rangle$ needs to be related to the (approximate) multiplicity of p_i in $\langle G \rangle$. This is difficult as p_i is in general only a local minimum of S and not an exact zero of $\langle G \rangle$. Furthermore, computing the multiplicity of a root for a polynomial system is a non-trivial and computationally expensive task. A notable attempt in this direction has been made by Sommese, Verschelde and Wampler in [66]. For our computations and experiments (see Example 5.4.5) we have used their software tool Bertini ([65]). Note, that in case we want to compute an $\mathcal{O}$-border basis for a 0-dimensional ideal for a given set of points, such that some points have multiplicity greater than 1, a generalised version of the Buchberger-Möller algorithm, as described by Abbott et al. in [69], can be used to compute the exact Gröbner basis of the ideal first which can later on be transformed into an $\mathcal{O}$-border basis.

Example 5.4.5. Let $P = \mathbb{R}[x_1, x_2]$, let $\mathcal{O} = \{1, x_1, x_2, x_1^2\}$, and let $G = \{g_1, .., g_4\}$ be the set of polynomials consisting of

$$\begin{aligned} g_1 &= x_2^2, \\ g_1 &= x_1 x_2, \\ g_1 &= x_1^2 x_2, \\ g_1 &= x_1^3 - 2x_1^2 + x_1. \end{aligned}$$

Then G is an exact $\mathcal{O}$-border basis. We now apply Algorithm 33 together with the ideas from Remark 5.4.4 to compute the zero set of G. First we form

$$S = x_2^4 + x_1^2x_2^2 + x_1^4x_2 + x_1^6 - 4x_1^5 + 6x_1^4 - 4x_1^3 + x_1^2$$

and compute $D = \{d_1, d_2\}$ where

$$\begin{aligned} d_1 &= \tfrac{\partial S}{\partial x_1} = 6x_1^5 + 4x_1^3x_2^2 - 20x_1^4 + 24x_1^3 + 2x_1x_2^2 - 12x_1^2 + 2x_1, \\ d_2 &= \tfrac{\partial S}{\partial x_2} = 2x_1^4x_2 + 2x_1^2x_2 + 4x_2^3. \end{aligned}$$

With the help of Bertini ([65]) or some comparable program, we compute the real zero set of D and obtain $p_1 \approx (1,0)$ with multiplicity $m(p_3) = 3$, $p_2 \approx (0.333, 0)$ with $m(p_2) = 1$, and $p_3 = (0,0)$ with $m(p_3) = 3$. The points p_1 and p_3 are both local minima, whereas p_2 is a local maximum. Because G is in fact an exact border basis we determine that the multiplicity of p_1 and p_2 as roots of G is two. Thus the algorithm returns the tuple $\mathbb{X} = [(0,0),(0,0),(1,0),(1,0)]$. Clearly, we could have obtained this result if we had computed the zero set of G directly.

In the next example we slightly perturb the exact system from Example 5.4.5. Now it is no longer possible to use standard exact algorithms to recover the zero set of G.

Example 5.4.6. Let $P = \mathbb{R}[x_1, x_2]$, let $\mathcal{O} = \{1, x_1, x_2, x_1^2\}$, and let $G = \{g_1, .., g_4\}$ with

$$\begin{aligned} g_1 &= x_2^2 - 0.002x_1 + 0.0002x_2 + 0.001, \\ g_2 &= x_1x_2 + 0.002x_1 + 0.0003x_2 - 0.002, \\ g_3 &= x_1^2x_2 - 0.003x_1 - 0.0002x_2 + 0.002, \\ g_4 &= x_1^3 - 1.998x_1^2 + 0.999x_1 - 0.0001x_2 + 0.003 \end{aligned}$$

be an approximate $\mathcal{O}$-border basis. The zero set of G is empty. Now we apply Algorithm 33 again. We form

$$\begin{aligned} S \approx\ & x_1^6 + x_1^4x_2^2 - 3.996x_1^5 + 5.99x_1^4 - 0.0062x_1^3x_2 + 0.9996x_1^2x_2^2 + x_2^4 - 3.986x_1^3 + \\ & 0.00839x_1^2x_2 - 0.0034x_1x_2^2 + 0.0004x_2^3 + 0.98603x_1^2 - 0.00419x_1x_2 + \\ & 0.002x_2^2 + 0.00597x_1 + 0.00001 \end{aligned}$$

and compute $D = \{d_1, d_2\}$ with

$$\begin{aligned} d_1 = \frac{\partial S}{\partial x_1} \approx\ & 6x_1^5 + 4x_1^3x_2^2 - 19.98x_1^4 + 23.96x_1^3 - 0.0186x_1^2x_2 + 1.9992x_1x_2^2 - \\ & 11.958x_1^2 + 0.0167x_1x_2 - 0.0034x_2^2 + 1.972x_1 - 0.0041x_2 + 0.0059, \\ d_2 = \frac{\partial S}{\partial x_2} \approx\ & 2x_1^4x_2 - 0.0062x_1^3 + 1.9992x_1^2x_2 + 4x_2^3 + \\ & 0.0083x_1^2 - 0.0068x_1x_2 + 0.0012x_2^2 - 0.0041x_1 + 0.004x_2. \end{aligned}$$

We obtain the points $p_1 \approx (0.998, 0.001)$ with $m(p_1) = 3$, $p_2 \approx (-0.003, -0.003)$, with $m(p_2) = 3$, and $p_3 \approx (0.333, 0)$ with $m(p_3) = 1$. Both p_1 and p_2 are local minima of S

and p_3 is a local maximum. Because of similar observations as in Example 5.4.5, we return $\mathbb{X} = [(0.998, 0), (0.998, 0), (-0.003, -0.003), (-0.003, -0.003)]$. We compute the exact $\mathcal{O}$-border basis $\tilde{G} = \{\tilde{g}_1, ..., \tilde{g}_4\}$ with

$$\begin{aligned}
\tilde{g}_1 &\approx x_2^2 + 0.000x_1^2 - 0.000x_1 + 0.006x_2 + 0.000,\\
\tilde{g}_2 &\approx x_1x_2 + 0.003x_2,\\
\tilde{g}_3 &\approx x_1^2x_2,\\
\tilde{g}_4 &\approx x_1^3 - 1.993x_1^2 + 0.990x_1 + 0.002.
\end{aligned}$$

This is a reasonably good approximation of the original system G. Note that only 3 digits precision were used during the computation. A higher precision would further improve the results.

In the following example we have to deal with complex solutions.

Example 5.4.7. Let $P = \mathbb{R}[x_1, x_2]$. Consider the order ideal $\mathcal{O} = \{1, x_1, x_2\}$ and the set of polynomials $G = \{g_1, g_2, g_3\}$, where $g_1 = x_2^2 - 0.99x_2$, $g_2 = x_1x_2 + 0.01$ and $g_3 = x_1^2 - 2.01x_2 + 1.99$. We apply Algorithm 33:

1. We have $S = x_1^4 + x_1^2x_2^2 + x_2^4 - 4.02x_1^2x_2 - 1.98x_2^3 + 3.98x_1^2 + 0.02x_1x_2 + 5.0202x_2^2 - 7.9998x_2 + 3.9602$.
2. We calculate $d_1 = \frac{\partial S}{\partial x_1} = 4x_1^3 + 2x_1x_2^2 - 8.04x_1x_2 + 7.96x_1 + 0.02x_2$ and $d_2 = \frac{\partial S}{\partial x_2} = 2x_1^2x_2 + 4x_2^3 - 4.02x_1^2 - 5.94x_2^2 + 0.02x_1 + 10.0404x_2 - 7.9998$ and form $D = \{d_1, d_2\}$.
3. We determine all minima of D up to a precision of 10^{-4} and obtain one real minimum at $p_1 = (-0.0100, 0.9900)$ and two complex conjugate minima at $p_{2,3} = (0.0099, 0.0001) \pm (1.4105, 0.0139)\, i$
4. We return $\mathbb{X} = \{p_1, p_2, p_3\}$.

If we calculate an exact border basis $\tilde{G} = \{\tilde{g}_1, \tilde{g}_2, \tilde{g}_3\}$ for $\mathbb{X}$ with respect to $\mathcal{O}$ we obtain:

$$\begin{aligned}
\tilde{g}_1 &\approx x_2^2 + 0.0097x_1 - 0.99x_2 + 0.0001,\\
\tilde{g}_2 &\approx x_1x_2 - 0.0001x_1 - 0.0098x_2 + 0.0196,\\
\tilde{g}_3 &\approx x_1^2 - 2.0098x_2 + 1.9896.
\end{aligned}$$

The recovered exact border basis is close to the approximate one as

$$\max\left(\|A_1 - \tilde{A}_1\|_\delta, \|A_2 - \tilde{A}_2\|_\delta\right) \approx 0.0137,$$

where the A_i are the multiplication matrices of G and the $\tilde{A}_i$ are the multiplication matrices of $\tilde{G}$.

5.4.1 The Polak-Ribière Conjugate Gradient Algorithm

As a preparation for the approximate version of Algorithm 33, we introduce the Polak-Ribière (PR) conjugate gradient algorithm, a variant of the Fletcher-Reeves algorithm, which is a standard tool in non-linear multivariate optimisation. Though we could use any non-linear optimisation method, we chose PR sine we have to deal with polynomial systems, for which the gradient

can easily be calculated. The PR algorithm makes good use of this additional information to enhance both speed and accuracy. For instance, it features faster convergence compared with the steepest descent algorithm. We only stress the facts which are important in our setting. More details about the method can be found in [67, Section 5.2].

Given a continuous function $f : \mathbb{C}^n \to \mathbb{R}_0^+$ we are interested in finding the local minima of f. We can rewrite f as a real function $\tilde{f} : \mathbb{R}^{2n} \to \mathbb{R}_0^+$ by splitting every complex indeterminate into its real and imaginary part. We are then in the fortunate situation that $\tilde{f}$ is continuously differentiable, and we can apply the widely available (real) implementations of the PR algorithm. Note that the PR algorithm also requires a starting value from where it will start the optimisation process, but the conversion of a point $x \in \mathbb{C}^n$ for f to $\tilde{x} \in \mathbb{R}^{2n}$ for $\tilde{f}$ is of course straightforward. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. Then we let $\partial f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right)$.

Given a starting point $x_0 \in \mathbb{R}^n$ and a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ the following algorithm computes a local minimum of f close to x_0 if it exists.

Algorithm 34: Polak-Ribière Conjugate Gradient Algorithm

Input: A continously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, a starting value $x_0 \in \mathbb{R}^n$, $\varepsilon \geq 0,\ 0 < \sigma < \varrho \leq \frac{1}{2}$

Output: A local minimum x_k of f

1 $k := 0,\ d_0 := -\partial f(x_0) = -\left(\frac{\partial f}{x_0}(), ..., \frac{\partial f}{x_0}()\right) \in \mathbb{R}^n$;

2 **while** $\|\partial f(x_k)\| > \varepsilon$ **do**

3 Choose $t_k > 0$ that satisfies both $f(x_k + t_k d_k) \leq f(x_k) + \sigma t_k \partial f(x_k)^{\mathrm{tr}} d_k$ and $\left|\partial f(x_k + t_k d_k)^{\mathrm{tr}} d_k\right| \leq \varrho \left|\partial f(x_k)^{\mathrm{tr}} d_k\right|$;

4 $x_{k+1} := x_k + t_k d_k$;

5 $\beta_k := \max\left(\frac{\partial f(x_{k+1})(\partial f(x_{k+1}) - \partial f(x_k))}{\|\partial f(x_k)\|^2}, 0\right)$;

6 $d_{k+1} := -\partial f(x_0) + \beta_k d_k$;

7 $k := k + 1$;

8 **end**

9 Return x_k;

Theorem 5.4.8 (Finiteness and convergence)**.** *Let $x_0 \in \mathbb{R}^n$, let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and let $0 < \sigma < \varrho \leq \frac{1}{2}$. If the level set $S = \{x \,|\, f(x) \leq f(x_0)\}$ is bounded and the function f is Lipschitz continuously differentiable in a neighbourhood N of S, i.e. there exists a constant $c > 0$ such that $\|\partial f(x) - \partial f(\tilde{x})\| \leq c\|x - \tilde{x}\|$ for all $x, \tilde{x} \in N$, the algorithm will terminate after a finite number of steps as the following convergence property holds:*

$$\liminf_{k \to \infty} \|\partial f(x_k)\| = 0.$$

Proof. See [67, Theorem 5.8]. □

Remark 5.4.9. The conditions which are checked in line 3 of the algorithm are known as the strong Wolfe conditions and assure that the chosen increment t_k decreases both f and its slope by a "sufficient" amount.

Remark 5.4.10. The parameter ε should be chosen in such a way that it is in the order of magnitude of $\varepsilon_{machine}$. A very small value of may degrade performance of the algorithm, while a too large value may stop the algorithm even though the local minimum has not been reached yet. As long as σ and ϱ are chosen such that $0 < \sigma < \varrho \leq \frac{1}{2}$ holds, then the actual values of σ and ϱ only affect the speed of convergence but not the convergence of the algorithm itself. According to [67, page 124], practical values are $\sigma = 10^{-4}$ and $\varrho = 0.1$.

Now we are in the position to introduce an approximate version of Algorithm 33 which can be applied if a priori a set of points $\tilde{\mathbb{X}}$ is known for which $\left\|\text{eval}_{\tilde{\mathbb{X}}}(g_i)\right\|$ is small for all $g_i \in G$. We are precisely in this situation if the approximate border basis is computed with one of the approximate variants of the Buchberger-Möller algorithm, like for instance the ABM algorithm (22) or AVI algorithm (21). In this case we will also assume that G is close to an exact border basis for a reduced set of points which means that all points will have multiplicity one.

Algorithm 35: Approximate Sum of Squares Minimisation Algorithm

Input: An ε-approximate $\mathcal{O}$-border basis $G = \{g_1, ..., g_\nu\} \subset K[x_1, ..., x_n]$ with $K = \mathbb{R}$ or $K = \mathbb{C}$, the number of elements in the order ideal $|\mathcal{O}| = \mu$, a set of points $\tilde{\mathbb{X}} = \{\tilde{p}_1, ..., \tilde{p}_\tau\}$ with $\tau \geq \mu$, $\varepsilon \geq 0$, and $0 < \sigma < \varrho \leq \frac{1}{2}$

Output: A list of μ points $\mathbb{X}$, such that all polynomials in G vanish approximately on $\mathbb{X}$

$S := \sum_{i=1}^{\nu} \bar{g}_i g_i$;
Convert S to a real polynomial S';
$M := [\,]$; $V := [\,]$;
for *i := 1* **to** τ **do**
  Convert $\tilde{p}_i$ to a real point $\tilde{p}_i'$;
  $p_i' :=$ Polak-Ribière-CG$(S', \tilde{p}_i', \varepsilon, \sigma, \varrho)$;
  Convert p_i' to a point p_i in K^n;
  $v_i := \text{eval}_{p_i}(S)$;
  Append(M, p_i); Append(V, v_i);
end
$\mathbb{X} := [\,]$;
for *i := 1* **to** μ **do**
  $g :=$ a point p_k in M such that v_k is minimal in V;
  Remove(M, p_k); Remove(V, v_k);
  for *j := 1* **to** $|V|$ **do**
    $v_j := v_j / \|p_j - g\|$;
  end
  Append$(\mathbb{X}, g)$;
end
return $\mathbb{X}$;

Let us elaborate on the additional and modified steps compared to Algorithm 33 and explain their purpose.

In line 6 we use the Polak-Ribière algorithm, as it is a well understood and efficient tool in numerical optimisation. The PR algorithm is a general method that can be applied in most situations. However, in some special cases, other methods may be more favourable, e.g. quasi-Newton methods such as the Broyden-Fletcher-Goldfarb-Shanno algorithm (compare [67, Section 8.1] for details). The numerical nature of line 6 leads to two major problems. It is no longer guaranteed that we find all minima, so we may miss out on the globally minimal ones. As with any numerical local optimisation algorithm, its effectiveness greatly depends on the closeness of the starting points to the the actual local minima.
Another issue with which we have to deal is that we have to identify identical minima that were obtained because we had two starting points which were in the neighbourhood of the same minimum. That is why we introduce lines 15 to 17 in our algorithm. They impose a "penalty" on neighbouring points of an already determined minimum. The proposed step is just one possibility and it would be useful to develop more refined methods.

Remark 5.4.11. In case no initial set of starting points $\tilde{\mathbb{X}}$ should be known a priori, it is of course possible to use the Simultaneous Quasi-Diagonalisation Algorithm (31) to obtain a suitable set $\tilde{\mathbb{X}}$.

