The Dissertation Committee for Mátyás Attila Sustik
certifies that this is the approved version of the following dissertation:

Structured Numerical Problems in Contemporary Applications

Committee:

Inderjit S. Dhillon, Supervisor

Todd Arbogast

Anna Gál

J S. Moore

William H. Press

Structured Numerical Problems in Contemporary Applications

by

Mátyás Attila Sustik, okleveles matematikus

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2013

To my Mother, from whom I inherited my mathematical instincts.

Acknowledgments

My advisor, Inderjit S. Dhillon kindly encouraged and guided me during the years I have worked towards this degree. He taught me that the purpose of scientific writing is not just to document and communicate the research to others, but it is also an integral part of the thought process: the sometimes arduous task of writing can promote the development of new ideas. I am grateful for his insights, help and patience; the latter was tested on a number of occasions, including when I would forget that the proper English term is "matrix A", and not "A matrix". It remains my motivation to continue earning his respect with the work I do.

I thank my committee, Todd Arbogast, Anna Gál, J S. Moore and William H. Press for their thoughtful comments and questions. I am especially grateful to J S. Moore for allowing me to audit his lectures and inviting me to participate in his research group meetings before I was a student at the University of Texas at Austin. His support was fundamental in helping me gain admission to the PhD program in the Department of Computer Science.

Ren-Cang Li was very kind to help me in my effort to improve an eigenvalue computation algorithm that he implemented and still supports for the LAPACK software package. I thank him for the stimulating discussions and his assistance with debugging and testing of my code which we both hope will replace his in the near future.

I also thank all the students in our research group for their help, advice and exchange of ideas. Joint work and publications with former students Joel A. Tropp and Brian Kulis taught me valuable skills about scientific research. I thank Suvrit Sra for the LaTeX tricks I learned from him. I wish them the best of luck in their careers.

My research also touched on many areas other than the ones covered in this dissertation, including automated theorem proving and inverse covariance matrix estimation. These experiences shaped my scientific thinking and helped me arrive where I am today and I think kindly all of those I met during my journey. I am very thankful for the joint work with Cho-Jui Hsieh, Pradeep Ravikumar and Ben Calderhead and I wish them the best of luck in all their endeavors.

I could not have completed the research I present here without the emotional support of my lovely wife Lina. When I was facing setbacks and disappointments she helped me pull through. When I was excited about an idea she listened and got excited about it together with me. I thank her for her encouragement and unwavering love.

Structured Numerical Problems in Contemporary Applications

Publication No. ____________________

Mátyás Attila Sustik, Ph.D.
The University of Texas at Austin, 2013

Supervisor: Inderjit S. Dhillon

The presence of structure in a computational problem can often be exploited and can lead to a more efficient numerical algorithm. In this dissertation, we look at structured numerical problems that arise from applications in wireless communications and machine learning that also impact other areas of scientific computing.

In wireless communication system designs, certain structured matrices (frames) need to be generated. The design of such matrices is equivalent to a symmetric inverse eigenvalue problem where the values of the diagonal elements are prescribed. We present algorithms that are capable of generating a larger set of these constructions than previous algorithms. We also discuss the existence of equiangular tight frames—frames that satisfy additional structural properties.

Kernel learning is an important class of problems in machine learning. It often relies on efficient numerical algorithms that solve underlying convex optimization problems. In our work, the objective functions to be minimized are the von Neumann and the LogDet Bregman matrix divergences. The algorithm that solves this optimization problem performs matrix updates based on repeated eigendecompositions of diagonal plus rank-one matrices in the case of von Neumann matrix divergence, and Cholesky updates in case of the LogDet Bregman matrix divergence. Our contribution exploits the low-rank representations and the structure of the constraint matrices, resulting in more efficient algorithms than previously known.

We also present two specialized zero-finding algorithms where we exploit the structure through the shape and exact formulation of the objective function. The first zero-finding task arises during the matrix update step which is part of the above-mentioned kernel learning application. The second zero-finding problem is for the secular equation; it is equivalent to the computation of the eigenvalues of a diagonal plus rank-one matrix. The secular equation arises in various applications, the most well-known is the divide-and-conquer eigensolver. In our solutions, we build upon a somewhat forgotten zero-finding method by P. Jarratt, first described in 1966. The method employs first derivatives only and needs the same amount of evaluations as Newton's method, but converges faster. Our contributions are the more efficient specialized zero-finding algorithms.

Table of Contents

List of Tables

List of Figures

Chapter 1

Overview

1.1 Motivation

Specialized algorithms can solve many structured numerical problems more efficiently. For example, eigensolvers exploiting the structure are available for symmetric, tridiagonal, diagonal plus rank-one, Toeplitz and many other types of matrices. These specialized algorithms outperform a general eigensolver in many respects. A different example is provided by the zero-finding algorithm specifically developed for the secular equation, which is part of the divide-and-conquer eigensolver. In this state of the art solution a modified version of Newton's method takes into account the shape of the function in order to accelerate the convergence. When we exploit the symmetry, a low-rank representation or other structure present in the problem formulation we can expect significant improvement in efficiency.

In this dissertation, we look at applications in wireless communication system design, machine learning and scientific computing in which numerical computational problems involving structured matrices arise. Our main contribution is a collection of algorithms which solve these computational problems more efficiently.

1.2 Contributions

In Chapters 2 and 3, we present our research on *frame*s. Frames are vector sets that can be viewed as a generalization of orthogonal vector bases [14, 26]. Unlike a base however, a set of frame vectors is overcomplete, which means that there is more than one way to combine them to represent elements of the vector space. However, the frame structure allows us to select one natural representation which is useful in wireless applications [58, 60]. We formulate frame construction as an inverse eigenvalue problem and recognize that the Chan-Li and Bendel-Mickey algorithms [4, 13] can be used to generate frames. We present generalized versions of the Chan-Li and Bendel-Mickey algorithms that can produce a larger class of these frame constructions. We also describe versions of the algorithms which operate on a factored representation that exploits low rank and hence reduces the memory requirements from $n(n+1)/2$ to $n(r+2)$, where r denotes the rank, and n is the matrix dimension.

Additional structural restrictions apply to the class of frames called *equiangular tight* frames which find applications in communications, coding theory, and sparse approximation, see [53, 57] and references therein. In Chapter 3 we use algebraic number theory to prove conditions on the existence of real and certain complex equiangular tight frames.