Remark 5.4.12. In practice it is advisable to use Algorithm 31 to obtain an initial solution and Algorithm 35 to further refine this result if the accuracy of the solution returned by the Simultaneous Quasi-Diagonalisation Algorithm is not satisfactory.

The following examples exemplify the individual steps of the approximate sums of squares minimisation algorithm.

Example 5.4.13. We will apply Algorithm 35 to Example 5.3.32. Let us briefly recall the setting: $P = \mathbb{R}[x_1, x_2]$, $\mathcal{O} = \{1, x_2, x_1, x_1x_2\}$, and $G = \{g_1, ..., g_4\}$ is a 0.06-approximate $\mathcal{O}$-border basis with

$$\begin{aligned} g_1 &\approx x_2^2 - 1.026x_2 + 0.063, \\ g_2 &\approx x_1^2 + 0.060x_1x_2 - 1.056x_1 - 0.032x_2 + 0.079, \\ g_3 &\approx x_1x_2^2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018, \\ g_4 &\approx x_1^2x_2 - 1.025x_1x_2 + 0.012x_1 + 0.012x_2 + 0.018. \end{aligned}$$

The set of starting points is given by $\tilde{\mathbb{X}} = \{(0,0), (0,1), (1,0), (1,1), (0.5, 0.5)\}$. We form

$$S \approx \sum_{i=1}^{4} g_i^2 = x^4y^2 + x^2y^4 - 2.051x^3y^2 - ... - 0.1349y + 0.0111.$$

Next we determine all local minima of S, using the points in $\tilde{\mathbb{X}}$ as starting points, up to a precision of 10^{-4}. We obtain the four distinct minima $p_1 \approx (0.0851, 0.0686)$, $p_2 \approx (0.9625, 0.9691)$, $p_3 \approx (0.0431, 0.9578)$, and $p_4 \approx (0.9797, 0.0502)$. These can also be seen in Figure 5.6, which is a three dimensional visualisation of the zero set of $S - x_3 = 0$. We form $\mathbb{X} = \{p_1, ..., p_4\}$ and

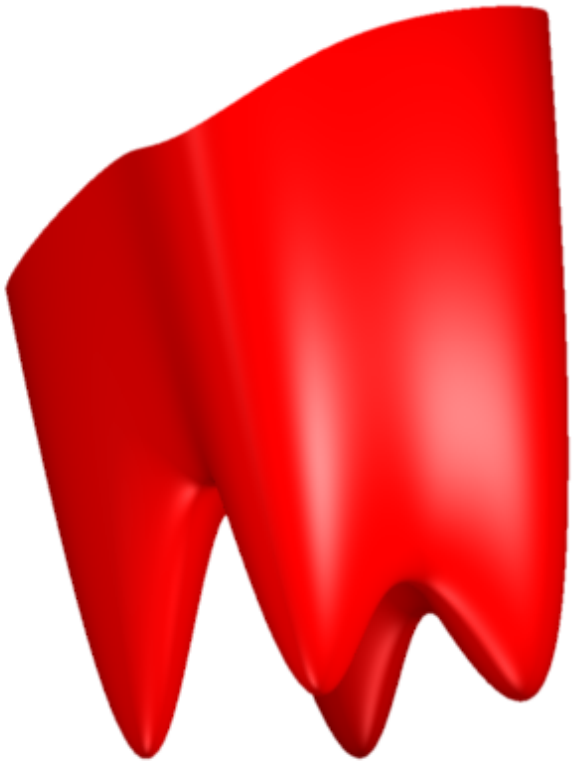

Figure 5.6: Visualisation of $S - x_3 = 0$ for Example 5.4.13.

calculate an exact border basis $\tilde{G} = \{\tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4\}$ for $\mathbb{X}$ with respect to $\mathcal{O}$. Finally we obtain

$$\begin{aligned}
\tilde{g}_1 &\approx x_2^2 + 0.0084x_1x_2 - 0.0192x_1 - 1.0277x_2 + 0.0674,\\
\tilde{g}_2 &\approx x_1^2 + 0.0522x_1x_2 - 1.0557x_1 - 0.0423x_2 + 0.0820,\\
\tilde{g}_3 &\approx x_1x_2^2 - 1.0182x_1x_2 + 0.0471x_1 - 0.0011x_2 + 0.0015,\\
\tilde{g}_4 &\approx x_1^2x_2 - 1.0022x_1x_2 - 0.0032x_1 + 0.0386x_2 + 0.0027.
\end{aligned}$$

We have thus found a close by exact border basis as

$$\max\left(\|A_1 - \tilde{A}_1\|_\delta, \|A_2 - \tilde{A}_2\|_\delta\right) \approx 0.041,$$

where the A_i are the multiplication matrices of G and the $\tilde{A}_i$ are the multiplication matrices of $\tilde{G}$. Note that this result is even a bit better than the one we have obtained with Algorithm 31 in Example 5.3.32.

Remark 5.4.14. [Weighting polynomials]

In some cases, especially in industrial applications, it may be necessary to unevenly distribute the error between the polynomials. For instance, we could want that the lower degree polynomials stay rather stable while we allow a larger degree of freedom in the higher degree ones.

If we have a look again at S, we realise that the polynomials g_i that have larger evaluations in the environment of a point play a more important role and dominate over the polynomials with

the smaller evaluations. As we are looking for minima of S, these will shift closer to the zero set of the polynomial with the larger evaluation in the neighbourhood of its own zero set.

The mentioned behaviour can be used to influence how well the coefficients of a certain polynomial g_i will be recovered. For this purpose, the addend $|g_i|^2$ in S just has to be multiplied by a scalar $w_i \in \mathbb{R}^+$. So if we choose $w_i > 1$ the weight of g_i in the sum will increase and thus the coefficients of g_i will be better recovered. Consequently, if we choose $w_i < 1$ then the weight of g_i in the sum will be reduced. The polynomial S will become $S = \sum_{i=1}^{\nu} w_i \, |g_i|^2$. Afterwards we apply the steps of Algorithm 35 from line 2 onwards. The whole computational process can be repeated several times with different weights $w_1, ..., w_\nu$ until the desired effect is achieved. Of course, the recovery of the polynomials with the smaller weights will suffer, so it greatly depends on the context if and how this technique can be used.

Our final example demonstrates the effect of assigning weights to the polynomials.

Example 5.4.15. In Example 5.4.13 we let $w_1 = 10$, and $w_2 = w_3 = w_4 = 1$. We perform the steps of Algorithm 35 and finally obtain $\tilde{G} = \{\tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4\}$ with

$$\begin{aligned}
\tilde{g}_1 &\approx x_2^2 + 0.0001x_1x_2 - 0.0011x_1 - 1.0260x_2 + 0.0637,\\
\tilde{g}_2 &\approx x_1^2 + 0.0503x_1x_2 - 1.0537x_1 - 0.0418x_2 + 0.0818,\\
\tilde{g}_3 &\approx x_1x_2^2 - 1.0258x_1x_2 + 0.0626x_1,\\
\tilde{g}_4 &\approx x_1^2x_2 - 1.0020x_1x_2 - 0.0032x_1 + 0.0388x_2 + 0.0026.
\end{aligned}$$

We have constructed a close by exact border basis as

$$\max\left(\left\|A_1 - \tilde{A}_1\right\|_\delta, \left\|A_2 - \tilde{A}_2\right\|_\delta\right) \approx 0.054,$$

where the A_i are the multiplication matrices of G and the $\tilde{A}_i$ are the multiplication matrices of $\tilde{G}$. Note that the coefficients of the first polynomial have been recovered better than in Example 5.4.13, as $\|g_1 - \tilde{g}_1\| \approx 0.001$ here and $\|g_1 - \tilde{g}_1\| \approx 0.021$ in Example 5.4.13.

6 Applications

Contents

In this chapter we present some applications of the algorithms which we discussed in this thesis. The primary focus will be on topics related, but not limited to, the oil industry.

Suppose that we are given a set $\mathbb{X} = \{p_1, ..., p_s\}$ of s points in $\mathbb{R}^n$, e.g. s measurements of n physical quantities like pressures, temperatures, valve settings, electrical currents, oil production, gas production, ... A known application of the AVI algorithm, as discussed in [28], is to find physically meaningful polynomial relations among the coordinates of $\mathbb{X}$. For instance, the question if and how the oil (or gas) production of a well can be approximately expressed in terms of the other measurements is of particular interest. The polynomial models that are derived for the oil production of the well may allow a deeper understanding of the underlying physical processes. Another objective is to identify redundant measurements that can already be expressed approximately in terms of other measurements. This information can for example be used to reduce costs because some measurement equipment can be turned off. As these tasks can also be performed via the ABM algorithm, we first discuss how that algorithm can be used in this fashion. Our main emphasis here is that no a priori assumptions are made about the shape of these polynomial models and that they are constructed using only the given data $\mathbb{X}$.

The physical "interpretation" of these models is a non-trivial task and requires domain specific knowledge. In addition to the algorithms that were discussed in Chapter 4, the corresponding technology was also developed in the Algebraic Oil Research Project ([1]). However, this subject is not covered here as these research results are property of Shell and therefore not available to the public. Nevertheless, the ABM algorithm as well as the extended ABM algorithm have proven their practical usefulness for the kind of tasks mentioned above, as they are used as the core algorithms in a proprietary software tool which is now available inside Shell. The so-called

Algebraic Oil-Tool (AO-Tool) was co-developed by the University of Passau, Shell International Exploration and Production B.V., and RISC Software GmbH.

An additional aim of the research underlying this thesis was to broaden the applicability of the data driven approach of the AVI and ABM algorithm to the area of seismic imaging. This topic is covered in Section 6.2. First we introduce the required background information. Then we briefly explain the main challenges related to seismic imaging and how those are traditionally met. Most of these traditional approaches have in common that relatively strong assumptions have to be made upfront, which may, however, not be satisfied in practice. Afterwards we explain how the extended ABM algorithm can be used in this context. Moreover, two examples using synthetic seismic datasets are discussed in more detail.

In Section 6.3 we talk about applications of the ABM algorithm to model unconventional geological structures via algebraic surfaces.

Finally, we illustrate in Section 6.4 how the Simultaneous Quasi-Diagonalisation Algorithm (31), which we have developed in Section 5.3, can be used to compute approximations of the zeros of a 0-dimensional radical ideal in a numerically stable way. Through a few example computations, we show that the Simultaneous Quasi-Diagonalisation Algorithm is numerically superior to LAPACK even if the input border basis is exact.

6.1 Revealing Polynomial Relations in Real Data

A useful application which was already discussed in [28] and [32] is the revelation of approximate polynomial relations in measured data. Given a set $\mathbb{X} = \{p_1, ..., p_s\} \subset \mathbb{R}^n$ of affine points - the measurements - the crude strategy of using either the AVI (21) or ABM (22) algorithm to obtain approximate polynomial relations among the coordinates of the points is the following:

1. Remove obviously incorrect data from the dataset $\mathbb{X}$, e.g. outliers, illegal values (e.g. NaN values), intervals of instrumentation failure, etc.
2. In case the input data are very noisy, apply appropriate filters (e.g. low-pass filters).
3. Scale the the dataset $\mathbb{X}$ such that the coordinates are in a "sensible" range and optionally apply application specific transformations. See Remarks 6.1.2 and 6.1.3 for a more detailed discussion.
4. Divide the dataset $\mathbb{X}$ into the parts $\mathbb{I}$ and $\mathbb{V}$. The first one, $\mathbb{I}$, is used for creating the models, the second one, $\mathbb{V}$, will be used to verify the fitting quality of the models and their predictive power.
5. Apply the AVI or ABM algorithm for a certain set of ε-values to the dataset $\mathbb{I}$. Please note that the values of ε that should be considered depend on the actual application. The noise level in the measurements and the amount of scaling which was applied are important factors to consider. However, information about the noise is not always available a priori, thus making several computations for different values of ε necessary.

6. Evaluate the quality of the results based on a mixture of criteria which are again driven by practical requirements of the application. These criteria may include but are not limited to the fit of the polynomials on the sets of points $\mathbb{I}$ and $\mathbb{V}$, the maximal degree of the polynomials, and the size of the support of the polynomials.

Remark 6.1.1. One advantage of using the ABM algorithm instead of the AVI algorithm in this context is the tighter error bound guaranteed by the algorithm (compare Algorithm 22 and Theorem 4.3.1).

Remark 6.1.2. For the correctness of the AVI algorithm it is important to scale the coordinates in such a way that they are in the interval $[-1, 1]$ (see Algorithm 21). For the correctness of the ABM algorithm there is no such requirement. However, there are practical limitations because we are working with a double floating point implementation of the algorithm. To avoid overflow, e.g. when computing the evaluations of the terms and when forming A^*A (see Algorithm 22), it is therefore also important to make sure that the coordinates are in a sensible range. The following rule of thumb can be used to determine such a range. Let d be the maximal degree of a polynomial relation that we expect to find. Let us by a slight abuse of notation interpret $\mathbb{X}$ as a matrix in $\mathrm{Mat}_{s,n}(\mathbb{C})$ and let $\|\mathbb{X}\|_{\max}$ (compare Definition 2.3.18) be the entry of $\mathbb{X}$ with largest absolute value. Because of how the ABM algorithm works internally a value of $\approx \|\mathbb{X}\|_{\max}^{2d}$ must be stored in a double floating point variable. The maximal value that a double floating point variable can hold is about $1.8 \cdot 10^{308}$. So $\mathbb{X}$ needs to be scaled in such a way that $\|\mathbb{X}\|_{\max}^{2d} \ll 1.8 \cdot 10^{308}$.

As pointed in Chapter 4 the result of the ABM family of algorithms is in general not invariant under scaling or translation of $\mathbb{X}$ (compare Definition 4.7.28). So another important role of scaling is to assign weights to the individual measurements in order to influence the computation. A sensible choice depends once again on the specific application and is in fact a non-trivial issue. As a rule of thumb it makes sense to norm measurements that have the same physical units in a similar way, additionally the maximal absolute values (excluding outliers) of the normed measurements should at most differ by a factor 10^4.

For some applications it may even make sense to apply more advanced transformation to the individual measurements. For instance it may be appropriate to apply the natural logarithm, the n-th root, etc. to some of the measurements because only after these transformations a low degree polynomial relation between the coordinates of the transformed $\mathbb{X}$ exists.

Remark 6.1.3. If a higher accuracy implementation of the ABM algorithm, for example, using the GNU MPFR library ([70]), would be used it would no longer be necessary to scale the input data in order to avoid overflow.

Remark 6.1.4. Note that not all polynomials returned by the algorithms can be interpreted as physical relations among the coordinates of the points. This can be attributed to the noise in the input data and to the fact that we compute an (approximate) border basis. In particular this means that also the "geometry" in which the measurements where obtained is captured in the polynomials, e.g. the sampling interval.

6.1.1 Finding Specific Relations

In the beginning of Section 6.1, we just sketched how the ABM and AVI algorithm can be used to find polynomial relations among different measurements if little to no information about the structure of the relations is available up front. However, sometimes additional information may be available or we may have specific requirements like expressing one measurement as a polynomial function in the other measurements (compare also Example 6.1.6).

Let us consider the case in which we want to test whether a certain measurement is redundant, notably because it can be expressed in terms of other measurements. Note that we are only able to find relations which are approximately polynomial or can be approximated reasonably well by polynomial expressions. Theoretically it would be possible to apply the standard ABM algorithm again but generally it will not be possible to solve directly for the indeterminate that is associated with the specific measurement that we are interested in. A more detailed discussion is contained in Section 4.4. However, using the extended ABM algorithm it is possible to search for such relations explicitly (up to a certain degree and residual error). Before we start with a high level description of the method, we explain the setup.

Given a set $\mathbb{X} = \{p_1, ..., p_s\} \subset \mathbb{R}^n$ of measurements taken at s points in time and another set of s affine points $\mathbb{M} = \{m_1, ..., m_s\} \subset \mathbb{R}$, we want to express $\mathbb{M}$ in terms of the measurements $\mathbb{X}$. For all $1 \leq i \leq s$ the points p_i and m_i were measured at the same time and thus belong together. We now describe the necessary steps:

1. Remove obviously incorrect data from the sets $\mathbb{X}$ and $\mathbb{M}$, e.g. outliers, illegal values (e.g. NaN values), etc. The datasets still need to be matched which means that, if a point is dropped from either dataset the corresponding point also needs to be removed from the other one.
2. In case the data sets $\mathbb{X}$ and $\mathbb{M}$ are very noisy, apply appropriate filters (e.g. low-pass filters).
3. Scale the the dataset $\mathbb{X}$ such that the coordinates are in a "sensible" range and optionally apply application specific transformations. See Remarks 6.1.2 and 6.1.3.
4. Divide the datasets $\mathbb{X}$ and $\mathbb{M}$ into a two parts $\mathbb{X}_1$,$\mathbb{X}_2$ and $\mathbb{M}_1$,$\mathbb{M}_2$. The first ones, $\mathbb{X}_1$ and $\mathbb{M}_1$, are used for creating the models. The second ones are used to verify their quality.
5. Apply the extended ABM algorithm to $\mathbb{X}_1$ and $\mathbb{M}_1$. The maximal allowed degree d and the initial values for ε and τ need to be determined in the context of the application (see Algorithm 24 for a detailed description of the meaning of the parameters). In practice it may be necessary to perform several computations for different values of d, ε and τ.
6. The set H returned by the extended ABM algorithm is either empty or contains candidate relations which can, for instance, be compared via their fit on the validation data or the size of their support and their degree.

Remark 6.1.5. The parameter τ in the extended ABM computation is a threshold number for the residual error of the least squares problems which are solved in line 14 of Algorithm 24. Polynomials are only accepted if the actual residual error is smaller than τ. If the relative

noise level κ in the data is roughly known it is thus possible to derive an initial estimate for τ by letting $\tau \approx \|\mathbb{M}_1\| \kappa$. However, information about the noise in the data may not always be available.

Another approach to derive an initial guess for τ can be pursued if it is known upfront that no meaningful (approximate) linear relations among the coordinates of the points exist. Then we know that the order ideal contains at least $x_1, ..., x_n$. We can thus let $A = \text{eval}_{\mathbb{X}_1}(x_1, ..., x_n)$ and choose τ such that $\tau \leq \left\| A^+ \mathbb{M}_1^{\text{tr}} - \mathbb{M}_1^{\text{tr}} \right\|$, where A^+ is the pseudoinverse of A. If more information is available about the structure of the order ideal, it also possible to derive better estimates for τ.

Example 6.1.6. [Production modelling of a well]

An important topic is to derive model equations for the oil and gas production of a well, because structural information about the internal workings of the well can be derived from it. Certainly it is possible to use the steps outlined above to derive such models. A different approach to modelling the oil or gas production $\mathbb{M}$, given some characteristic measurements $\mathbb{X}$ of one well, i.e. measurements that capture a significant amount of different physical states of the well, was also suggested in [28]. We compare both methods and detail the advantages of using the extended ABM algorithm.

The central idea behind the strategy suggested in [28] is that the order ideal $\mathcal{O}$ is treated as an "approximate basis" of P/I, so all other relevant relations can essentially be modelled using $\mathcal{O}$. The steps can be spelled out in the following way:

1. Prepare the input datasets $\mathbb{X}$ and $\mathbb{M}$ just like in steps 1-4 when applying the extended ABM algorithm in the beginning of this subsection.
2. Compute the sets $\mathcal{O}$ and G by applying the AVI or ABM algorithm to $\mathbb{X}_1$ for a suitable value of ε. In general, $\mathcal{O}$ is an order ideal and G an approximate $\mathcal{O}$-border basis. See Algorithm 22 and Theorem 4.3.1 for further details.
3. Compute the evaluation matrix A of $\mathcal{O}$ with respect to $\mathbb{X}_1$, i.e. $A = \text{eval}_{\mathbb{X}_1}(\mathcal{O})$.
4. Solve the least squares problem $\min_x \left\| Ax - \mathbb{M}_1^{\text{tr}} \right\|$.
5. Assess the quality of the result via the fit of the validation data $\left\| \text{eval}_{\mathbb{X}_2}(\mathcal{O}) - \mathbb{M}_2^{\text{tr}} \right\|$ and the condition number of the underlying least squares problem.

The major disadvantage of this method is that there is no direct control on the fitting. So, a smaller value of ε in the AVI/ABM computation may not necessarily lead to an improvement in the fitting of the production data. This behaviour is described in more detail in Subsection 4.4.1. Another problem is that the algorithm will always return a solution even if $\mathbb{X}$ and $\mathbb{M}$ are completely unrelated. This means that an additional validation step is always necessary to decide if the model is suitable.

When applying the extended ABM algorithm it is guaranteed that for a particular model $h_i \in H$, computed by the algorithm, the residual $\left\| \text{eval}_{\mathbb{X}_1}(\text{supp}(h_i)) - \mathbb{M}_1^{\text{tr}} \right\|$ will not exceed the given parameter τ. Furthermore, if H is empty we have the guarantee that no (approximate) polynomial model, such that we can express $\mathbb{M}_1$ approximately in terms of $\mathbb{X}_1$, exists.

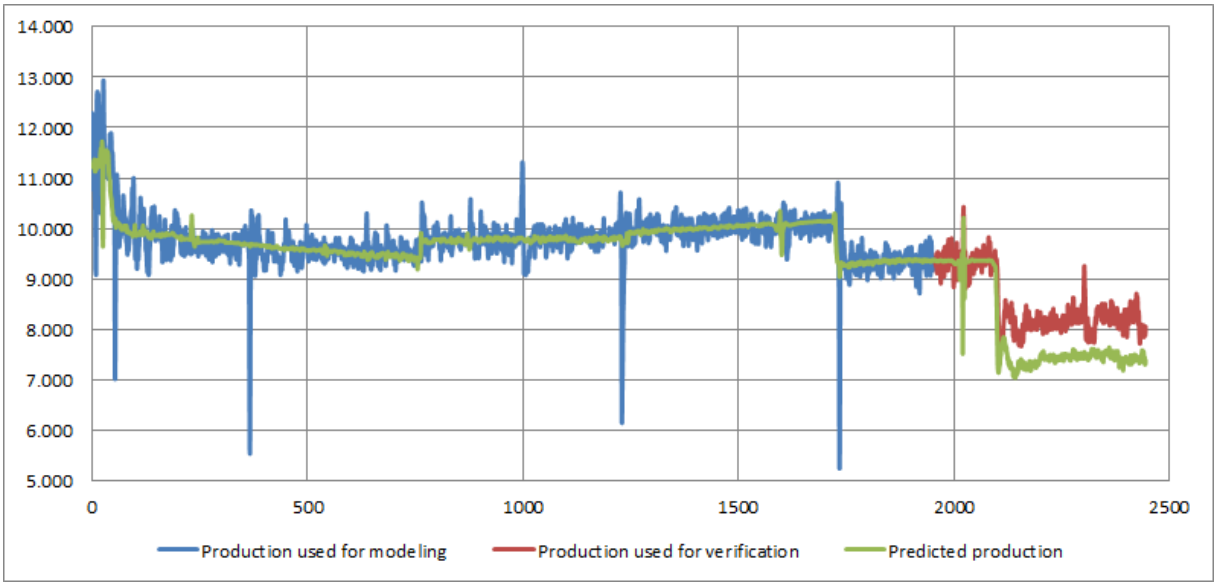

Figure 6.1: Production model obtained using the ABM algorithm followed by least squares fitting

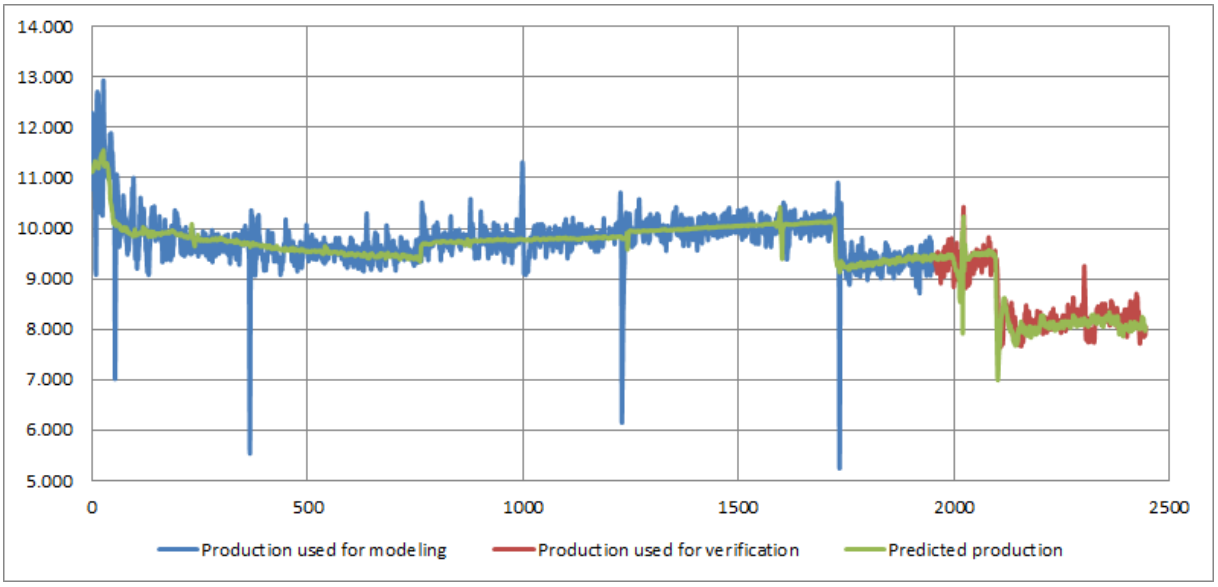

Figure 6.2: Production model obtained using the extended ABM algorithm

Nevertheless, satisfactory results can be obtained with this approach as can be seen in Figure 6.1. In this case 80% of the measurements were used to create the model. The remaining 20% of the data are also depicted. It can be seen that the model is still reasonably accurate in the validation part, though slightly off. The average error per point is about 0.352 in the modelling part of the data and about 0.677 in the verification part. Figure 6.2 depicts the model obtained using the extended ABM algorithm. Already by visual inspection it is possible to tell that the obtained model has a better fit in the validation part of the data. We compute that the average error per point is about 0.357 in the modelling part of the data and about 0.322 in the verification part.

6.2 Seismic Imaging

Seismic imaging is an important part of seismology. It is in general concerned with recovering structural information about the subsurface rock properties. The starting point is in most cases a set of so-called seismograms. These are created via an (artificial) acoustic source which emits sound waves into the underground. The rock behaves like an elastic medium. Thus the waves propagate in the subsurface according to the elastic wave equation. Whenever the properties of the subsurface change abruptly, the acoustic wave gets (partially) reflected at this interface. Therefore, a part of the wave will eventually reach the surface again where it is recorded via so-called geophones. The information about the source receiver distance and the travel time of an acoustic wave is exploited in the seismic imaging process to create an "image" of the subsurface, which depicts changes in density and/or velocity.

Accurate detection of rock types and characteristic geological formations is particularly important in the oil and mining industries, as probable locations of reservoirs can be determined in this way. The cost of performing a seismic survey is comparatively small in relation to the cost for test drilling or digging.

6.2.1 Basic Principles of Seismic Wave Propagation

One fundamental law underlying the propagation of sound waves is the wave equation. Here we present it for the one dimensional case. It is a second order linear partial-differential equation which relates time, space, and the amplitude of a wave. It is introduced here mainly since it plays an important role in conventional seismic imaging. Nevertheless, it is also important for our purposes because once we have found the boundary values, it can be used to simulate the propagation of a wave and thus its effect in the seismogram. With this knowledge the particular wave can be removed (approximately) from the seismogram, thus revealing more of its structure.

Definition 6.2.1. [Wave equation]

Let t be a time variable, let x be a spatial variable in $\mathbb{R}^n$, and let $u(x,t) : \mathbb{R}^{n+1} \to \mathbb{R}$ be a scalar function which encodes the amplitude of a wave. The second order differential equation

$$\frac{\partial^2 u(x,t)}{\partial^2 t} = c^2 \nabla^2 u(x,t)$$

where $c \in \mathbb{R}$ and where ∇ is the Laplace operator is called the **1D wave equation**.

Remark 6.2.2. Initial values or boundary values (see [72, page 8ff]) need to be specified in order to determine one wave.

Remark 6.2.3. In order to solve the wave equation numerically it is common practice to use finite difference methods to achieve both practical runtimes and good control over the expected error in the computed solution. See [72, Section 5.3] for a detailed discussion.

Another central physical law which we use is called Snell's law.

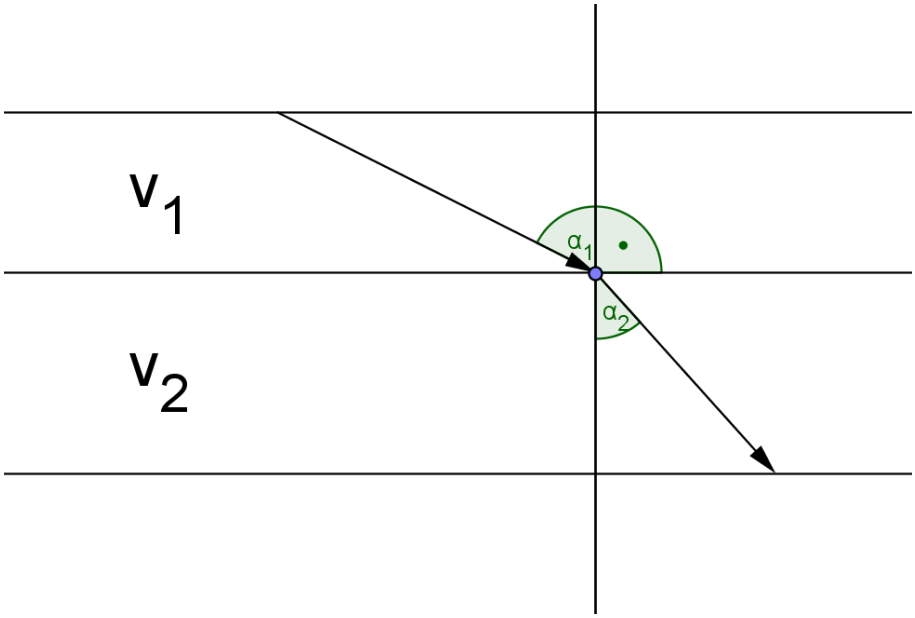

Figure 6.3: Relation between the angle of incidence and the angle of reflection

Definition 6.2.4. [Snell's law]

The following equation which establishes a relation between the angles of incidence α_1 and reflection α_2 and the velocities v_1 and v_2 at the interface of two isotropic media (compare Figure 6.3) is known as **Snell's law**:

$$\frac{\sin(\alpha_1)}{\sin(\alpha_2)} = \frac{v_1}{v_2}.$$

Remark 6.2.5. Snell's law for n layers can be stated in the following way

$$\frac{\sin(\alpha_1)}{v_1} = \ldots = \frac{\sin(\alpha_n)}{v_n}.$$

Further details and more theoretical background can, for example, be found in [71].

6.2.2 Established Methods

Now we briefly introduce two established classes of techniques for seismic imagining, together with their advantages and shortcomings. In fact there are a lot of algorithms available and still being developed as this area is a subject of active research. However, most of these methods can be classified into two categories.

Migration Techniques

Before we start describing the so-called migration techniques, we introduce the notion of zero offset data which plays an important role in some of them. We talk about **zero offset data** if the

source of the acoustic wave and the recording receiver are placed at the same location. Such data sets are of particular interest because they allow to easily relate depth and subsurface velocity via the equation $t = \frac{2h_1}{v_1}$ (compare Figure 6.7). If the subsurface geometry is very simple, it is even possible to read back the relevant information about the underground directly from the zero offset data without any further processing. Unfortunately though, it is not possible in practice to directly record such data because the signal generated by the seismic source would strongly interfere with the reflected waves recorded by the receiver. There exist methods like dip moveout correction (compare [71, Section 16.5] for details) which can transform a normal dataset into a reasonable approximation of a zero offset one.

Migration techniques are a family of algorithms which are usually applied directly to the seismogram after it has been transformed to zero offset in order to relocate seismic events to their true spatial locations. This is important because the apparent location of some seismic events in the original seismogram may be misleading. These techniques are particularly useful when we deal with complex geological structures such as faults, salt bodies, or fractures. In these cases it is no longer possible to extract accurate spatial locations from zero offset data. It is possible to distinguish between two different subcategories, namely time migration and depth migration.
Time migration does not require an initial velocity model and is computationally faster than depth migration. A disadvantage, though, is that it is implicitly assumed that the lateral changes in velocity are not significant. Unfortunately, this assumption only holds for rather simple geological configurations. Some well-known time migrations algorithms include Stolt migration and finite-difference migration (compare [74, Section 18.3]).
Depth migration requires the knowledge of an initial velocity model and is computationally more involved than pure time migration. If a good estimate of the subsurface velocity model is available this technique is reasonably accurate. The major disadvantage here is that it is not trivial to obtain a reasonably accurate subsurface velocity model without extensive a priori knowledge. Initial velocity models are usually refined iteratively. Nevertheless, if the initial estimate is too far off from the true velocity model, the final result of the imagining process can be quite inaccurate. Standard depth migration methods include Kirchhoff migration, Gaussian beam migration, and reverse time migration. A detailed description can be found in [74, Chapter 18].
So, a common feature of all migration techniques is that they are either based on rather strong assumptions about the subsurface or that they require a significant amount of prior knowledge to work properly.