Kernel learning is an important problem in machine learning, which is an important tool in classifications and clustering, see [18, 39] for some recent applications. In mathematical terms, the goal is to find a positive-definite matrix, the so-called *kernel matrix*, that satisfies some linear constraints that are extracted from observations, and also minimizes a given distance measure. The resulting kernel matrix can be used for clustering, classification and other machine learning tasks. In Chapter 4 we discuss specific numerical algorithms based on Bregman's algorithm [8], that are central to the solution of such a kernel learning problem. We exploit the specific form of the distance measures used, which are the von Neumann and LogDet *matrix divergence*s [12] in this application. We also extend the formulation to include rank-deficient matrices and subsequently exploit the low-rank decomposition of the solution matrix. Our algorithms take advantage of the factored form that captures the structure and provide significant gains in efficiency by reducing the memory requirements from $O(n^2)$ to $O(nr)$ while the computational cost of the repeated matrix updates decreases from $O(n^2)$ to $O(r^2)$.

In Chapter 5, we take a closer look at the zero-finding problem that is central to the kernel learning application that we presented in Chapter 4. The definition of the objective function for this zero-finding task involves the matrix exponential and that makes the evaluation of the function and its derivative costly. We present efficient zero-finding algorithms that exploit the structure present in the function definition, namely we take advantage of the sharing of computation between the evaluation of the function and its derivative. We find that the additional cost of computing the derivative is sufficiently less than the cost of the function evaluation alone, and that makes using Newton's method more efficient than non-derivative based approaches. We also exploit the shape of the objective function through a reparametrization. Furthermore, a seemingly forgotten improvement to Newton's method due to Jarratt [40], which does not need additional function evaluations, allows us to speed up convergence even further.

The zero-finding algorithm due to Jarratt deserves further attention. If we assume that the cost of the function evaluation is relatively expensive, this zero-finder is faster than Newton's method; it has order of convergence $1 + \sqrt{3} \sim 2.732$ [40]. Given the function and derivative values, the cost of computing the next approximation step is constant for both Newton's and Jarratt's method. In many application the function evaluation step can easily dominate the computational cost. This was true for the function involving the matrix exponential in Chapter 5 and it is also true for the secular equation [10,45]. In Chapter 6 we discuss our improvement to the secular equation solver by adapting Jarratt's method for this special case. Finding the zeros of the secular equation amounts to finding the eigenvalues of a symmetric diagonal plus rank-one matrix and it is an important step of the divide and conquer eigensolver [16]. The divide and conquer algorithm is highly parallelizable [25] and it is one of the prominent methods that computes the eigenvalues of a symmetric or singular values of an arbitrary matrix. Our improved algorithm exhibits faster convergence and this reduces the maximum number of iterations needed per zero. Reducing the maximum work needed per zero is important in a parallel implementation often more so than reducing the total computational work. We also present a proof aided by a computer algebra package that establishes the order of convergence for Jarratt's method and when it is embedded in our secular equation solver algorithm.

The work that we present in this dissertation has resulted in the following publications and should be referred to for further exposition:

[24] **Generalized finite algorithms for constructing Hermitian matrices with prescribed diagonal and spectrum** with I. S. Dhillon, R. W. Heath Jr. and J. A. Tropp appeared in *SIAM Journal on Matrix Analysis and Applications, vol. 27, no. 1, pages 61–71, May 2005.*

[55] **On the Existence of Equiangular Tight Frames** with J. A. Tropp, I. S. Dhillon, and R. W. Heath Jr. appeared in *Linear Algebra and its Applications, vol. 426:2-3, pages 619-635, October 2007.*

[41] **Learning Low–Rank Kernel Matrices** with B. Kulis and I. S. Dhillon appeared in *Proceedings of the Twenty–third International Conference on Machine Learning (ICML), pages 505-512, July 2006.*

[42] **Low–Rank Kernel Learning with Bregman Matrix Divergences** with B. Kulis and I. S. Dhillon appeared in *Journal of Machine Learning Research, 2009, pages 341–376.*

[56] **On a zero-finding problem involving the matrix exponential** with I. S. Dhillon appeared in *SIAM Journal on Matrix Analysis and Applications 2012, vol. 33, no. 4, pages 1237–1249.*

Chapter 2

Wireless Communication Application

In this chapter we discuss a matrix construction problem that has arisen in wireless communications [58, 60]. It turns out that $d \times N$ matrices, $d < N$, with d identical nonzero singular values and with prescribed column norms satisfy a certain "sum capacity" bound and "minimum squared correlation" property that is important in wireless applications. The column vectors form an over–complete set and they yield a *natural* representation for each vector of the vector space. These matrices are called *frames* and can be viewed as generalizations of orthonormal basis; they. The problem can be formulated as a symmetric inverse eigenvalue problem where the values of the diagonal elements are pre–specified.

We present new algorithms that can replace the diagonal entries of a Hermitian matrix by any set of diagonal entries that majorize the original set without altering the eigenvalues of the matrix. Both the Bendel–Mickey [4] and the Chan–Li algorithms [13] are special cases of the proposed procedures.

Another example of an application where our algorithms can be applied is the problem to construct Hermitian matrices with unit diagonal and prescribed nonnegative eigenvalues. Such matrices are called correlation matrices; Davies and Higham [17] discuss several applications that require such matrices, ranging from the generation of test matrices for eigenvalue solvers to the design of statistical experiments.

Our algorithm generalizes the Bendel–Mickey and Chan–Li algorithms. Like them, our techniques use a sequence of $(N-1)$ or fewer plane rotations; the crux of the matter is the strategy for selecting the planes of rotation.

The generalized algorithms can produce a richer set of matrices with prescribed diagonal entries and eigenvalues, making it possible to find solutions that satisfy additional properties or better suit the application. For numerical examples and further discussion see [24].

It is well known that any positive semi-definite matrix $A \in \mathbb{M}_N$ can be expressed as the product X^*X where $X \in \mathbb{M}_{d,N}$ and $d \geq \operatorname{rank} A$. With this factorization, the two–sided transformation $A \mapsto Q^*AQ$ is equivalent to a one–sided transformation $X \mapsto XQ$. In consequence, the machinery of our new algorithms requires little adjustment to produce these factors, yielding the 'one–sided' versions of the algorithms.

We finish the chapter with a discussion about the existence of *equiangular tight frames.* These $d \times N$ frame matrices need to satisfy additional structural properties and they exist only for specific values of d and N.

2.1 Background

We briefly introduce (tight) frames, majorization, partially ordered sets, plane rotations and the Chan–Li and Bendel–Mickey algorithms prior to our generalized algorithm presentation. For further details we refer to [24].

2.1.1 Frames

Frames are an over–complete set of vectors which yield one *natural* representation for each element of the vector space while may allow many other representations. Frames can be viewed as generalizations of orthonormal basis.

Any vector x of a d–dimensional vector space can be written in the form

$$x = \sum_{i=1}^{d} (e_i^* x) e_i,$$

where the vectors e_i $(i = 1, 2, \ldots d)$ form an orthonormal basis. We generalize this concept by considering an over–complete set of unit–length vectors denoted by f_i $(i = 1, 2, \ldots, N,\ N > d)$ for which

$$x = \alpha \sum_{i=1}^{N} (f_i^* x) f_i, \tag{2.1}$$

holds for every x vector in the space with a fixed $\alpha \in \mathbb{R}$. We call such a system of vectors a *tight* frame[1].