Full Waveform Inversion Techniques

As the propagation of waves in the subsurface can be modelled with the help of the appropriate wave equation and the corresponding velocity (and density) model of the subsurface it is a natural idea to try to use the wave equation to forward model the wave in the subsurface. In this way a synthetic seismogram can be generated that is matched with the data that was actually recorded via the geophones. Iteratively the model of the subsurface velocity and density is updated in such a way that the misfit between the simulated synthetic seismogram and the actual seismogram is minimised. Various methods exist for the updating process of the velocity (and density)

models. A comprehensive overview is contained in the book [73]. This family of techniques has only recently drawn more attention as it is computationally very demanding. Even today it cannot be applied in its pure form to large 3D datasets, e.g. datasets containing several hundred gigabytes of data. A major challenge is that the solutions obtained in this way may be non-unique and additionally they may be ill-conditioned. In practice it is impossible to compute all solutions. Therefore, it is necessary to fall back to local numerical optimisation techniques which suffer from the common problems, specific to all non-global optimisation procedures, of getting stuck in local minima and their strong dependence on the starting values.

6.2.3 Recovery of the Velocity Field using the Extended ABM Algorithm

In the following we explain a novel approach to seismic imaging. It is based on the extended ABM algorithm (24), and uses no strong assumptions like the a priori knowledge of the velocity field or the subsurface geometry.
The idea is the following. Given a seismogram, the extended ABM algorithm is applied from top to bottom to individual wavefronts, i.e. we process the wavefronts chronologically with respect to their arrival times. From the approximate interpolation polynomials returned by the algorithm, geometric information can be extracted. It encodes the subsurface wave speed. The general structure of the algorithm can be formulated in the following way.

Algorithm 36: Ext-ABM Seismic Imaging

Input: A seismogram
Output: A subsurface velocity model

Repeat steps 1-7 in a top down fashion until either an inconsistency is found, or the bottom of the seismogram is reached.

1 Use contour tracking techniques to track individual wavefronts. Build a probabilistic model of possible continuations containing the different paths together with their likelihood. These can be used for backtracking if an inconsistency is found;
2 Pick the most likely path and apply the extended ABM algorithm to it;
3 Assess the numerical quality of the result by checking the fitting, the condition number of the underlying least squares problem, and the maximal degree;
4 Compare the resulting polynomials against known geometric structures (which is in the most simple case the branch of a hyperbola);
5 Extract information about subsurface wave speed by exploiting the geometric information encoded in the polynomials;
6 Update the subsurface model and the probabilistic models. In case an inconsistency is found use backtracking and pick the next candidate path;
7 Use forward modelling to simulate the wave propagation in the already known subsurface. Remove the seismic energy from these events from the seismogram;
return subsurface velocity model;

Remark 6.2.6. It is possible to search the seismogram (or smaller parts of it) for all known curves, which have been precomputed and can thus be interpreted, via Hough transforms. Un-

fortunately, this process is, at least in its basic form, quite costly, with respect to memory and computational resources and cannot be applied to larger seismograms. However, if it is possible to split the input dataset into reasonably small parts, Hough transformations are a viable option. For details about this technique and some improved variants like the Fast Hough Transform see [75, Section 10.3].

Path Tracking within the Seismic Data

An important step in the process proposed above is the tracking of individual seismic wavefronts. Classical image processing techniques like contour extraction and contour tracking can be used. However, in most practical situations, these techniques need to be augmented by additional information which we posses in the form of structural information (e.g. that the actual waves are smooth) and the probabilistic models which we build. This is necessary because of multiple reflections, noise, signal cancellation, and weak signals, to name just a few complications. In general, it is therefore difficult to find a continuous path in the data using methods which rely only on image processing techniques.

Advantages and Disadvantages

We briefly mention the advantages and disadvantages of the method that we have described.

Disadvantages:

- It is difficult to track one wave in complicated scenarios (e.g. multiple reflections, "ghosts", ...), thus tracking one wave may require advanced techniques like probabilistic models combined with backtracking.
- Requires pre-computation of different geological scenarios to allow matching of computed polynomials (curves/surfaces).

Advantages:

- Fast even on large datasets, because one wave trace is treated at a time.
- Constructs a subsurface model using only the input data. Does not make strong assumptions about the geometry of the subsurface.
- Can be combined with traditional methods. For example, the output can be used as the initial setup for other algorithms which perform local optimisation of the subsurface models.

6.2.4 Examples

Let us now assume that a 3D seismogram is available. This means that the measured amplitude of the acoustic signal is a function in the spatial coordinates x and y, such that $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x, y) = a_{xy}$. Additionally, we assume that the source and receiver positions S and R are known.

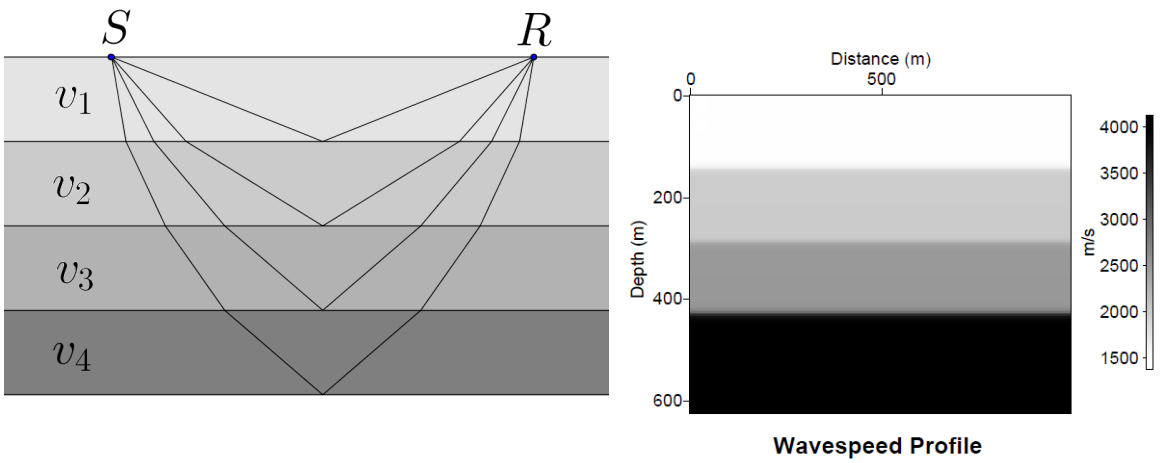

Figure 6.4: The layout and the velocity profile of four parallel layers

For the sake of simplicity consider the case of several parallel layers in the subsurface as depicted in Figure 6.4.

We will limit this example to a 2D seismic shot. This means that we let y or x be constant. Figure 6.5 shows a synthetic 2D seismogram, which we would obtain as the result of a seismic survey performed on media with the velocity profile given in Figure 6.4.

As the surface wave contains no information about the subsurface, we remove it using standard preprocessing techniques. Another option would be to simulate the surface wave and then to "subtract" the data from the shot record.

One Layer

Now we start by recovering the depth and velocity of the first subsurface layer.

In Figure 6.6 we have marked the first subsurface wave using standard contour tracking techniques. Note that only a small subset of the points would be needed as input for the algorithm, allowing for a practical margin of noise and uncertainty in the data.

We will now hint (compare Figure 6.7) how an actual equation can be derived in this simple case and how the result of the extended ABM algorithm can then be used to derive the desired information about the subsurface velocity. Denote the (surface) distance between source and receiver by d and the travel time between source and receiver by t. The first layer has a velocity of v_1 and a depth of h_1. Using elementary geometry we note that

$$t = \frac{2\sqrt{\left(\frac{d}{2}\right)^2 + h_1^2}}{v_1}.$$

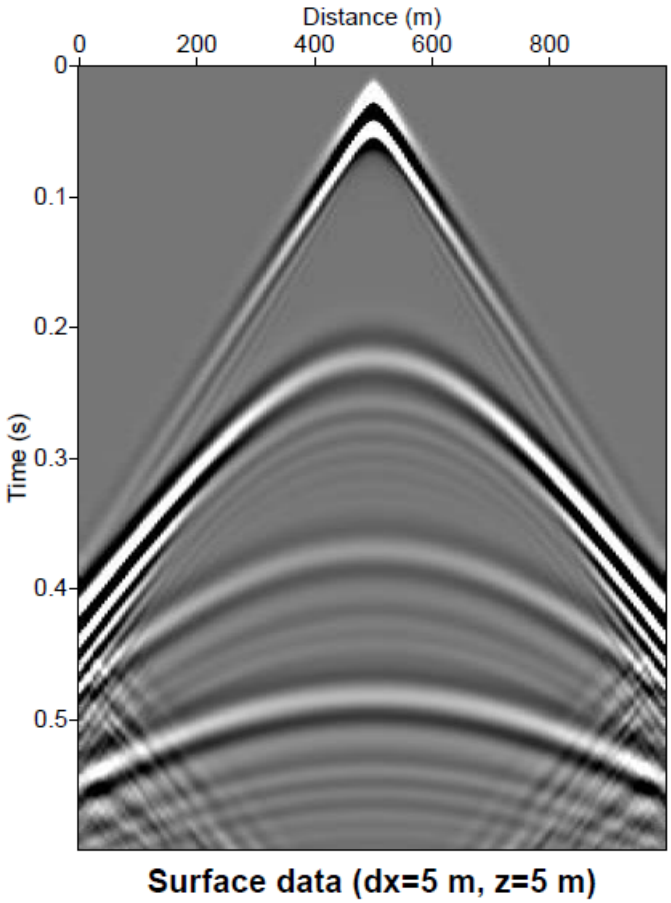

Figure 6.5: 2D Seismogram for four parallel layers

By squaring both sides of this equation we obtain the following hyperbolic relation:

$$t^2 = \frac{1}{v_1^2} d^2 + \frac{4h_1^2}{v_1^2}. \tag{6.1}$$

From this equation we can see that t^2 is a polynomial function in d. So it becomes apparent how we can apply the extended ABM algorithm. Let $\mathbb{M} = \left(t_1^2, ..., t_s^2\right)$, $\mathbb{X} = (d_1, ..., d_s)$, and $\varepsilon = 0.05$. In this constellation the set H returned by the extended ABM algorithm (see Algorithm 24 and Theorem 4.4.3) contains exactly one polynomial, namely

$$t^2 \approx 0.0000004788 d^2 + 0.0484.$$

Of course, in practice the geological constellation is not known upfront. So if the situation differs significantly from a parallel layer geometry, the polynomial returned by the algorithm will not have the shape of the right-hand side of equation 6.1. In this case it would have to be matched against a set of other known situations (e.g. dipping reflectors) and the one matching best would be picked. In our example we thus find a match with the hyperbolic equation (6.1). The parameters of interest can now be read off directly from the coefficients $v_1 \approx \sqrt{\frac{1}{0.0000004788}} \approx 1445\, m/s$ and $h_1 \approx \frac{1}{2}\sqrt{0.0484 * 1445^2} \approx 160 m$.

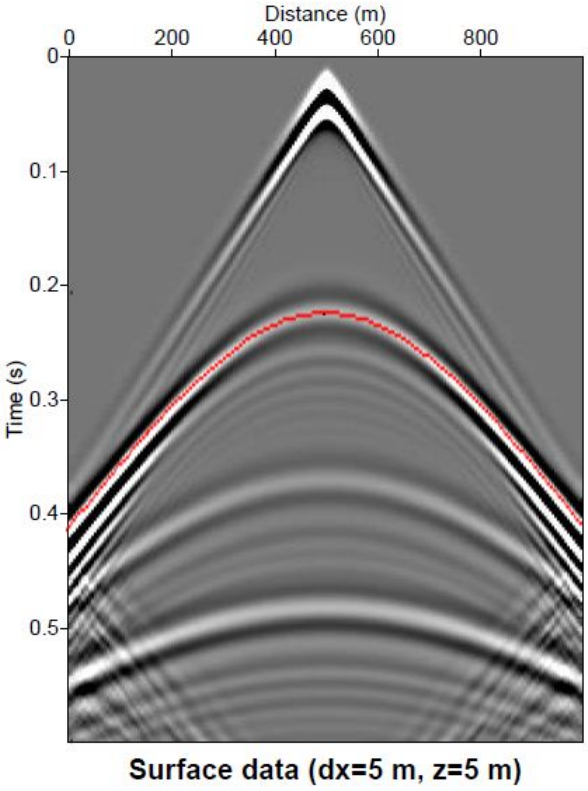

Figure 6.6: First subsurface wave marked in red

Using the information that we have obtained it is now possible to update the seismogram by forward propagating the wave along the interface which we have found and to remove its seismic energy from the seismogram. This counteracts wave cancellation and leads to a cleaner seismogram which can now be targeted again in the same fashion with the methods described above.

Two and more layers

Additionally, we will now explain how equations for the second layer, and with the same recipe for the n-th layer, can be derived and how these can be (approximately) solved with the help of the extended ABM algorithm.

The following image illustrates the situation for two layers. For convenience we use the naming scheme for our variables given in Figure 6.8.

Let us note that $\sum_{i=1}^{n} d_i = d$ and let us additionally assume that d_1, v_1 up to d_{n-1}, v_{n-1} have already been computed. By d we denote the horizontal distance between source S and receiver R.

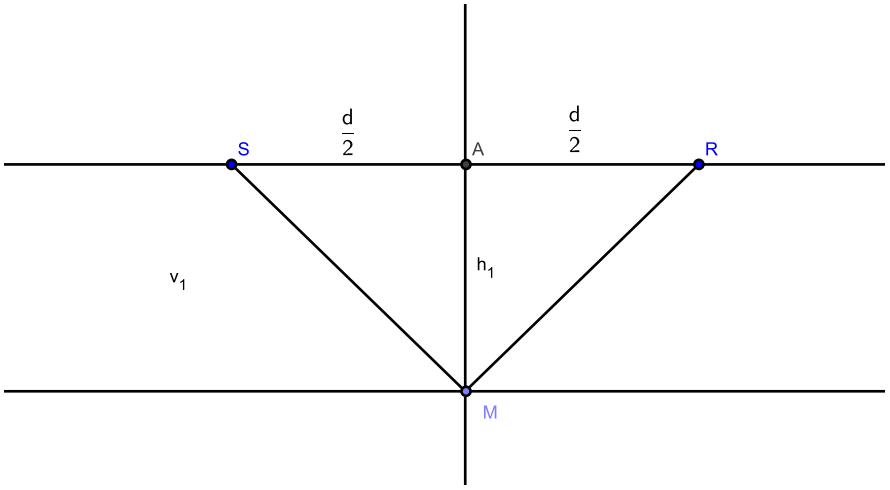

Figure 6.7: Subsurface geometry of one parallel flat layer

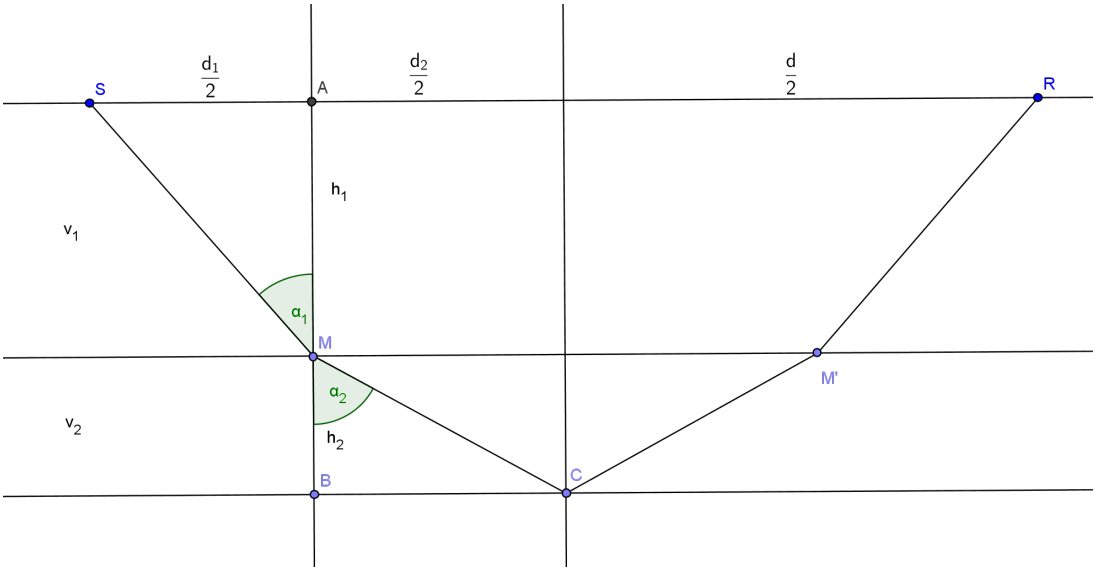

Figure 6.8: Subsurface geometry of two layers

Using basic geometry we derive that

$$t = 2\sum_{i=1}^{n} \frac{\sqrt{\frac{d_i^2}{4} + h_i^2}}{v_i}. \tag{6.2}$$

If we let $d = 0$, all d_i will be zero as well. In practice it is difficult to measure the acoustic signal at the point S. However, the assumption that we have data for $d = 0$ available is not crucial and only helps to simplify the following computations. So if we let $d = 0$, equation 6.2 becomes $t = 2\sum_{i=1}^{n} \frac{h_i}{v_i}$. Then it is possible to establish a direct relation between h_n and v_n, namely $t - 2\sum_{i=1}^{n-1} \frac{h_i}{v_i} = 2\frac{h_n}{v_n}$. Let us denote the ratio $2\frac{h_n}{v_n}$ by c_n.

We can now use the law of sines to establish that $\frac{\sin(\alpha_1)}{\frac{1}{2}d_1} = \frac{1}{\sqrt{d_1^2+h_1^2}}$ and $\frac{\sin(\alpha_n)}{\frac{1}{2}d_n} = \frac{1}{\sqrt{d_n^2+h_n^2}}$.

By combining this with Snell's law, we obtain the equation $\frac{d_1}{\sqrt{d_1^2+h_1^2}}\frac{\sqrt{d_n^2+h_n^2}}{d_n} = \frac{v_1}{v_n}$. Next we solve for d_n and arrive at $d_n = \frac{h_n v_n d_1}{\sqrt{v_1^2(d_1^2+h_1^2)-v_n^2 d_1^2}}$. So, the equation for the total distance is

$$d = \sum_{i=1}^{n-1} d_i + \frac{h_n v_n d_1}{\sqrt{v_1^2\left(d_1^2+h_1^2\right)-v_n^2 d_1^2}}. \tag{6.3}$$

If we put everything together we obtain

$$\begin{aligned} t &= 2\sum_{i=1}^{n-1}\frac{\sqrt{h_i^2+d_i^2}}{v_i} + 2\frac{v_1 d_n\sqrt{d_1^2+h_1^2}}{v_n^2 d_1} \\ &= 2\sum_{i=1}^{n-1}\frac{\sqrt{h_i^2+d_i^2}}{v_i} + 2\frac{v_1\sqrt{d_1^2+h_1^2}}{v_n^2 d_1}\frac{h_n v_n d_1}{\sqrt{v_1^2\left(d_1^2+h_1^2\right)-v_n^2 d_1^2}} \\ &= 2\sum_{i=1}^{n-1}\frac{\sqrt{h_i^2+d_i^2}}{v_i} + 2\frac{v_1\sqrt{d_1^2+h_1^2}}{\sqrt{v_1^2\left(d_1^2+h_1^2\right)-v_n^2 d_1^2}}\frac{h_n}{v_n} \\ &= 2\sum_{i=1}^{n-1}\frac{\sqrt{h_i^2+d_i^2}}{v_i} + c_n\frac{v_1\sqrt{d_1^2+h_1^2}}{\sqrt{v_1^2\left(d_1^2+h_1^2\right)-v_n^2 d_1^2}}. \end{aligned}$$

So in total

$$t = 2\sum_{i=1}^{n-1}\frac{\sqrt{h_i^2+d_i^2}}{v_i} + c_n\frac{v_1\sqrt{d_1^2+h_1^2}}{\sqrt{v_1^2\left(d_1^2+h_1^2\right)-v_n^2 d_1^2}}. \tag{6.4}$$

We have now derived two equations which relate the total travel time of the acoustic wave t with the source-receiver distance d. Let us assume that we used the extended ABM algorithm to model t in d. This allows us to get rid of noise and other unwanted effects which may initially obfuscate the structure of the wave.

Now we can choose d_1 arbitrarily within the range of measured values. This means that the only remaining unknown in the equations is v_n. Because of physical constraints, we can assume v_n to be bounded by a minimal velocity $v_{\min}$ and a maximal velocity $v_{\max}$. Starting with $v_n = v_{\min}$, we evaluate both formulas and determine whether the pair (d,t) belongs (approximately) to the acoustic wave in question. If this is not the case and as long as $v_n \leq v_{\max}$, we increase v_n by $\triangle v$ (e.g. $5^{m}/_{s}$). Note that no expensive mathematical operations are involved when we evaluate equations 6.3 and 6.4. It is thus possible to obtain an accurate value for v_n within a short period of time.

Now let us apply this technique to obtain the depth and velocity of the second layer. Using the extended ABM algorithm we obtain the polynomial approximation

$$t \approx 0.0000003650d^2 + 0.3796520026$$

for the relation between d and t. First we compute $c_2 \approx 0.379 - 2\frac{160}{1445} \approx 0.158$. Next we choose $d_1 = 50$, which is within the range of the data which we have measured. If we now iterate

over all "reasonable" values of v_2 we observe that for $v_2 \approx 2000^m/_s$ the equations evaluate to $d \approx 201.41m$ and $t \approx 0.387$. This is very close to $0.0000003650 \cdot 201.41^2 + 0.3796520026 \approx 0.394$. For this value of v_2 we obtain $h_2 \approx \frac{1}{2} 0.158^1/_s \cdot 2000^m/_s = 158m$.

6.3 Revealing Unconventional Geological Structures

Another field of application for the ABM algorithm is the modelling and approximation of complex geological structures in the subsurface. The aim is to approximate those structures via algebraic surfaces, which permit a very compact representation compared to e.g. representations via triangulated surfaces. An additional advantage is the availability of techniques, which make it possible to track the deformation of the algebraic surface over time. In the future, this could allow to relate the production strategy to actual physical changes in the reservoir and to predict its development over time. Some theoretical background on algebraic surfaces can be found in [76].

Definition 6.3.1. [Real Algebraic Surface]

Let $K = \mathbb{R}$, $P = \mathbb{R}[x, y, z]$, and $f \in P$ be a non-constant polynomial. Then the zero set of f

$$\left\{ (x, y, z) \in \mathbb{R}^3 \middle| f(x, y, z) = 0 \right\}$$

is called a **real algebraic surface**.

Example 6.3.2. [3D Unit Sphere]

The 3D unit sphere has the equation $x^2 + y^2 + z^2 = 1$. It is a real algebraic surface.

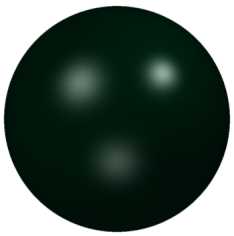

Figure 6.9: Visualisation of the zero set of $x^2 + y^2 + z^2 = 1$

6.3.1 Approximation of Geological Structures using the ABM Algorithm

Let us assume we are given points $\mathbb{X} = \{x_1, ..., x_s\} \subset \mathbb{R}^3$ which lie on the surface between two distinct geological structures in the underground. These points can, for instance, be obtained using level set methods on the subsurface density or velocity profiles. Then we can apply the ABM algorithm and analyse the set of polynomials which we obtain to check whether they can represent geological structures. Potentially the most simple polynomials are the ones which should be investigated first.

Example 6.3.3. The example was created in the following way. We started with a torus in $\mathbb{R}^3$ given by the equation $x^4 + 2x^2y^2 + y^4 + 2x^2z^2 + 2y^2z^2 + z^4 - 250x^2 - 250y^2 + 150z^2 + 5625$ and sampled randomly 400 points $\mathbb{X}$ on the surface of the torus. Additionally, the points were perturbed by Gaussian white noise. The aim now was to recover the geometric information in form of an algebraic equation.

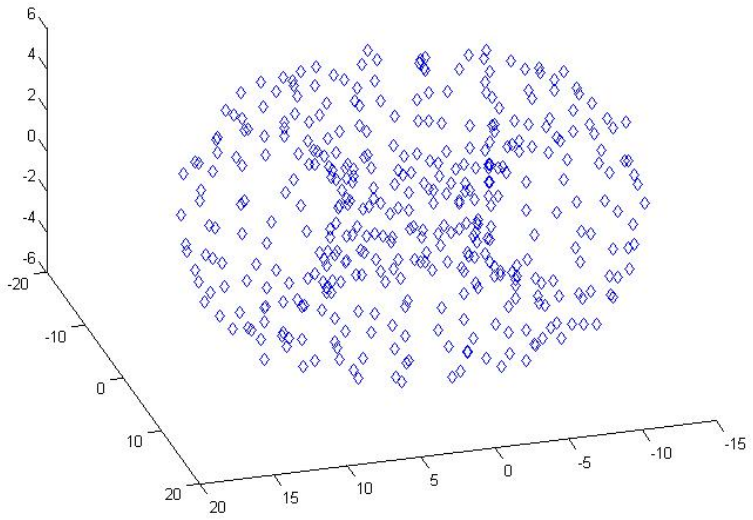

Figure 6.10: Perturbed point cloud in $\mathbb{R}^3$

We apply the ABM algorithm with $\varepsilon = 0.1$ and obtain the polynomial

$$g_1 \approx x^4 + 2x^2y^2 + y^4 + 2.3x^2z^2 + 2.2y^2z^2 - 276.8x^2 - 278.2y^2 + 10822.8$$

as the first element of G. Clearly, the similarity between the equation of the torus, which we used as input, and the recovered solution is visible. The visualisation of g_1 as an algebraic surface can be seen in Figure 6.11. It depicts a slightly deformed torus, which could be considered an unconventional geological structure.

Advantages and Disadvantages

Next, we briefly discuss the advantages and disadvantages of the method that we have described.

Disadvantages:

- Requires as input a detailed velocity/density model.
- May produce algebraic surfaces which have good fit, but which cannot depict real geological objects (e.g. singularities).

Advantages:

- Does not make any initial assumptions on the geometry of the subsurface.
- Compact description compared to triangulated surfaces.

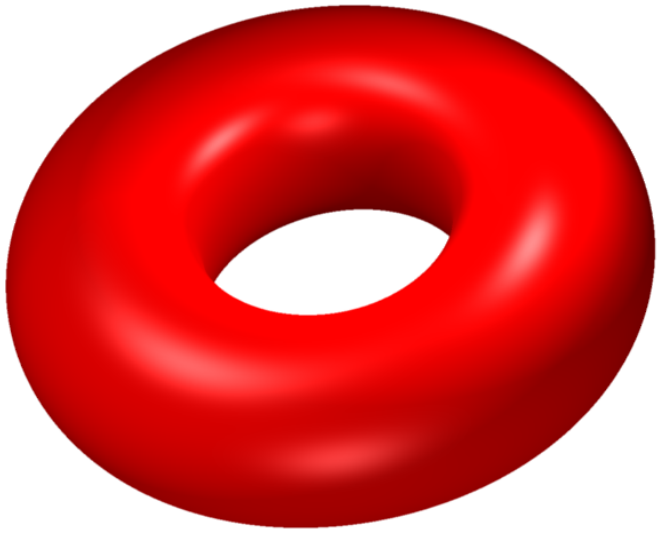

Figure 6.11: Picture of the recovered torus

- Robust against noise.
- Rich theory provided by algebraic geometry.
- Fast computation with the help of the ABM algorithm.

6.4 Stable Computation of Polynomial Roots

In this section, we will explain in more detail how the Simultaneous Quasi-Diagonalisation Algorithm (31) can be used to compute the roots of a zero-dimensional polynomial system in a stable way.

Let $P = \mathbb{C}[x_1, ..., x_n]$, let $\mathcal{O} = \{t_1, ..., t_\mu\}$ be an order ideal, and let $G = \{g_1, ..., g_\nu\}$ be an $\mathcal{O}$-border basis for a zero-dimensional radical ideal. If we skip line 23 in the eigenvector algorithm (30) and return the set $\mathbb{X} = \{p_1, ..., p_\mu\}$, then $\mathbb{X}$ is a numerical approximation of $\mathcal{Z}(\langle G \rangle)$ (compare Remark 5.1.8 and Theorem 5.1.9). The algorithm involves forming a random linear combination of the transposed multiplication matrices associated with G and afterwards computing the eigenvectors of this matrix. A similar approach is also advocated by Stetter in [49] on page 52. We will present numerical evidence that simultaneously diagonalising the multiplication matrices is more stable than the approach of Algorithm 30 even if G is an exact border basis.

Example 6.4.1. First, we briefly outline the setup which we use to evaluate the numerical accuracy of the Simultaneous Quasi-Diagonalisation Algorithm. Let $P = \mathbb{R}[x_1, x_2, x_3]$. We start with a finite set of s generic points $\mathbb{X} = \{p_1, ..., p_s\} \subset [0, 1]^3$ and obtain, with the help of Algorithm 18, an $\mathcal{O}$-border basis G for $\mathcal{I}(\mathbb{X})$. Using both Algorithm 30 and Algorithm 31 we compute $\mathcal{Z}(\langle G \rangle)$ and compare the Euclidean distance to the input data set $\mathbb{X}$. Please note that LAPACK version 3.4.2 was used for the computation of the eigenvectors in the eigenvector

algorithm. Twenty random linear combinations were picked and the error was averaged. Additionally, we used the implementation of Algorithm 18 in ApCoCoA-1.8 ([20]) to perform our benchmark computations. In Table 6.1 we can see that the SIMQDIAG algorithm returns about one additional correct digit after the comma. So even if we deal with exact border bases it is still beneficial to use Algorithm 31 if high numerical accuracy is desired.

s	35	40	45	50	55
Residual Algorithm 30	$1 \cdot 10^{-9}$	$1.4 \cdot 10^{-8}$	$3.5 \cdot 10^{-8}$	$3.9 \cdot 10^{-8}$	$4.4 \cdot 10^{-8}$
Residual SIMQDIAG	$< 1 \cdot 10^{-10}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-9}$	$1.8 \cdot 10^{-9}$	$2.0 \cdot 10^{-9}$

Table 6.1: Comparison of numerical accuracy for a generic set of points (exact border basis).

Remark 6.4.2. In case the roots have to be determined with even greater accuracy, it is possible to implement Algorithm 31 with higher precision floating point data-types like quad instead of double.

Example 6.4.3. Consider the same setup as in Example 6.4.1. However, the $\mathcal{O}$-border basis G was this time computed in double floating point and not in exact arithmetic. In Table 6.2 the accuracy of the eigenvector method is compared with the simultaneous diagonalisation algorithm. We observe that if G was only computed in finite precision arithmetic the advantage of Algorithm 31 becomes more evident. This is not too surprising as the Simultaneous Quasi-Diagonalisation Algorithm is specifically designed for this purpose.

s	35	40	45	50	55
Residual Algorithm 30	$3.7 \cdot 10^{-6}$	$5.2 \cdot 10^{-6}$	$9.7 \cdot 10^{-6}$	$1.1 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$
Residual SIMQDIAG	$1.2 \cdot 10^{-8}$	$4.8 \cdot 10^{-8}$	$1.8 \cdot 10^{-7}$	$4.3 \cdot 10^{-7}$	$5.1 \cdot 10^{-7}$

Table 6.2: Comparison of numerical accuracy for a generic set of points (approx. border basis).

Finally, we present an example where the roots of an $\mathcal{O}$-border basis G are very close to each other.

Example 6.4.4. Let $P = \mathbb{R}[x_1, x_2, x_3]$ and

$$\mathbb{X} = \{(0,0,0), (0,0,0.00000001), (1,1,1), (1,1,1.00000001), (1,1.00000001,1)\}.$$

We apply Algorithm 18 and afterwards we try to find back the roots of the computed border basis. In total we perform 200 runs and average the results. We observe that the eigenvector algorithm recovers the set $\mathbb{X}$ with a Euclidean error of about $2.8 \cdot 10^{-4}$ while Algorithm 31 achieves a relative accuracy of about $1.3 \cdot 10^{-9}$. In fact the result delivered by the eigenvalue algorithm is not satisfactory as the achieved accuracy is not sufficient to distinguish the clustered input points. Clearly the underlying eigenvalue problem becomes more ill-conditioned the closer the distance between the points in $\mathbb{X}$ gets (compare Corollary 2.9.7). Furthermore, this result showcases that working with the multiplication matrices simultaneously delivers superior accuracy compared to working with one random linear combination alone. Please note that the input border basis is exact and the inaccuracy is only introduced in the computation of the roots.