[1]Uniform tight frame and Parseval tight frame are also used in the literature.

We form the $d \times N$ matrix S consisting of f_i as its column vectors. By substituting $e_1, e_2, \ldots, e_d$ into (2.1) we deduce that $SS^* = (1/\alpha)I$:

$$\frac{1}{\alpha}\delta_{jk} = \frac{1}{\alpha}e_j^* e_k = \left(\sum_{i=1}^{N}(f_i^* e_j) f_i\right)^* \cdot e_k = \sum_{i=1}^{N}(f_i^* e_j)(f_i^* e_k).$$

Since $N = \mathrm{trace}(S^*S) = \mathrm{trace}(SS^*)$, we also have $\alpha = d/N$.

The $G = S^*S$ Gram–matrix has a unit diagonal and its eigenvalues are 0 and N/d with multiplicities $N-d$ and d respectively. Given an $N \times N$ matrix G with such spectrum and diagonal, one can produce a frame matrix S via the reduced eigendecomposition of G, hence the algorithms of this section can be used to generate frames.

2.1.2 Majorization

An intuitive definition of majorization is that one vector majorizes another if the former has "more average" entries than the latter. We make this notion precise:

Definition 1. *Let $\boldsymbol{a}$ be a real, N-dimensional vector, and denote its kth smallest component by $a_{(k)}$. This number is called the kth* order statistic *of $\boldsymbol{a}$.*

Definition 2. *Let $\boldsymbol{a}$ and $\boldsymbol{z}$ be real N–dimensional vectors, and suppose that their order statistics satisfy the following relationships.*

$$\begin{aligned} a_{(1)} &\le z_{(1)}, \\ a_{(1)} + a_{(2)} &\le z_{(1)} + z_{(2)}, \\ &\vdots \end{aligned}$$

$$a_{(1)} + a_{(2)} + \cdots + a_{(N-1)} \leq z_{(1)} + z_{(2)} + \cdots + z_{(N-1)}, \qquad \textit{and also}$$

$$a_{(1)} + a_{(2)} + \cdots + a_{(N)} = z_{(1)} + z_{(2)} + \cdots + z_{(N)}.$$

Then we say that $\boldsymbol{z}$ majorizes $\boldsymbol{a}$*, and we write* $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$[2]*. If each of the inequalities is strict, then* $\boldsymbol{z}$ strictly majorizes $\boldsymbol{a}$*, and we write* $\boldsymbol{z} \succ \boldsymbol{a}$*.*

Majorization defines the precise relationship between the diagonal entries and eigenvalues of a Hermitian matrix.

Theorem 1 (Schur–Horn)**.** *The diagonal entries of a Hermitian matrix majorize its eigenvalues. Conversely, if* $\boldsymbol{a} \succcurlyeq \boldsymbol{\lambda}$*, then there exists a Hermitian matrix with diagonal entries listed by* $\boldsymbol{a}$ *and eigenvalues listed by* $\boldsymbol{\lambda}$*.*

I. Schur demonstrated the necessity of the majorization condition in 1923, and A. Horn proved the converse some thirty years later [37].

2.1.3 Plane Rotation

We can use a plane rotation to modify the diagonal entries of a Hermitian matrix while preserving its spectrum. Let us suppose that A is a 2×2 matrix with diagonal $\boldsymbol{a}$ that we wish to transform to $\boldsymbol{z}$, where $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$. Without loss of generality, we can assume that $a_1 \leq z_1 \leq z_2 \leq a_2$. We can explicitly construct a real plane rotation Q so that the diagonal of Q^*AQ equals $\boldsymbol{z}$. Recall that a two–dimensional plane rotation is an orthogonal matrix of the

[2]Note that the direction of the partial ordering is reversed in some treatments, but we follow the convention in Horn & Johnson's book [37].

form

$$Q = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \tag{2.2}$$

where $c^2 + s^2 = 1$ [29]. The desired plane rotation yields the matrix equation

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^* \begin{bmatrix} a_1 & a_{21}^* \\ a_{21} & a_2 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} z_1 & z_{21}^* \\ z_{21} & \widetilde{z}_2 \end{bmatrix}. \tag{2.3}$$

The equality of the upper–left entries allows us to determine the values of c and s. Details are in [24, page 4].

2.1.4 Bendel–Mickey and Chan–Li Algorithms

In the sequel, we use $\mathbb{M}_N$ to denote the set of complex $N \times N$ matrices and $\mathbb{M}_{d,N}$ to denote the set of complex $d \times N$ matrices.

The Bendel–Mickey algorithm produces random (Hermitian) correlation matrices with given spectrum [4]. Suppose that $A \in \mathbb{M}_N$ is a Hermitian matrix with trace $A = N$. If A does not have a unit diagonal, we can locate two diagonal elements so that $a_{jj} < 1 < a_{kk}$; otherwise, the trace condition would be violated. It is then possible to construct a real rotation Q in the jk–plane for which $(Q^*AQ)_{jj} = 1$. The transformation $A \mapsto Q^*AQ$ preserves the conjugate symmetry and the spectrum of A, but it reduces the number of non–unit diagonal entries by at least one. Therefore, at most $(N-1)$ rotations are required before the resulting matrix has a unit diagonal.

The Chan–Li algorithm, on the other hand, was developed as a constructive proof of the Schur–Horn Theorem [13]. Suppose that $\boldsymbol{a} \succcurlyeq \boldsymbol{\lambda}$. The

Chan–Li algorithm begins with the diagonal matrix $\Lambda \overset{\text{def}}{=} \operatorname{diag} \boldsymbol{\lambda}$. Then it applies a sequence of $(N-1)$ cleverly chosen (real) plane rotations to generate a real, symmetric matrix A with the same eigenvalues as Λ but with diagonal entries listed by $\boldsymbol{a}$.

The Bendel–Mickey algorithm is a surjective map from the set of Hermitian matrices with spectrum $\boldsymbol{\lambda}$ onto the set of correlation matrices with spectrum $\boldsymbol{\lambda}$. If the initial matrix is chosen uniformly at random (which may be accomplished with standard techniques [52]), the result may be construed as a random correlation matrix. The distribution of the output, however, is unknown [35]. On the other hand, due to the special form of the initial matrix and the rigid choice of rotations, the Chan–Li algorithm cannot construct very many distinct matrices with a specified diagonal.