7 Conclusions and Outlook

In this thesis we have studied in detail the computation of approximate border bases. For this purpose, we have introduced several new algorithms in Chapter 4, most prominently the ABM algorithm, the Extended ABM algorithm and their variants. Compared to the state of the art AVI algorithm, these improve both on the runtime as well as particular properties like the fitting quality of the polynomials in the approximate border basis. The numerical properties of the algorithms were analysed in detail as well as their worst case runtime. Additionally, we managed to establish upper bounds for the departure from an exact border basis of the computed approximate border basis. These results were put into perspective in Section 4.7, where we compared the performance and properties of our algorithms with other state of the art approaches. For instance, we managed to outperform the SOI and NBM significantly while we obtained comparable results (see Subsections 4.7.2 and 4.7.3).
In addition to the computation of approximate border basis, we have investigated the so-called rational recovery problem in Chapter 5 where we try to find a close by exact border basis to a given approximate one. As a new contribution, we have shown how the problem can be tackled with the help of simultaneous quasi-diagonalisation of the involved multiplication matrices. In order to compute this quasi-diagonalisation, we have suggested a new variant of the Jacobi algorithm which handles complex valued input data and several matrices in parallel. For this algorithm we have proved convergence with respect to the involved transformations. Furthermore, we have compared the performance of our simultaneous quasi-diagonalisation algorithm with other algorithms in this area. In this way we have collected numerical evidence that our algorithm converges in less iterations than the other investigated methods. Additionally in the examples that we investigated, the common departure from diagonality was smaller if we used our algorithm.

Finally, in Chapter 6, we have described several current and potential future applications of the ABM and the extended ABM algorithm in the context of the (oil) industry. For example, the extended ABM algorithm has been applied in Subsection 6.1.1 to model the oil or gas production of a well directly, which requires additional steps when using either the AVI or ABM algorithm. Our new method gives more control about the error in the modelling process. Additionally, we have suggested a novel approach to seismic imaging using the extended ABM algorithm that requires less a priori assumptions than traditional approaches. However, the framework around the algorithm still needs to be further developed in order to achieve competitive performance with existing techniques on real world data.

The ABM algorithm has also found additional applications, which are proprietary to Shell, namely in the context of the production allocation problem. The task at hand is to determine for a multi zone well the contributions to the total production of the individual zones and the interactions of the zones when producing together. The corresponding algorithms and the necessary framework has been implemented by the author and is part of the Algebraic Oil Tool, which is currently undergoing practical testing within Shell.

Even though the algorithms discussed in this thesis overcome many drawbacks of the AVI algorithm, it is still possible to continue research in this direction as greater control than currently available over the departure from an exact border basis would be desirable. A promising direction might be to pay attention to the change in the singular values of the evaluation matrices from one computation step to another one in the ABM algorithm. We will briefly sketch the underlying idea. For this purpose, let us first recall the common setting of Chapter 4. Let $\mathbb{X} = \{p_1, ..., p_s\} \subset \mathbb{C}^n$ be a finite set of affine points, let $P = \mathbb{C}[x_1, ..., x_n]$, and let $\varepsilon \geq 0$. In Remark 4.3.3, we have explained why an upper bound for the departure from an exact border basis for the result of the ABM algorithm does not depend on the parameter ε. Recall that in the ABM algorithm we add a new term t_i to the set $\mathcal{O}$ if its evaluation with respect to $\mathbb{X}$ is "sufficiently" linearly independent of the evaluations of the elements which are already in $\mathcal{O}$. Now let $A = \text{eval}_{\mathbb{X}}(\mathcal{O})$ and let $\tilde{A} = (\text{eval}_{\mathbb{X}}(t_i), A)$. In case the matrix A has a smallest singular value σ which is slightly greater than ε and the matrix $\tilde{A}$ has a smallest singular value $\tilde{\sigma}$ which is equal to ε, then the matrix $\tilde{A}$ has an ε-approximate kernel and we add a new polynomial g_k to G which contains t_i as a border term. Note that the smaller $|\sigma - \tilde{\sigma}|$ is, the smaller is also the leading coefficient of g_k. This means that a minor reduction of the smallest singular value signifies that $\text{eval}_{\mathbb{X}}(t_i)$ is almost completely linearly independent of $\text{eval}_{\mathbb{X}}(\mathcal{O})$. Therefore, t_i should in fact be added to $\mathcal{O}$. We could address this problem by introducing an additional parameter $\kappa > 0$ that is checked and enforces that $|\sigma - \tilde{\sigma}| > \kappa$. In this way we could establish tighter bounds on the border coefficients of the polynomials in G which would also improve the upper bound for the departure of the computed approximate border basis from an exact border basis. The details of such an algorithm still have to be worked out and a careful analysis of its properties is necessary.

Another path that could be pursued would be to integrate the modified QR decomposition proposed by Sauer in [41] (compare also Section 4.7) into the ABM algorithm. In this way, the ∞-norm instead of the Euclidean norm of the residual error would be minimised. Again the properties of the resulting algorithm would need a thorough analysis.

A new direction which is currently being explored, but which is out of the scope of this thesis, is the development of a new variant of the extended ABM algorithm which can be used to address sparse approximation problems and problems in compressive sensing. The underlying assumption is that the signal that we want to approximate is a superposition of only a few basis signals. Once a proper basis has been chosen, it is possible to obtain a sparse representation with respect to these functions in a greedy way. This development is particularly interesting because it could help to reduce the cost to perform a seismic survey as less geophones would be required to reconstruct the acoustic signals.

With respect to the rational recovery problem we note that the methods that we have described

in Chapter 5 try to construct a suitable set of points which are used as input for the Buchberger-Möller algorithm in order to construct an exact border basis. However, another interesting direction would be to further investigate direct methods, like the one proposed in [44], which works directly on the (approximate) multiplication matrices and therefore provides enhanced numerical stability.

A future direction of research in the direction of simultaneous quasi diagonalisation could be to extend state of the art algorithms like the QR algorithm or the Divide and Conquer algorithm to the case of several matrices and to compare their performance and convergence properties with the algorithm discussed in this thesis.

Acknowledgements

First of all I would like to thank Professor Dr. Martin Kreuzer, my supervisor, who kindly offered me a position as a research assistant in the Algebraic Oil project and who gave me the opportunity to do a doctorate in applied mathematics. His continuous support during the writing of this thesis was very much appreciated. Without his input and the fruitful discussions this work would not have been possible in its current form.

Next, I would like to thank my second assessor Professor Dr. Tomas Sauer. He was very helpful with respect to the numerical aspects of this work. Additionally, he made me familiar with concepts from H-basis theory. I am also grateful to his student Johannes Czekansky, who provided me with example computations of the approximate H-basis algorithm.

Furthermore, I am very grateful to Dr. Henk Poulisse, who invented the Algebraic Oil project. His deep insights as well as his intuition were always of great value and a driving force behind all the developments in the project. In our many discussions he always managed to keep me motivated. His very detailed remarks and his suggestions on this thesis were invaluable.

Special thanks also go to Dr. Corina Baciu, the project leader in Risjwijk (NL), and to Stefan Kaspar, a former colleague, who were always helpful with proof reading and very useful comments.

Additionally, I would like to thank all of my colleagues but particularly Stefan Schuster, Dr. Georg Maier, Markus Kriegel, Kai Beermann, Christian Kell, Johannes Nagler, and Bilge Sipal for their support, and extensive proof reading. I am also indebted to Andreas Schindler who shared his LaTeX template files with me.

While working at the University of Passau my position was funded by Shell International Exploration and Production B.V., Rijswijk, the Netherlands.

8 Appendix

Contents

8.1 Overview of Functions Which Were Implemented in ApCoCoA

The ABM algorithm

The ABM algorithm (22) computes an approximate vanishing ideal for a given set of points $\mathbb{X} \subset \mathbb{Q}^n$ with respect to a threshold number ε. The algorithm is similar to AVI but it has the following differences. It works term by term versus degree by degree while constructing the polynomials in the approximate border basis. Additionally, ABM also does not rely on the direct computation of the SVD but uses eigenvectors for the fitting of the polynomials. The scaling of the input data is no longer a necessary prerequisite as in the AVI algorithm but influences the properties of the resulting approximate border basis G and the corresponding order ideal $\mathcal{O}$.

Properties

If run in strict border basis mode (which is the default) the set G will be an approximate border basis and most polynomials will be ε-approximately vanishing. The reason why some polynomials may have worse evaluations is related to the fact that they have to be accepted because otherwise $\mathcal{O}$ would not be a proper order ideal.

If run in non border basis mode all polynomials will be ε-approximately vanishing but $\mathcal{O}$ may contain gaps and is thus no order ideal.

Depending on whether one is only interested in the border basis G or also additionally in the order ideal $\mathcal{O}$ one can activate or deactivate if $\mathcal{O}$ is returned. The parameter is not publicly exposed but can be set via the numabm.cpkg package.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.ABM(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

The last three parameters are optional.

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.ABM(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

Points has to be a CoCoA matrix containing $\mathbb{X}$ and **Epsilon** ε is a rational number. These parameters always have to be supplied.

Delta is used for truncating small coefficients of polynomials, if not supplied the default value of 0.00000000001 will be used.

ForbiddenTerms is a list which contains the terms which are not allowed to show up in the order ideal.

NormalizeType is an integer in the range [1..4]. It determines if and how the points in $\mathbb{X}$ will be normalised. The default value is 2. If **NormalizeType** equals 1, each coordinate is divided by the maximal absolute value of the corresponding column of the matrix. This ensures that all coordinates of the points are in [-1,1]. With **NormalizeType**=2 no normalisation is done at all. **NormalizeType**=3 shifts each coordinate to [-1,1]. So its minimum is mapped to -1 and the maximum to one, describing a unique affine mapping. The last option is **NormalizeType**=4. In this case, each coordinate is normalised using the column's Euclidean norm.

By default the algorithm returns a list which contains G and $\mathcal{O}$.

Non public options in the numabm.cpkg package

The whole range of options for the algorithm is not exposed publicly but may be set via the numabm.cpkg package.

The line that needs to be changed is

OptionList := [0,0,0,0,0,1,0,1,1000];

The first option is either 1 (TRUE) or 0 (FALSE) and controls the isAVI check. In the server window additional log output is generated. The result of the algorithm is not changed.

The second option is either 1 (TRUE) or 0 (FALSE) and controls the isAppBB check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The third option is either 1 (TRUE) or 0 (FALSE) and controls the remainingLinearRealtionsInOI check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The forth option is either 1 (TRUE) or 0 (FALSE) and controls if re-projection of polynomials is activated. If activated all polynomials in which terms were left out, because of either too small coefficients or a too small contribution to a polynomial, will be reprojected.

The fifth option is either 1 (TRUE) or 0 (FALSE) and controls the switch for the rational recovery. This option is reserved for future use and does not affect the result at the moment.

The sixth option is either 1 (TRUE) or 0 (FALSE) and controls the strict border basis mode. If activated (default) always an approximate border basis will be returned, but some polynomials may not vanish ε-approximately. If deactivated all polynomials will be guaranteed to vanish ε-approximately but may not form an approximate border basis.

The seventh option is either 1 (TRUE) or 0 (FALSE) and controls if the border basis variant of ABM is supposed to be used. Its value needs to be set to false.

The eigth option is either 1 (TRUE) or 0 (FALSE) and controls which kind of truncation mode for small terms/coefficients is used. If disabled (default) terms which have a coefficient less than **Delta** will be discarded. If enabled terms which have an average contribution less than **Delta** will be discarded. If option 6 is activated the changed polynomials will be reprojected to give the best possible fit with respect to the new terms.

The ninth option is a positive integer and sets the maximum degree after which the computation will be terminated. This can be used to stop a computation that would otherwise take too long. Note that only a truncated result will be returned which may not represent a full approximate border basis.

Example

The following CoCoA example explains the basic steps how ABM can be used in production modelling:

```
X := Mat([[...]]); -- measurements
P := Mat([[...]]); -- measured production data
Epsilon := 0.1; -- threshold number
AppBB := Num.ABM(X, Epsilon); -- apply the ABM algorithm
ProdPoly := $apcocoa/numerical.ProjectAVI(X, P, AppBB[2]);
-- find a production polynomial with respect to the OI
-- contained in AppBB[2]
```

The extended ABM algorithm

The extended ABM algorithm is, as the name suggests, an extension of the ABM algorithm. While the latter solves the homogeneous least squares problem to construct relations among the coordinates of the points in $\mathbb{X}$, the first one delivers additionally polynomials which have when evaluated at the set of points $\mathbb{X}$ approximately the values of an additional set of 1D points $\mathbb{V}$. The evaluation error can be explicitly specified by the parameter τ. It is thus well-suited for approximately interpolating the dataset $\mathbb{V}$.

Note that the current implementation of the extended ABM algorithm in ApCoCoA only returns the sets $\mathcal{O}$ and H but not G.

For the actual computation of the polynomials in H the QR decomposition is used. Higher numerical stability could be achieved by switching to QR with column pivoting or the SVD, but the speed of the computation would decrease accordingly.

Properties

For a given set of points $\mathbb{X}$, an additional set of 1D points $\mathbb{V}$, and threshold numbers ε and τ the algorithm returns two sets H and $\mathcal{O}$, where H contains polynomials h_i such that $\left\|\text{eval}_{\mathbb{X}}(h_i) - \mathbb{V}^{\text{tr}}\right\| < \tau$. The underlying $\mathcal{O}$-border basis G is currently not returned but can be obtained relatively easy with the help of the set $\mathcal{O}$. If run in strict border basis mode some polynomials in G may have a larger evaluation than ε, otherwise all polynomials in G have smaller evaluations than ε but $\mathcal{O}$ may not be a proper order ideal.

Be aware that if τ is not chosen properly (i.e. very small) the set H returned by the algorithm may be empty. As the error τ is a priori not always known it is possible to terminate the calculation after a certain degree is reached, in order to avoid long processing times. The maximum degree can be set in the numextabm.cpkg package.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.EXTABM(Points:MAT, Val:MAT, Epsilon:RAT, Tau:RAT, ForbiddenTerms:LIST, NormalizeType:INT):Object

The last two parameters are optional.

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.EXTABM(Points:MAT, Val:MAT, Epsilon:RAT, Tau:RAT, ForbiddenTerms:LIST, NormalizeType:INT):Object

Points has to be a CoCoA matrix containing $\mathbb{X}$, **Val** has to be a CoCoA matrix containing $\mathbb{V}$, **Epsilon** is a rational number that has the same meaning as in the ABM algorithm, and **Tau** is a rational number that specifies when a polynomial will be accepted into the set H. These four parameters always have to be supplied. **Points** and **Val** need to have the same number of rows.

ForbiddenTerms is a list which contains the terms which are not allowed to show up in the order ideal.

NormalizeType is an integer in the range [1..4]. It determines if and how the points in $\mathbb{X}$ will be normalised. The default value is 2. If **NormalizeType** equals 1, each coordinate is divided by the maximal absolute value of the corresponding column of the matrix. This ensures that all coordinates of the points are in [-1,1]. With **NormalizeType**=2 no normalisation is done at all. **NormalizeType**=3 shifts each coordinate to [-1,1]. So its minimum is mapped to -1 and the maximum to one, describing a unique affine mapping. The last option is **NormalizeType**=4. In this case, each coordinate is normalised using the column's Euclidean norm.

The algorithm returns a list which contains H and $\mathcal{O}$.

Non public options in the numextabm.cpkg package

The whole range of options for the algorithm is not exposed publicly but may be set via the numextabm.cpkg package.

The line that needs to be changed is

OptionList := [0,0,0,0,0,1,0,1,6];

The first option is either 1 (TRUE) or 0 (FALSE) and controls the isAVI check. In the server window additional log output is generated. The result of the algorithm is not changed. At the moment this option has not been implemented for the ext ABM and has no effect.

The second option is either 1 (TRUE) or 0 (FALSE) and controls the isAppBB check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The third option is either 1 (TRUE) or 0 (FALSE) and controls the remainingLinearRealtionsInOI check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed. At the moment this option has not been implemented for the ext ABM and has no effect.

The fourth option is either 1 (TRUE) or 0 (FALSE) and controls if re-projection of polynomials is activated. If activated all polynomials in which terms were left out, because of either too small coefficients or a too small contribution to a polynomial, will be reprojected.

The fifth option is either 1 (TRUE) or 0 (FALSE) and controls the switch for the rational recovery. This option is reserved for future use and does not affect the result at the moment.

The sixth option is either 1 (TRUE) or 0 (FALSE) and controls the strict border basis mode. If activated (default) always an approximate border basis will be returned, but some polynomials

may not vanish ε-approximately. If deactivated all polynomials will be guaranteed to vanish ε-approximately but may not form an approximate border basis.

The seventh option is either 1 (TRUE) or 0 (FALSE) and controls if the border basis variant of ABM is supposed to be used. Its value needs to be set to false.

The eighth option is either 1 (TRUE) or 0 (FALSE) and controls which kind of truncation mode for small terms/coefficients is used. If disabled (default) terms which have a coefficient less than **Delta** will be discarded. If enabled terms which have an average contribution less than **Delta** will be discarded. If option 6 is activated the changed polynomials will be reprojected to give the best possible fit with respect to the new terms.

The ninth option is a positive integer and sets the maximum degree after which the computation will be terminated. This can be used to stop a computation that would otherwise take too long. Note that only a truncated result will be returned which may not represent a full approximate border basis.

Example

The following CoCoA example explains the basic steps how the extended ABM can be used in direct production modelling:

```
X := Mat([[...]]); -- measurements
P := Mat([[...]]); -- measured production data
Tau := DetermineResidualErrorForLinearOI();
-- use linear OI as a starting point for the error estimates
AppBB := Num.EXTABM(X, P, 0.01, Tau); -- apply the ABM algorithm
-- AppBB[1] contains a set of possible production polynomials
-- pick the "best" polynomial with respect to the production
-- which was not used for modelling
```

The next CoCoA example illustrates how the extended ABM can be used for modelling wavefronts in seismic imaging:

```
X := Mat([[...]]); -- 3D spatial uniform coordinates
T := Mat([[...]]); -- arrival time of wavefront
Tau := 0.2; -- depending on the noise in the measurements
RelEqu := Num.EXTABM(X, T, 0.01, Tau);
-- RelEqu[1] contains relations which express the arrival time
-- in terms of spatial coordinates
```

The BB ABM algorithm

The border basis ABM algorithm (25) is a variant of the ABM algorithm which enforces that the leading coefficients of the border terms are exactly one already during the computation. In the ABM algorithm this border shape is achieved through division by the leading coefficient.

The polynomials in G returned by the algorithm have without normalisation an evaluation at $\mathbb{X}$ which is less than ε. Note that in the ABM and extended ABM algorithm the polynomials have to be normalised to have coefficient vector norm one, so that this property holds.

Properties

The algorithm uses as input a set of points $\mathbb{X}$ and a threshold number ε. If run in strict border basis mode the algorithm will return a set of polynomials G which form an approximate border basis with respect to the order ideal $\mathcal{O}$. Most polynomials will have an evaluation which is less than ε at $\mathbb{X}$. If the algorithm is run in the non strict border basis mode all polynomials will have an evaluation which is less than ε at the points $\mathbb{X}$ but $\mathcal{O}$ may not be a proper order ideal.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.BBABM(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

The last two parameters are optional.

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.BBABM(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

Points has to be a CoCoA matrix containing $\mathbb{X}$ and **Epsilon** ε is a rational number. These parameters always have to be supplied.

Delta is used for truncating small coefficients of polynomials, if not supplied the default value of 0.00000000001 will be used.

ForbiddenTerms is a list which contains the terms which are not allowed to show up in the order ideal.

NormalizeType is an integer in the range [1..4]. It determines if and how the points in $\mathbb{X}$ will be normalised. The default value is 2. If **NormalizeType** equals 1, each coordinate is divided by the maximal absolute value of the corresponding column of the matrix. This ensures that all coordinates of the points are in [-1,1]. With **NormalizeType**=2 no normalisation is done at all. **NormalizeType**=3 shifts each coordinate to [-1,1]. So its minimum is mapped to -1 and the maximum to one, describing a unique affine mapping. The last option is **NormalizeType**=4. In this case, each coordinate is normalised using the column's Euclidean norm.

The algorithm returns a list which contains G and $\mathcal{O}$.

Non public options in the numabm.cpkg package

The CoCoAL interface to the BB ABM algorithm is also implemented in the same package as the ABM algorithm.

The whole range of options for the algorithm is not exposed publicly but may be set via the numabm.cpkg package.

The line that needs to be changed is

OptionList := [0,0,0,0,0,1,1,0,1000];

The first option is either 1 (TRUE) or 0 (FALSE) and controls the isAVI check. In the server window additional log output is generated. The result of the algorithm is not changed.

The second option is either 1 (TRUE) or 0 (FALSE) and controls the isAppBB check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The third option is either 1 (TRUE) or 0 (FALSE) and controls the remainingLinearRealtionsInOI check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The fourth option is either 1 (TRUE) or 0 (FALSE) and controls if re-projection of polynomials is activated. If activated all polynomials in which terms were left out, because of either too small coefficients or a too small contribution to a polynomial, will be reprojected.

The fifth option is either 1 (TRUE) or 0 (FALSE) and controls the switch for the rational recovery. This option is reserved for future use and does not affect the result at the moment.

The sixth option is either 1 (TRUE) or 0 (FALSE) and controls the strict border basis mode. If activated (default) always an approximate border basis will be returned, but some polynomials may not vanish ε-approximately. If deactivated all polynomials will be guaranteed to vanish ε-approximately but may not form an approximate border basis.

The seventh option is either 1 (TRUE) or 0 (FALSE) and controls if the border basis variant of ABM is supposed to be used. Its value is true in the BB ABM algorithm.

The eighth option is either 1 (TRUE) or 0 (FALSE) and controls which kind of truncation mode for small terms/coefficients is used. If disabled (default) terms which have a coefficient less than **Delta** will be discarded. If enabled terms which have an average contribution less than **Delta** will be discarded. If option 6 is activated the changed polynomials will be reprojected to give the best possible fit with respect to the new terms.

The ninth option is a positive integer and sets the maximum degree after which the computation will be terminated. This can be used to stop a computation that would otherwise take too long. Note that only a truncated result will be returned which may not represent a full approximate border basis.

Example

The following CoCoA example explains the basic steps how the BB ABM can be used:

```
X := Mat([[...]]); -- measurements
Epsilon := 0.1; -- an error estimate
AppBB := Num.BBABM(X, Epsilon); -- apply the ABM algorithm
-- AppBB[1] contains an approximate border basis and
-- AppBB[2] the corresponding order ideal
```

The complex ABM algorithm

The complex ABM algorithm is essentially a complex version of the ABM algorithm (22). Now the input dataset $\mathbb{X}$ may contain complex points and the real ABM algorithm becomes a special case of the complex ABM algorithm, as both algorithms produce the same output if only real points are contained in $\mathbb{X}$.

Properties

Please refer to the properties of the ABM algorithm (8.1) for details.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.CABM(Real:MAT, Complex:MAT, Epsilon:RAT, Delta:RAT,
NormalizeType:INT):Object

The last two parameters are optional.

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.CABM(Real:MAT, Complex:MAT, Epsilon:RAT, Delta:RAT,
NormalizeType:INT):Object

Real has to be a CoCoA matrix containing the real part of $\mathbb{X}$, **Complex** has to be a CoCoA matrix containing the complex part of $\mathbb{X}$ and **Epsilon** ε is a rational number. These parameters always have to be supplied. **Real** and **Complex** need to have the same dimensions.

Delta is used for truncating small coefficients of polynomials, if not supplied the default value of 0.00000000001 will be used.

NormalizeType is an integer in the range $[1..4]$. It determines if and how the points in $\mathbb{X}$ will be normalised. The default value is 2. If **NormalizeType** equals 1, each coordinate is divided

by the maximal absolute value of the corresponding column of the matrix. This ensures that all coordinates of the points are in [-1,1]. With **NormalizeType**=2 no normalisation is done at all. **NormalizeType**=3 shifts each coordinate to [-1,1]. So its minimum is mapped to -1 and the maximum to one, describing a unique affine mapping. The last option is **NormalizeType**=4. In this case, each coordinate is normalised using the column's Euclidean norm.

The algorithm returns a list which contains G and $\mathcal{O}$. In G two polynomials belong together. They have the same support and the first one contains the real part, while the second one contains the complex part.

Non public options in the numcabm.cpkg package

Note that some of the functions are not yet implemented in the complex version but will be added in one of the upcoming releases.

The whole range of options for the algorithm is not exposed publicly but may be set via the numcabm.cpkg package.

The line that needs to be changed is

OptionList := [0,0,0,0,0,1,1,0,1000];

The first option is either 1 (TRUE) or 0 (FALSE) and controls the isAVI check. In the server window additional log output is generated. The result of the algorithm is not changed.

The second option is either 1 (TRUE) or 0 (FALSE) and controls the isAppBB check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The third option is either 1 (TRUE) or 0 (FALSE) and controls the remainingLinearRealtionsInOI check. In the server window additional log output is generated which contains the result. The result of the algorithm is not changed.

The fourth option is either 1 (TRUE) or 0 (FALSE) and controls if re-projection of polynomials is activated. If activated all polynomials in which terms were left out, because of either too small coefficients or a too small contribution to a polynomial, will be reprojected.

The fifth option is either 1 (TRUE) or 0 (FALSE) and controls the switch for the rational recovery. This option is reserved for future use and does not affect the result at the moment.

The sixth option is either 1 (TRUE) or 0 (FALSE) and controls the strict border basis mode. If activated (default) always an approximate border basis will be returned, but some polynomials may not vanish ε-approximately. If deactivated all polynomials will be guaranteed to vanish ε-approximately but may not form an approximate border basis.

The seventh option is either 1 (TRUE) or 0 (FALSE) and controls if the border basis variant of ABM is supposed to be used. Its value needs to be set to false.

The eighth option is either 1 (TRUE) or 0 (FALSE) and controls which kind of truncation mode for small terms/coefficients is used. If disabled (default) terms which have a coefficient less than

Delta will be discarded. If enabled terms which have an average contribution less than **Delta** will be discarded. If option 6 is activated the changed polynomials will be reprojected to give the best possible fit with respect to the new terms.

The ninth option is a positive integer and sets the maximum degree after which the computation will be terminated. This can be used to stop a computation that would otherwise take too long. Note that only a truncated result will be returned which may not represent a full approximate border basis.

Example

The following CoCoA example demonstrates how to call the CABM algorithm:

```
XReal := Mat([[...]]); -- contains the real components of the data
XComp := Mat([[...]]); -- contains the complex components of the data
Epsilon := 0.1; -- an error estimate
AppBB := Num.CABM(XReal, XComp, Epsilon); -- apply the CABM algorithm
```

The BM algorithm for border bases

The BM algorithm for border bases (18) computes a vanishing ideal for a given set of points $\mathbb{X} \subset \mathbb{Q}^n$. The algorithm is implemented in full precision arithmetic, which can cost a considerable amount of time if n and/or $\mathbb{X}$ is large.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.ABM(Points:MAT, 0):Object

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.ABM(Points:MAT, 0):Object

Points has to be a CoCoA matrix containing $\mathbb{X}$ and **Epsilon** has to be set to zero. These parameters always have to be supplied.

The algorithm returns a list which contains G and $\mathcal{O}$.

Example

The following CoCoA example explains how to call the BM algorithm:

```
X := Mat([[...]]); -- measurements
Epsilon := 0; -- threshold number
BB := Num.ABM(X, Epsilon); -- apply the BM algorithm
```

The AVI algorithm

The AVI algorithm (21) computes an approximate vanishing ideal for a given set of points $\mathbb{X} \subset [-1,1]^n \subset \mathbb{Q}^n$ with respect to a threshold number ε and a truncation number τ. The algorithm was proposed by Kreuzer, Poulisse et al. in [28] and computes degree by degree while constructing the polynomials in the approximate border basis. The algorithm relies on the computation of a SVD for the fitting of the polynomials. The scaling of the input data is a necessary prerequisite and influences the properties of the resulting approximate $\mathcal{O}$-border basis G.

Properties

The algorithm will return two sets G and $\mathcal{O}$ with properties which are recalled in 21.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.AVI(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

The last six parameters are optional.

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.AVI(Points:MAT, Epsilon:RAT, Delta:RAT, ForbiddenTerms:LIST,
NormalizeType:INT):Object

Points has to be a CoCoA matrix containing $\mathbb{X}$ and **Epsilon** ε is a rational number. These parameters always have to be supplied.

Delta is used for truncating small coefficients of polynomials, if not supplied the default value of 0.00000000001 will be used.

ForbiddenTerms is a list which contains the terms which are not allowed to show up in the order ideal.

NormalizeType is an integer in the range [1..4]. It determines if and how the points in $\mathbb{X}$ will be normalised. The default value is 2. If **NormalizeType** equals 1, each coordinate is divided by the maximal absolute value of the corresponding column of the matrix. This ensures that all coordinates of the points are in [-1,1]. With **NormalizeType**=2 no normalisation is done at all. **NormalizeType**=3 shifts each coordinate to [-1,1]. So its minimum is mapped to -1 and the maximum to one, describing a unique affine mapping. The last option is **NormalizeType**=4. In this case, each coordinate is normalised using the column's Euclidean norm.

The algorithm returns a list which contains G and $\mathcal{O}$.

Example

The following CoCoA example explains the basic steps how AVI can be used in production modelling:

```
X := Mat([[...]]); -- measurements
P := Mat([[...]]); -- measured production data
Epsilon := 0.1; -- threshold number
AppBB := Num.AVI(X, Epsilon); -- apply the AVI algorithm
ProdPoly := $apcocoa/numerical.ProjectAVI(X, P, AppBB[2]);
-- find a production polynomial with respect to the OI
-- contained in AppBB[2]
```

The Eigenvalue Algorithm

The eigenvalue algorithm (30) computes the zero set of a zero-dimensional polynomial ideal over $P = \mathbb{Q}[x_1, ..., x_n]$. The algorithm expects as input a border basis of the ideal. The border basis does not have to be exact. The result remains reasonably stable as long as the input is a δ-approximate border basis, where δ is in the order of magnitude of the machine accuracy $\varepsilon_{machine}$.

Properties

The algorithm expects as input an (almost) exact $\mathcal{O}$-border basis G and the order ideal $\mathcal{O}$. It will return two matrices containing the real and imaginary parts in double accuracy of the points in the zero set of G.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.RatPoints(BB:LIST, OrderIdeal:LIST)):LIST of MAT

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.RatPoints(BB:LIST, OrderIdeal:LIST)):LIST of MAT

BB has to be a list of polynomials of an (almost) exact $\mathcal{O}$-border basis, **OrderIdeal** is a list containing the elements of $\mathcal{O}$.

The algorithm returns a list containing two matrices. The first one contains the real part and the second one the imaginary part of the points in the zero set of G.

Examples

```
Points := Mat([[2/3,0,0],[0,10,0],[0,0,1/3]]); -- some real points
R := Num.ABM(Points, 0); -- use the ABM algorithm to compute
                         -- an exact border basis
Points := Num.RatPoints(R[1], R[2]);
```

The Approximate Diagonalisation Algorithm

The approximate diagonalisation algorithm (31) computes a set of approximate eigenvectors for a set of square matrices. With the help of the approximate eigenvector matrix the original matrices can be approximately diagonalised.

Properties

Given a set of n square matrices $A_1, ..., A_n \in \mathrm{Mat}_m(\mathbb{Q})$ the algorithm computes a matrix V and its inverse V^{-1} containing the approximate joint eigenvectors of $A_1, ..., A_n$. The user must specify a maximal number of iterations which are performed before the result is returned. A practical value for the maximal number of iterations is 8, after which the matrix V usually has converged.

CoCoAL interface

The algorithm can be called directly from the ApCoCoA UIs (both Eclipse and QT) via the command:

Num.SimDiag(A:LIST, MaxIt:INT):[B:MAT, C:MAT]

Note that if the algorithm is to be called inside another function, the alias Num cannot be used and one has to give the full function name which is

$apcocoa/numerical.SimDiag(A:LIST, MaxIt:INT):[B:MAT, C:MAT]

A is a list of squrare matrices containing rational entries. **MaxIt** is an integer which determines the maximal number of iterations.

The algorithm returns a list containing two matrices. The first one contains the approximate eigenvectors of $A_1, ..., A_n$. The second one is the inverse of the first matrix.