2.2 Generalized Bendel–Mickey Algorithm

Let $\boldsymbol{z}$ and $\boldsymbol{a}$ be N–dimensional vectors for which $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$. We will show how to transform a Hermitian matrix A with diagonal $\boldsymbol{a}$ and spectrum $\boldsymbol{\lambda}$ into one with diagonal $\boldsymbol{z}$ and spectrum $\boldsymbol{\lambda}$ using a sequence of plane rotations. It is enough to prove the result when the components of $\boldsymbol{a}$ and $\boldsymbol{z}$ are sorted in ascending order, so we place that restriction in the sequel.

Suppose that $\boldsymbol{a} \neq \boldsymbol{z}$. On account of the majorization relationship, it is possible to select indices $i < j$ that satisfy two properties: $a_i < z_i \leq z_j < a_j$ and $a_k = z_k$ for all k strictly between i and j. If $z_i - a_i \leq a_j - z_j$, then we construct a plane rotation Q in the (i, j)-plane such that $(Q^* A Q)_{ii} =$

z_i. Otherwise, we find Q such that $(Q^*AQ)_{jj} = z_j$. Either rotation can be calculated from simple equations, see [24, page 5].

To see that this strategy can be repeated, we just need to check that $\boldsymbol{z}$ majorizes the diagonal of Q^*AQ. In the first case, the plane rotation transforms a_i to z_i and a_j to $a_i + a_j - z_i$, while the remaining diagonal entries do not change. Since $a_i < z_i \le z_j \le a_i + a_j - z_i < a_j$ the diagonal entries of Q^*AQ remain in ascending order. The first $(i-1)$ majorization conditions are clearly unaffected. Notice that

$$\sum_{\ell=1}^{i-1} a_\ell + z_i \le \sum_{\ell=1}^{i-1} z_\ell + z_i,$$

which proves the ith majorization condition. The next $(j-i-1)$ majorization inequalities follow in consequence of a_{kk} being equal to z_k whenever $i < k < j$. The rest of the majorization conditions hold since

$$\sum_{\ell=1}^{i-1} a_\ell + z_i + \sum_{k=i+1}^{j-1} a_k + (a_i + a_j - z_i) = \sum_{\ell=1}^{j} a_\ell \le \sum_{\ell=1}^{j} z_\ell.$$

The argument in the case when $z_i - a_i > a_j - z_j$ is similar. It follows that our rotation strategy may be applied until $\operatorname{diag} A = \boldsymbol{z}$. This proof leads to Algorithm 1. The computed Hermitian matrix has diagonal entries $\boldsymbol{z}$ and eigenvalues equal to that of A.

The algorithm requires about $12N^2$ real floating–point operations if conjugate symmetry is exploited, and the storage of about $N(N+1)/2$ complex floating–point numbers.

Algorithm 1: Generalized Bendel–Mickey algorithm

```
Input  : Matrix $A$: $N \times N$ Hermitian with diagonal $\boldsymbol{a}$; vector $\boldsymbol{z}$:
         satisfies $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$, where both $\boldsymbol{a}$ and $\boldsymbol{z}$ are arranged in
         ascending order.
Output: Updated matrix $A$ that is orthogonally similar and has
         diagonal $\boldsymbol{z}$.
1 repeat
2     Find $i < j$ for which $a_i < z_i$, $z_j < a_j$ and $a_k = z_k$ for $i < k < j$
      (in our implementation we pick the smallest such $i$). If no such
      pair exists, we are either done ($\boldsymbol{z} = \boldsymbol{a}$) or the majorization
      condition is violated.
3     Construct a plane rotation $Q$ in the $(i, j)$–plane to transform $a_i$
      to $z_i$ in the case $z_i - a_i \le a_j - z_j$ or transform $a_j$ to $z_j$
      otherwise.
4     Replace $A$ by $Q^*AQ$.
5 until the diagonal is transformed to $\boldsymbol{z}$
```

2.3 Generalized Chan–Li Algorithm

Distinct algorithms arise by changing the strategy for selecting the planes of rotation. Let $\boldsymbol{z}$ and $\boldsymbol{a}$ be N–dimensional vectors for which $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$. As before, we assume that they are sorted in ascending order, and suppose that A is a Hermitian matrix with diagonal $\boldsymbol{a}$. We now exhibit a different method for transforming the diagonal of A to $\boldsymbol{z}$ while preserving its eigenvalues. It can be viewed as a generalization of the Chan–Li algorithm [13].

We will use induction on the dimension, so grant us for a moment that we can perform the advertised feat on Hermitian matrices of size $(N - 1)$. Now we consider N–dimensional vectors for which $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$, and suppose that $\operatorname{diag} A = \boldsymbol{a}$. The majorization condition implies that $a_1 \le z_1 \le z_N \le a_N$, so it is always possible to select a least integer $j > 1$ so that $a_{j-1} \leq z_1 \leq a_j$. Let P_1 be a permutation matrix for which

$$\operatorname{diag}(P_1^* A P_1) = (a_1, a_j, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_N). \tag{2.4}$$

Observe that $a_1 \leq z_1 \leq a_j$ and $a_1 \leq a_1 + a_j - z_1 \leq a_j$. We compute (see [24, page 6]) a two–dimensional plane rotation Q_2 that sets the upper left entry of

$$Q_2^* \begin{bmatrix} a_1 & a_{j1}^* \\ a_{j1} & a_j \end{bmatrix} Q_2 \tag{2.5}$$

to z_1. If we define the rotation

$$P_2 \overset{\text{def}}{=} \begin{bmatrix} Q_2 & \mathbf{0}^* \\ \mathbf{0} & I_{N-2} \end{bmatrix}, \tag{2.6}$$

then

$$P_2^* P_1^* A P_1 P_2 = \begin{bmatrix} z_1 & \boldsymbol{v}^* \\ \boldsymbol{v} & A_{N-1} \end{bmatrix}, \tag{2.7}$$

where $\boldsymbol{v}$ is an appropriate vector and A_{N-1} is an appropriate submatrix with

$$\operatorname{diag}(A_{N-1}) = (a_1 + a_j - z_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_N). \tag{2.8}$$

In order to apply the induction hypothesis, it remains to check that the vector $(z_2, z_3, \ldots, z_N)$ majorizes the diagonal of A_{N-1}. We accomplish this in three steps. First, recall that $a_k \leq z_1$ for $k = 2, \ldots, j-1$. Therefore,

$$\sum_{k=2}^{m} z_k \geq (m-1)\, z_1 \geq \sum_{k=2}^{m} a_k \tag{2.9}$$

for each $m = 2, \ldots, j-1$. The sum on the right-hand side obviously exceeds the sum of the smallest $(m-1)$ entries of the vector $\operatorname{diag} A_{N-1}$, so the first