Example

```
-- example 1
-- using exact multiplication matrices
BBasis := Num.ABM([[2/3,0,0], [0,10,0], [0,0,1/3]], 0);
MM1 := Transposed(BB.MultMat(1, BBasis[2], BBasis[1]));
MM2 := Transposed(BB.MultMat(2, BBasis[2], BBasis[1]));
Result := Num.SimDiag([MM1, MM2], 8);
Dec(Result[2]*MM1*Result[1],3);
Dec(Result[2]*MM2*Result[1],3);
-- example 2
-- using approximate multiplication matrices
M1 := Mat([[0, 0, -0.079, -0.018],[0, 0, 0.032, -0.012],
           [1, 0, 1.056, -0.012],[0, 1, -0.060, 1.025]]);
M2 := Mat([[0, -0.063, 0, -0.018],[1, 1.026, 0, -0.012],
           [0, 0, 0, -0.012], [0, 0, 1, 1.025]]);
M1 := Transposed(M1);
M2 := Transposed(M2);
Result := Num.SimDiag([M1,M2], 8);
Dec(Result[2]*M1*Result[1],3); -- M1 approximately diagonalised
Dec(Result[2]*M2*Result[1],3); -- M2 approximately diagonalised
```

8.2 Pseudo Code

In this thesis we make use of pseudo code to write down algorithms. The advantage of pseudo code compared with a fully featured programming languages like e.g. C++/C#/Pascal/... is the higher level of abstraction which it provides. This allows us to write down algorithms in a more compact form without having to pay too much attention to details which might obfuscate the central ideas of the individual procedures.

As we use the algorithm2e package for typesetting the algorithms the control structures (e.g. while or for loops, if ... then ... else clauses, ...) have a Pascal like syntax. We give a short overview of the notation and the conventions which we use for our pseudo code.

If the code should not be self explanatory we will use comments to clarify some of its properties.

Comments

```
/* This is a multi line comment which can be used to describe more complex
   issues                                                                  */
Instructions;
// This is a single line comment
Instructions;
```

In our notation we distinguish between assignments and logical operators.

Assignment and logical operators

```
// Assigns {1,2,3} to the variable A
A := {1,2,3};
// Assigns the boolean value true to B
B := (A = A);
```

Control Structures

The following represents an non-exhaustive overview of some common control structures used throughout computer sciences and how they are represented in this work.

If ... then ... else clause

if *Condition 1* **then**
  Instructions;
else if *Condition 2* **then**
  Instructions;
else
  Instructions;
end

While loop

while *Condition* **do**
  Instructions;
end

For loop

for $i = 1$ **to** *100* **do**
  Instructions;
end

Data Structures

Variables containing elementary data types like integers, real/complex numbers (represented by floating point numbers) and strings will receive no special notation. If the data type of a variable is clear from the context and no ambiguity can arise we will not specify it explicitly.

Data structures

```
// A is a list
A := [1, 5, 5, 9];
// L is an empty list
L := [ ];
// B is a set
B := {1, 5, 9};
/* Algorithm(P_1, P_2) returns two objects (the types should be clear from
   the context) which are stored in the variables b_1 and b_2            */
[b_1, b_2] := Algorithm(P_1, P_2);
```

List Manipulations

Let $A = [a_1, a_2, a_3]$ and $B = [b_1, b_2, b_3]$ be lists.

List manipulations

```
// Concatenates the lists A and B such that C = [a_1, a_2, a_3, b_1, b_2, b_3]
C := concat(A, B);
// Removes the element b_2 from the list C
remove(C, b_2);
/* Returns the minimal element in the list C, if the elements in C can be
   ordered                                                               */
min(C);
/* Returns the maximal element in the list C, if the elements in C can be
   ordered                                                               */
max(C);
```

Matrix Manipulations

As we are often manipulating matrices in our algorithms we will declare the abbreviations for some common matrix operations which we utilise in our pseudo code. Let $A \in \mathrm{Mat}_{m,n}(K)$, $C \in \mathrm{Mat}_{m,k}(K)$, and $R \in \mathrm{Mat}_{l,n}(K)$ be arbitrary matrices.

Matrix manipulations

```
// Return the number of rows and columns of A
m := rows(A); n := cols(A);
// Returns true if A is the zero matrix and false otherwise
isZeroMatrix(A);
/* Returns the row and column index of the first non-zero element in
   matrix A. The matrx is searched row wise. If A is the zero matrix
   [0,0] will be returned                                          */
[r,c] := findFirstNonZeroIndex(A);
// Swaps the i-th and the j-th row of a matrix
swapRows(A,i,j);
// Swaps the i-th and the j-th column of a matrix
swapColumns(A,i,j);
/* Returns a matrix M ∈ Mat_{i,j}(K) whose entries are taken column-wise from
   A. Throws an error if i·j ≠ m·n                                 */
M := reshape(A,i,j);
// Appends the matrix C at the right-hand side of A
M := ( A  C ) ∈ Mat_{m,n+k}(K);
// Appends the matrix C at the left-hand side of A
M := ( C  A ) ∈ Mat_{m,n+k}(K);
// Appends the matrix R at the bottom of A
M := ( A ) ∈ Mat_{m+l,n}(K);
     ( R )
// Appends the matrix R at the top of A
M := ( R ) ∈ Mat_{m+l,n}(K);
     ( A )
```

Bibliography

[1] The Algebraic Oil Research Project, see
`http://www.algebraic-oil.uni-passau.de/`.

[2] B. Otto. *Linear Algebra With Applications. Third Edition.* Prentice Hall, Upper Saddle River New Jersey, 1995.

[3] B. S. W. Schröder. *Mathematical Analysis: A Concise Introduction.* John Wiley & Sons, Hoboken New Jersey, 2007.

[4] J. Fox. *Applied Regression Analysis, Linear Models, and Related Methods.* Sage Publications, 1997.

[5] L. N. Trefethen, D. Bau III. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics (SIAM), 1997.

[6] G. H. Golub, C. F. van Loan. *Matrix Computations. Third Edition.* The Johns Hopkins University Press, Baltimore MD, 1996.

[7] J. W. Demmel. *Applied numerical linear algebra.* Society for Industrial and Applied Mathematics (SIAM), 1997.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms. Second Edition.* MIT Press. September 2001.

[9] R. A. Horn, C. R. Johnson. *Matrix analysis.* Cambridge University Press, 1985. Reprinted with corrections 1990.

[10] A. S. Householder. Unitary Triangularization of a Nonsymmetric Matrix. Journal of the ACM 5 (1958), pp. 339–342.

[11] M. J. D. Powell and J. K. Reid. On applying Householder transformations to linear least squares problems. In Proc. IFIP Congress 1968, North-Holland, Amsterdam, The Netherlands (1969), pp. 122–126.

[12] G. Golub and W. Kahan. Calculating the singular values and pseudoinverse of a matrix. SIAM, Journal on Numerical Analysis 2 (1965), pp. 205-224.

[13] J. H. Wilkinson. Convergence of the LR, QR and related algorithms. The Computer Journal 8 (1965), pp. 77-84.

[14] D. A. Harville. *Matrix Algebra from a Statistician's Perspective.* Springer New York, 1997.

[15] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal and C. A. Sagastizábal. *Numerical Optimization - Theoretical and Practical Aspects. Second Edition.* Universitext. Springer Berlin, 2006.

[16] Y. Saad. *Numerical Methods for Large Eigenvalue Problems - Revised Edition.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2011.

[17] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. *LAPACK Users' Guide. Third Edition.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, SIAM, 1999.

[18] CoCoATeam, CoCoA: a system for doing Computations in Commutative Algebra, see `http://cocoa.dima.unige.it`.

[19] J. Abbott and A. M. Bigatti, CoCoALib: a C++ library for doing Computations in Commutative Algebra, see `http://cocoa.dima.unige.it/cocoalib`.

[20] ApCoCoA: Applied Computations in Commutative Algebra, see `http://www.apcocoa.org`.

[21] B. Buchberger and H. M. Möller. The construction of multivariate polynomials with preassigned zeros. Proceedings of EUROCAM'82, Lecture Notes in Computer Sciences 144, Springer Heidelberg (1982), pp. 24-31.

[22] A. Kehrein and M. Kreuzer. Characterizations of border bases. Journal of Pure and Applied Algebra Vol. 196 (2005), pp. 251-270.

[23] A. Kehrein and M. Kreuzer. Computing border bases. Journal of Pure and Applied Algebra Vol. 205 (2006), pp. 279-295.

[24] B. Mourrain, P. Trébuchet. Generalized normal forms and polynomial system solving. Proceedings of the International Symposium on Symbolic and Algebraic Computation (2005), pp. 253-260.

[25] B. Mourrain. A new criterion for normal form algorithms. M. Fossorier, H. Imai, S. Lin, A. Poli (eds.). Proceedings of AAECC-13. Honolulu 1999. LNCS 1719. Springer Heidelberg (1999), pp. 440-443.

[26] B. Mourrain. Pythagore's dilemma, symbolic-numeric computation, and the border basis method. Symbolic-Numeric Computations (Trends in Mathematics) (2007), pp. 223-243.

[27] J. Abbott, A. Bigatti, M. Kreuzer, L. Robbiano. Computing ideals of points. Journal of Symbolic Computation Vol. 30 (2000), pp. 341-356.

[28] D. Heldt, M. Kreuzer, S. Pokutta and H. Poulisse. Approximate computation of zero-dimenisonal polynomial ideals. Journal of Symbolic Computation Vol. 44 (2009), pp. 1566-1591.

[29] C. Fassino. An Approximation to the Gröbner Basis of Ideals of Perturbed Points: Part I. Preprint 2007.

[30] M. Torrente. *Application of algebra in the oil industry.* Ph.D. Thesis. Scuola Normale Superiore, Pisa, 2009.

[31] C. Paige, Z. Strakos. Bounds for the least squares distance using scaled total least squares. Numerische Mathematik Vol. 91 No. 1 (2002), pp. 93-115.

[32] M. Kreuzer and H. Poulisse. Subideal border bases. Mathematics of Computation Vol. 80 No. 274 (2011), pp. 1135–1154.

[33] J. Abbott, C. Fassino, and M. Torrente. Thinning out redundant empirical data. Mathematics in Computer Science Vol. 1 No. 2 (2007), pp. 375-392.

[34] J. Abbott, C. Fassino, and M. Torrente. Stable border bases for ideals of points. Journal of Symbolic Computation Vol. 43 (2008), pp. 883-894.

[35] C. Fassino: Almost vanishing polynomials for sets of limited precision points. Journal of Symbolic Computation Vol. 45 (2010), pp. 19-37.

[36] G. Dahlquist, Å. Björck, N. Anderson. *Numerical Methods.* Englewood Cliffs New Jersey, 1974.

[37] F. S. Macaulay. *The Algebraic Theory of Modular Systems.* Cambridge Tracts in Mathematics and Mathematical Physics Vol. 19, Cambridge University Press, 1916.

[38] H. M. Möller and T. Sauer. H-bases for polynomial interpolation and system solving. Advances in Computational Mathematics 12 (2000), pp. 335-362.

[39] H. M. Möller and T. Sauer. H-Bases I: The foundation. Curve and Surface fitting: Saint–Malo 1999, Vanderbilt University Press (2000), pp. 325–332.

[40] H. M. Möller and T. Sauer. H-Bases II: Applications to Numerical Problems. Curve and Surface fitting: Saint-Malo 1999, Vanderbilt University Press (2000), pp. 333-342.

[41] T. Sauer. Approximate varieties, approximate ideals, and dimension reduction. Numerical Algorithms 45 (2007), pp. 295-313.

[42] T. Sauer. Polynomial interpolation in several variables: lattices, differences, and ideals. Studies in Computational Mathematics 12 (2006), pp. 191-230.

[43] J. Czekanski. Example computations for the approximate H-basis algorithm. Private email conversation. University Giessen, April 2013.

[44] M. Kreuzer, H. Poulisse and J. Limbeck. Rational approximation of border bases. Preprint 2013.

[45] M. Kreuzer and L. Robbiano. *Computational Commutative Algebra 1.* Springer Heidelberg, 2000.

[46] M. Kreuzer and L. Robbiano. *Computational Commutative Algebra 2.* Springer Heidelberg, 2005.

[47] IEEE 754: reprinted in SIGPLAN Notices Vol. 22 Nr. 2 (1987), pp. 9-25.

[48] W. Auzinger and H. Stetter. An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. R.G. Agarwal, Y.M. Chow, S.J. Wilson (eds.). International Conference on Numerical Mathematics, Singapore 1988. Birkhäuser ISNM 86, Basel (1988), pp. 11-30.

[49] H. Stetter. *Numerical Polynomial Algebra.* SIAM, Philadelphia, 2004.

[50] J. Cai. Computing Jordan Normal Forms Exactly for Commuting Matrices in Polynomial Time. International Journal of Foundations of Computer Science Vol. 5 (1994), pp. 293-302.

[51] L. Mirsky. *An introduction to Linear Algebra.* Oxford University Press, New York, 1955.

[52] A. Bernstein. Almost eigenvectors for almost commuting matrices. SIAM Journal of Applied Mathematics Vol. 21. No. 2 (1971), pp. 232–235.

[53] T. Fu and X. Gao. Simultaneous diagonalization with similarity transformation for non-defective matrices. Conference Proceedings of ICASSP 2006, Vol. 4.

[54] A. J. van der Veen, P. B. Ober, and E. F. Deprettere. Azimuth and elevation computation in high resolution DOA estimation. IEEE Transactions on Signal Processing Vol. 40 (1992), pp. 1828–1832.

[55] M. Haardt and J. A. Nossek. Simultaneous Schur decomposition of several nonsymmetric matrices to achieve automatic pairing in multidimensional harmonic retrieveal problems. IEEE Transactions on Signal Processing Vol. 46 (1998), pp. 161–169.

[56] P. Strobach. Bi-iteration multiple invariance subspace tracking and adaptive ESPRIT. IEEE Transactions on Signal Processing Vol. 48 (2000), pp. 442–456.

[57] L. de Lathauwer, T. de Moor, and J. Vandewalle. Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition. SIAM Journal of Matrix Analysis and Applications Vol. 26 No. 2 (2004), pp. 295-327.

[58] A. Ruhe. On the quadratic convergence of a generalization of the Jacobi method to general matrices. BIT Numerical Mathematics Vol. 8 No. 3 (1968), pp. 210-231.

[59] P. J. Eberlein. A Jacobi-like method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix. Journal of the Society for Industrial and Applied Mathematics Vol. 10 No. 1 (1962), pp. 74–88.

[60] L. Mirsky. On the minimization of matrix norms. The American Mathematical Monthly Vol. 65 No. 2 (1958), pp. 106-107.

[61] H. H. Goldstine and L. P. Horwitz. A procedure for the diagonalization of normal matrices. Journal of the Association for Computing Machinery Vol. 6 (1959), pp. 176-195.

[62] A. Gerstner, R. Byers, and V. Mehrmann. Numerical methods for simultaneous diagonalisation. SIAM. Journal of Matrix Analysis and Applications Vol. 14 No. 4 (1993), pp. 927-949.

[63] E. Organick. *A FORTRAN IV Primer.* Addison-Wesley, 1966.

[64] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Paperback Edition. Oxford University Press, New York, 1988.

[65] D. J. Bates, J. D. Hauenstein, A. J. Sommese, and C. W. Wampler. Bertini: Software for Numerical Algebraic Geometry. Available at `http://www.nd.edu/~sommese/bertini/`.

[66] A. J. Sommese, J. Verschelde, and C. W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. SIAM Journal of Numerical Analysis Vol. 38 (2001), pp. 2022-2046.

[67] J. Nocedal and S. J. Wright. *Numerical Optimization. Springer Series in Operations Research.* Springer New York, 1999.

[68] D. H. Brandwood. A complex gradient operator and its application in adaptive array theory. IEEE Proceedings for Communications, Radar and Signal Processing Vol. 130 No. 1 (1983), pp. 11-16.

[69] J. Abbott, M. Kreuzer, and L. Robbiano. Computing zero-dimensional schemes. Journal of Symbolic Computation Vol. 39 (2005), pp. 31-49.

[70] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. ACM Transactions on Mathematical Software Vol. 33 No. 2 (2007), pp. 1-14.

[71] C. Liner. *Elements of 3-D Seismology. Second Edition.* PennWell, 2004.

[72] K. W. Morton, D. F. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction.* Cambridge University Press, 2005.

[73] A. Fichtner. *Full seismic waveform modelling and inversion. Advances in Geophysical and Environmental Mechanics and Mathematics.* Springer Berlin, 2011.

[74] É. Robein. *Velocities, time-imaging, and depth-imaging in reflection seismics: principles and methods.* EAGE Publications, 2003.

[75] S. Linda and S. George. *Computer Vision.* Prentice-Hall, 2001.

[76] L. Badescu and M. Vladimir. *Algebraic Surfaces.* Springer New York, 2001.