$(j-2)$ majorization inequalities are in force. Second, we use the fact that $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$ to calculate that, for $m = j, \dots, N$,

$$\sum_{k=2}^{m} z_k = \sum_{k=1}^{m} z_k - z_1 \geq \sum_{k=1}^{m} a_k - z_1 = (a_1 + a_j - z_1) + \sum_{k=2}^{j-1} a_k + \sum_{k=j+1}^{m} a_k. \tag{2.10}$$

Once again, observe that the sum on the right–hand side exceeds the sum of the smallest $(m-1)$ entries of the vector $\operatorname{diag} A_{N-1}$, so the remaining majorization inequalities are in force. Finally, rearranging the relation $\sum_{k=1}^{N} z_k = \sum_{k=1}^{N} a_k$ yields $\sum_{k=2}^{N} z_k = \operatorname{trace} A_{N-1}$. In consequence, the induction furnishes a rotation Q_{N-1} that sets the diagonal of A_{N-1} equal to the vector $(z_2, \dots, z_N)$. Defining

$$P_3 \overset{\text{def}}{=} \begin{bmatrix} 1 & \mathbf{0}^* \\ \mathbf{0} & Q_{N-1} \end{bmatrix}, \tag{2.11}$$

we see that conjugating A by the orthogonal matrix $P = P_1 P_2 P_3$ transforms the diagonal entries of A to $\boldsymbol{z}$ while retaining the spectrum $\boldsymbol{\lambda}$. This proof leads to Algorithm 2 which has the same functionality and complexity as the generalized Bendel–Mickey algorithm.

2.4 One–sided Algorithms

It is well known that any positive semidefinite matrix $A \in \mathbb{M}_N$ can be expressed as the product X^*X where $X \in \mathbb{M}_{d,N}$ and $d \geq \operatorname{rank} A$. With this factorization, the two–sided transformation $A \mapsto Q^*AQ$ is equivalent to a one–sided transformation $X \mapsto XQ$. In consequence, the machinery of the generalized Bendel–Mickey algorithm requires little adjustment to produce these

Algorithm 2: Generalized Chan–Li algorithm

```
Input  : Matrix $A$: $N \times N$ Hermitian with diagonal $\boldsymbol{a}$; vector $\boldsymbol{z}$:
         satisfies $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$, where both $\boldsymbol{a}$ and $\boldsymbol{z}$ are arranged in
         ascending order.
Output: Updated matrix $A$ that is orthogonally similar and has
         diagonal $\boldsymbol{z}$.
1 Set $i = 1$.
2 repeat
3     Find the least $j > i$ so that $a_{j-1} \leq z_i \leq a_j$.
4     Use a symmetric permutation to set $a_{i+1}$ equal to $a_j$ while
      shifting diagonal entries $i+1, \dots, j-1$ one place down the
      diagonal.
5     Construct a plane rotation $Q$ in the $(i, i+1)$–plane
      transforming the diagonal accordingly.
6     Replace $A$ by $Q^* A Q$.
7     Use a symmetric permutation to re–sort the diagonal entries of
      $A$ in ascending order.
8     $i \to i + 1$.
9 until $i = N$
```

factors. The following algorithm generalizes the one–sided version proposed by Davies and Higham in [17].

Suppose that $\boldsymbol{z}$ and $\boldsymbol{a}$ are non–negative vectors of length N with ascending entries. Assume, moreover, that $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$. The following algorithm takes as input a $d \times N$ complex matrix X whose squared column norms are listed by $\boldsymbol{a}$ and transforms it into a matrix with the same singular spectrum and with squared column norms listed by $\boldsymbol{z}$. The algorithm requires about $12dN$ real floating–point operations and storage of $N(d+2)$ complex floating–point numbers including the desired column norms and the current column norms. A similar modification of our generalized Chan–Li algorithm also leads to a

Algorithm 3: One–sided generalized Bendel–Mickey algorithm

```
Input : Matrix $X$: $d \times N$ with squared column norms $\boldsymbol{a}$; vector
        $\boldsymbol{z}$: satisfies $\boldsymbol{z} \succcurlyeq \boldsymbol{a}$, where both $\boldsymbol{a}$ and $\boldsymbol{z}$ are arranged in
        ascending order.
Output: Updated matrix $X$ with squared column norms $\boldsymbol{z}$ and
        preserved singular spectrum.
1 repeat
2     Find $i < j$ for which $\|\boldsymbol{x}_{i-1}\|_2^2 < z_i$, $z_j < \|\boldsymbol{x}_j\|_2^2$ and $\|\boldsymbol{x}_k\|_2^2 = z_k$
      for $i < k < j$. If no such pair exists, we are either done or the
      majorization condition is violated.
3     Form the quantities
          $a_i = \|\boldsymbol{x}_i\|_2^2, \qquad a_{ji} = \langle \boldsymbol{x}_j,\ \boldsymbol{x}_i \rangle \qquad \text{and} \qquad a_j = \|\boldsymbol{x}_j\|_2^2.$
4     Construct a plane rotation $Q$ in the $(i, j)$–plane to transform $a_i$
      to $z_i$ in the case $z_i - a_i \le a_j - z_j$, or transform $a_j$ to $z_j$
      otherwise.
5     Replace $X$ by $XQ$.
6 until all column norms are as desired
```

one–sided version.

Chapter 3

On the Existence of Equiangular Tight Frames

Frames with well separated vectors (considering the pairwise angles) are of special interest. In this chapter we discuss frames for which the pairwise angles are the same, they are called *equiangular tight frame*s or ETFs for short.

Equiangular line systems do not need to have the tight frame property; they first appeared in the literature on discrete geometry [44, 59]. The earliest results on systems that meet both the frame property and satisfy the equiangular conditions appear in Welch's work [62]. More recently, ETFs have found applications in communications, coding theory, and sparse approximation [53, 57]. In particular, Holmes and Paulsen have shown that an ETF provides an error-correcting code that is maximally robust against two erasures [36]. ETFs are sometimes called *Maximum Welch-Bound-Equality Sequences*, or *optimal Grassmannian frames*, or *two-uniform frames.* We prefer the more descriptive term "equiangular tight frame."

3.1 Background

We start by introducing the formal definition for equiangular tight frames or ETFs. Next, we summarize some necessary facts from the theory of algebraic integers.

3.1.1 Equiangular tight frame

We begin with a formal definition.

Definition 3. *Let S be a $d \times N$ matrix whose columns are $\boldsymbol{s}_1, \dots, \boldsymbol{s}_N$. The matrix S is called an* equiangular tight frame *if it satisfies three conditions.*

1. *Each column has unit norm: $\|\boldsymbol{s}_n\|_2 = 1$ for $n = 1, \dots, N$.*

2. *The columns are equiangular. For some nonnegative α, we have*
$$|\langle \boldsymbol{s}_m,\ \boldsymbol{s}_n \rangle| = \alpha \qquad \text{when } 1 \le m < n \le N.$$

3. *The columns form a tight frame. That is, $SS^* = (N/d)\, I$.*

If the entries of S are real (resp. complex) numbers, we refer to S as a real ETF *(resp.* complex ETF*). Note that condition (3) implies $N \ge d$.*

By associating each column of an ETF with its one–dimensional span, we may view an ETF as a collection of lines through the origin. The number α in the definition represents the cosine of the acute angle between each pair of lines. Connected with this geometric interpretation are some known facts.

Proposition 1. *Fix $d > 1$, and suppose that S is a $d \times N$ matrix with unit norm columns. Then*
$$\max_{i \ne j} |\langle \boldsymbol{s}_i,\ \boldsymbol{s}_j \rangle| \ge \sqrt{\frac{N-d}{d\,(N-1)}}.$$
This bound is attained if and only if S is an ETF.

The inequality is originally due to Welch [62]. Strohmer and Heath offer a direct argument that also gives the second statement [53]. A very insightful proof appears in [15].

The literature also contains upper bounds on the number of equiangular lines that can exist in a Euclidean space. The usual proof of these results [19] is not very accessible. An elegant and elementary argument in [55, page 20] relies only on matrix theory.

Theorem 2. *An upper bound on the number N of equiangular lines that can be constructed in a d–dimensional Euclidean space is*

$$\begin{array}{ll} N \leq \frac{1}{2}\, d\,(d+1) & \textit{in } \mathbb{R}^d, \textit{ and} \\ N \leq d^2 & \textit{in } \mathbb{C}^d. \end{array} \tag{3.1}$$

3.1.2 Algebraic integers

Our proofs on the existence of real and certain complex equiangular frames depend on some basic facts from field theory. Lang's textbook is a standard introduction to this material [43]. We review the essential definitions and facts that we explicitly need in our results.

The ring of integers and the field of rationals are denoted by $\mathbb{Z}$ and $\mathbb{Q}$ respectively. A polynomial whose coefficients are drawn from a subfield $\mathbb{F}$ of the complex numbers is referred to as a *polynomial over* $\mathbb{F}$. The complex number ω is *algebraic* over $\mathbb{F}$ if it is the root of some polynomial over $\mathbb{F}$. An *algebraic integer* is the root of a monic polynomial with integer coefficients.

Fact 1. *The algebraic integers form a ring, i.e., they are closed under addition and multiplication.*

Fact 2. *The roots of a monic polynomial with algebraic integer coefficients are also algebraic integers.*

The *minimal polynomial* of ω over $\mathbb{F}$ is the (unique) lowest degree monic polynomial over $\mathbb{F}$ that contains ω among its roots.

Fact 3. *A minimal polynomial over* $\mathbb{F}$ *has simple roots.*

Two numbers that have the same minimal polynomial over $\mathbb{F}$ are called *algebraic conjugates* over $\mathbb{F}$.

Fact 4. *Suppose that* ω *and* ζ *are algebraic conjugates over* $\mathbb{F}$*. If* p *is a polynomial over* $\mathbb{F}$ *that has* ω *as a root with multiplicity* m*, then* ζ *is also a root of* p *with multiplicity* m*.*

Fact 5. *Suppose that* ζ_p *is a primitive pth root of unity. The ring of algebraic integers in the field* $\mathbb{Q}(\zeta_p)$ *equals the ring* $\mathbb{Z}[\zeta_p]$*.*

Fact 6. *The set of real algebraic integers in* $\mathbb{Z}[\zeta_p]$ *coincides with the ring* $\mathbb{Z}[2 \operatorname{Re} \zeta_p]$*.*

The following lemma that is needed for our results on real ETFs illustrates how to use the above facts.

Lemma 1. *Let A be an Hermitian matrix whose entries are algebraic integers. Then the eigenvalues of A are real algebraic integers.*

In addition, assume that the entries of A belong to a subfield $\mathbb{F}$ of the complex numbers. If A has an eigenvalue ω whose multiplicity differs from the multiplicity of every other eigenvalue, then ω belongs to $\mathbb{F}$.

Proof. The matrix A is Hermitian, so its eigenvalues are real numbers. By definition, an eigenvalue of A is a root of the characteristic polynomial $t \mapsto \det(t\,\mathbf{I} - A)$. Since the entries of A are algebraic integers, Fact 1 implies that the characteristic polynomial is a monic polynomial with algebraic integer coefficients. Then Fact 2 shows that the eigenvalues of A are algebraic integers.

Assume that the entries of A belong to $\mathbb{F}$. Thus, the eigenvalues of A are algebraic over $\mathbb{F}$. Since ω has a different multiplicity from the other eigenvalues of A, Fact 4 precludes the possibility that ω might have any algebraic conjugates over $\mathbb{F}$. Applying Fact 3, we see that the minimal polynomial of ω over $\mathbb{F}$ is $t \mapsto t - \omega$. Thus, ω belongs to $\mathbb{F}$. $\square$

We will also need to use cyclotomic polynomials; for an excellent introduction we refer to [61]. We recall he relevant definitions and facts.. The minimal polynomial of ζ_p is called the pth *cyclotomic polynomial*, and it is denoted by $\Phi_p(t)$. The roots of $\Phi_p(t)$ are precisely the primitive pth roots of unity. Note that $\Phi_1(t) = t - 1$.

Fact 7. *The cyclotomic polynomials satisfy the identity*

$$\prod_{a|p} \Phi_a(t) = t^p - 1 \tag{3.2}$$

where the symbol $a|p$ means that a divides p.

3.2 Real Equiangular Tight Frames

Assume that $1 < d < N$, and suppose that S is a $d \times N$ real ETF. In this section, we will see that the pair (d, N) must satisfy rigid integrality conditions. Denote the absolute inner product between columns by α, and recall from Proposition 1 that

$$\alpha = \sqrt{\frac{N-d}{d(N-1)}}. \tag{3.3}$$

Next, construct the *signature matrix* of the ETF:

$$A \quad = \quad \frac{1}{\alpha}\,(S^*S - I).$$

Since the inner products between columns of S have magnitude α, the off–diagonal entries of A are 1 or -1. Therefore, this matrix completely encodes the pattern of phases in the inner products. It is identical with the signature matrix considered by Holmes and Paulsen [36, Def. 3.1].

Our primary analysis is based on a detailed study of the eigenvalues of A using methods of field theory. Observe that A is Hermitian; it has a zero diagonal; and its off–diagonal entries have unit modulus. Since an ETF satisfies the equation $SS^* = (N/d)\, I$, it follows that the two eigenvalues of A are

$$\lambda_1 = -\frac{1}{\alpha} = -\sqrt{\frac{d(N-1)}{N-d}} \qquad \text{and} \qquad \lambda_2 = \frac{N-d}{d\,\alpha} = \sqrt{\frac{(N-1)(N-d)}{d}}$$

with respective multiplicities $N-d$ and d. The key idea in our proof is that λ_1 and λ_2 cannot take general real values because the entries of A are severely limited.

Following [36], we note that

$$\lambda_1 \lambda_2 = -(N-1)$$

and that A satisfies the quadratic matrix equation[1]

$$A^2 - (\lambda_1 + \lambda_2)A - (N-1)I = 0. \tag{3.4}$$

This point can be verified by a direct calculation.

For a real ETF, the off-diagonal entries of the signature matrix A equal ± 1. Although we are only interested in real ETFs, it is more natural to consider the case where the entries of the signature matrix are roots of unity. In this setting, the possible values of λ_1 and λ_2 are already quite special.

Theorem 3. *Assume that $1 < d < N-1$ and $N \neq 2d$. Suppose that S is a $d \times N$ ETF whose signature matrix A has entries in the ring $\mathbb{Z}[\zeta_p]$. Then the eigenvalues λ_1 and λ_2 of A both belong to $\mathbb{Z}[2\,\mathrm{Re}\,\zeta_p]$.*

[1]In fact the left hand side is the minimal polynomial of A.

Proof. Since $N \neq 2d$, the two eigenvalues of A have different multiplicities. The entries of A are algebraic integers in $\mathbb{Q}(\zeta_p)$, so Lemma 1 implies that λ_1 and λ_2 belong to the set of real algebraic integers in $\mathbb{Q}(\zeta_p)$. Facts 5 and 6 identify this set as $\mathbb{Z}[2\operatorname{Re}\zeta_p]$. □

As we have noted, a real ETF generates a matrix A whose off-diagonal entries equal ± 1. We may apply the previous theorem with $p = 1$ to obtain our first result for real ETFs.

Corollary 1. *Assume that $1 < d < N - 1$ and $N \neq 2d$. Suppose that S is a $d \times N$ real ETF. Then the eigenvalues λ_1 and λ_2 of the signature matrix A are ordinary integers.*

A simple consequence of this corollary is the weak integrality condition stated in the paper of Holmes and Paulsen [36, Thm. 3.3].

Corollary 2 (Holmes–Paulsen)**.** *If $d < N$ and a real $d \times N$ ETF exists, then*

$$(N - 2d)\sqrt{\frac{N-1}{d\,(N-d)}} \qquad \textit{is an integer.}$$

Proof. When $d = 1$, $d = N - 1$ or $N = 2d$, the result is obvious. Otherwise, we introduce the value of α from (3.3), and we find that $(\lambda_1 + \lambda_2)$ equals the stated expression. Since λ_1 and λ_2 are integers, the result follows instantly. □

Holmes and Paulsen established this condition for real ETFs by looking at the components of the matrix equation (3.4). They do not appear to recognize that λ_1 and λ_2 must in fact be integers. We note that in the technical report [54] we apply Theorem 3 to more general signature matrices.

In the next theorem we claim stricter conditions on λ_1 and λ_2.

Theorem 4. *Assume that $1 < d < N - 1$ and $N \neq 2d$. Suppose that S is a real $d \times N$ ETF. Then the eigenvalues λ_1 and λ_2 of the signature matrix are both odd integers.*

Our proof adapts an argument of P. M. Neumann quoted in [44]. The basic idea is to derive from the signature matrix A another integer matrix with known eigenvalues. Then we apply field–theoretic methods to see that these eigenvalues must lie in a discrete set.

Proof. Let us form a new matrix M whose entries all equal zero or one:

$$M = \tfrac{1}{2}\left(\mathbf{J} - \mathbf{I} - A\right)$$

where the symbol $\mathbf{J}$ denotes a conformal matrix of ones. We have ruled out the possibility that $N \leq d+1$, so the eigenvalue λ_1 of A has geometric multiplicity at least two. In consequence, the $(N-1)$-dimensional null space of $\mathbf{J}$ must intersect the invariant subspace of A associated with λ_1. Any vector in this intersection is an eigenvector of M with eigenvalue $\mu_1 = -\tfrac{1}{2}\left(1+\lambda_1\right)$. A similar argument establishes that $\mu_2 = -\tfrac{1}{2}\left(1+\lambda_2\right)$ is also an eigenvalue of M.

Corollary 1 establishes that λ_1 and λ_2 are integers, so the eigenvalues μ_1 and μ_2 must be rational numbers. The entries of M are integers, so Lemma 1 proves that the eigenvalues of M are also algebraic integers. We conclude that μ_1 and μ_2 are rational integers. That is, λ_1 and λ_2 are odd. $\square$

This result has another striking consequence.

Corollary 3. *If $1 < d < N-1$ and a real $d \times N$ ETF exists, then N is even.*

Proof. When $N \neq 2d$ this point follows immediately from Theorem 4 and the fact that $\lambda_1 \lambda_2 = -(N-1)$. $\square$

It was observed in [7,36,53] that real equiangular tight frames naturally give rise to regular two–graphs and vice versa. It is also known that regular two–graphs naturally give rise to strongly regular graphs with certain parameter sets [11, Ch. 4]. In consequence, there is also a natural connection between real ETFs and certain strongly regular graphs. We used the full power of this correspondence in [54] to prove:

Theorem 5. *If there exists a real $d \times 2d$ ETF, then d is odd and $(2d-1)$ is the sum of two squares.*

We end this section on real ETFs with an example demonstrating how to create a real equiangular tight frame from a suitable strongly regular graph, see also [54]. We consider the strongly regular graph with parameters: $(15, 6, 1, 3)$, depicted on Figure 3.1. We denote the incidence matrix of this graph by A. One can verify that this matrix satisfies $A^2 = 2A + 15I$. The eigenvalues are equal to $\lambda_1 = -3$ and $\lambda_2 = 5$, the latter with multiplicity $d = 6$. Finally, the matrix $G = -\frac{1}{3}A + I$, factors as $S^T S$ (e.g. by the spectral decomposition) providing 16 frame vectors of $\mathbb{R}^6$. Figure 3.2 shows the frame vectors as the row vectors for typesetting reasons.

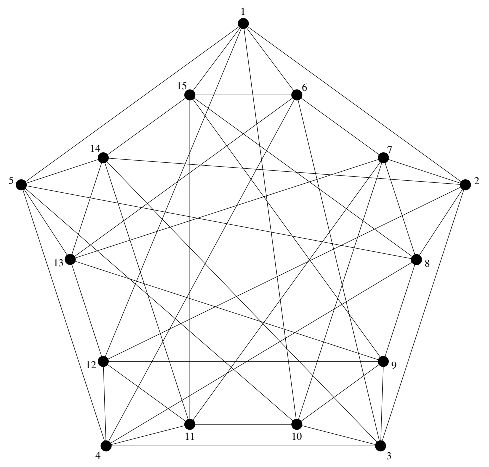


Figure 3.1: Strongly regular graph with parameters $(15, 6, 1, 3)$.

Figure 3.2: Frame matrix (transposed) generated from the strongly regular graph with parameters $(15, 6, 1, 3)$.

$$\begin{pmatrix}
-0.1436 & 0.0386 & 0.6623 & -0.4633 & -0.3333 & -0.4620 \\
-0.7718 & 0.4187 & 0.2162 & -0.2004 & 0.3333 & -0.1767 \\
0.6783 & 0.4346 & 0.0137 & -0.1800 & -0.3333 & -0.4554 \\
-0.6362 & -0.2930 & -0.2162 & -0.5925 & -0.3333 & -0.0235 \\
-0.1356 & 0.7116 & 0.4324 & 0.3921 & -0.3333 & -0.1528 \\
0.1464 & -0.5292 & -0.2162 & -0.1633 & -0.3333 & -0.7169 \\
0.6282 & -0.3801 & 0.4461 & -0.2630 & 0.3333 & -0.2857 \\
-0.6675 & -0.2522 & 0.2025 & 0.4089 & -0.3333 & -0.4143 \\
-0.1544 & -0.1439 & 0.4461 & -0.6921 & 0.3333 & 0.4077 \\
0.0108 & 0.1824 & 0.2162 & 0.2288 & 0.3333 & -0.8697 \\
0.4818 & 0.1491 & 0.6623 & -0.0997 & -0.3333 & 0.4312 \\
0.0421 & 0.1416 & -0.2025 & -0.7725 & 0.3333 & -0.4789 \\
-0.0501 & -0.8147 & 0.4324 & -0.0829 & -0.3333 & 0.1697 \\
-0.1043 & 0.6708 & 0.0137 & -0.6092 & -0.3333 & 0.2380 \\
-0.1857 & -0.1031 & 0.8648 & 0.3092 & 0.3333 & 0.0169 \\
0.0000 & 0.0000 & 0.0000 & 0.0000 & -1.0000 & 0.0000
\end{pmatrix}.$$

3.3 Complex Equiangular Tight Frames

We present one integrality condition on a special class of complex ETFs called *unital ETF*s. For these frames each entry of the scaled frame matrix $d^{1/2}S$ is a pth root of unity. The proof relies on well–known properties of the cyclotomic polynomial that we mentioned in Section 3.1.2

Theorem 6. *Fix $d > 1$, and suppose that there exists a $d \times N$ unital ETF of degree $p = q^s$, where q is a prime. Then q divides N.*

Proof. Suppose that $d^{-1/2}X$ is a $d \times N$ unital ETF of degree p. Let $\boldsymbol{x}^T$ and $\boldsymbol{y}^T$ be the first two rows of X. By ETF condition 3 in Definition 3, the two rows are orthogonal: $\langle \boldsymbol{x},\ \boldsymbol{y} \rangle = 0$. Since the entries of $\boldsymbol{x}$ and $\boldsymbol{y}$ are all powers of ζ_p, it follows that their inner product is a sum of N powers of ζ_p, not necessarily distinct. Therefore,

$$\langle \boldsymbol{x},\ \boldsymbol{y} \rangle = \sum_{k=0}^{p-1} c_k\, \zeta_p^k = 0$$

where $\{c_k\}$ is a set of nonnegative integers that sum to N.

Define the polynomial $u(t) = \sum_{k=0}^{p-1} c_k t^k$. We have established that $u(\zeta_p) = 0$ and that $u(1) = N$. It also follows that the minimal polynomial Φ_p of ζ_p divides the polynomial u. In particular, $\Phi_p(1)$ divides $u(1)$. We can complete the proof by showing that $\Phi_p(1) = q$ whenever $p = q^s$ for a prime number q and positive integer s.

We prove by induction on s that $\Phi_{q^s}(1) = q$. For the base case $s = 1$, we invoke Fact 7:

$$\Phi_1(t)\Phi_q(t) = t^q - 1 = (t-1)(t^{q-1} + t^{q-2} + \cdots + t + 1).$$

Since $\Phi_1(t) = t-1$, the polynomial $\Phi_q(t)$ must be equal to $t^{q-1}+t^{q-2}+\cdots+t+1$. Substitute $t = 1$ to complete the argument.

For the inductive step, assume the statement is true for each positive integer $j < s$. We invoke Fact 7 again to obtain

$$\prod_{j=0}^{s} \Phi_{q^j}(t) = t^{q^s} - 1 = (t-1)(t^{q^s-1} + t^{q^s-2} + \cdots + 1).$$

Cancel the factor $\Phi_1(t) = t - 1$ from both sides to arrive at the identity:

$$\prod_{j=1}^{s} \Phi_{q^j}(t) = t^{q^s-1} + t^{q^s-2} + \cdots + 1.$$

Substitute $t = 1$ to obtain $\prod_{j=1}^{s} \Phi_{q^j}(1) = q^s$. By the induction hypothesis, the left-hand side equals $q^{s-1}\Phi_{q^s}(1)$. We conclude that $\Phi_{q^s}(1) = q$ for each positive integer s. □

Chapter 4

Kernel Learning Application

Kernel learning is an important problem in machine learning. In mathematical terms, the goal is to find a matrix that satisfies the constraints extracted from previous observations and enable a classification, clustering or similar task to be carried out on new data. Fast image search [39] and information–theoretic metric learning [18] are recent applications benefiting from our work.

4.1 Background

We briefly introduce kernel learning and some technical tools we will need, namely, matrix divergences and Bregman's algorithm. For further details we refer to [41, 42].

4.1.1 Kernel Learning

Consider a set of training points $\mathbf{a}_1, ..., \mathbf{a}_n$, and transform the data using a non–linear function ψ. This mapping represents a transformation of the data to a higher–dimensional feature space. A kernel function is a function κ that gives the inner product between two vectors in the feature space:

$$\kappa(\mathbf{a}_i, \mathbf{a}_j) = \psi(\mathbf{a}_i) \cdot \psi(\mathbf{a}_j).$$

It is often possible to compute this inner product without explicitly computing the expensive mapping of the input points to the higher–dimensional feature space. Generally, given n points $\mathbf{a}_i$, we form an $n \times n$ matrix K, called the kernel matrix, whose (i, j) entry corresponds to $\kappa(\mathbf{a}_i, \mathbf{a}_j)$. In kernel–based algorithms, the only information needed about the input data points is the inner products; hence, the kernel matrix provides all relevant information for learning in the feature space. A kernel matrix is always positive semidefinite. See [51] for more details.

Despite the popularity of kernel methods in machine learning, many kernel–based algorithms scale poorly. To improve scalability, the use of low–rank kernel representations has been proposed. Given an $n \times n$ kernel matrix K, if the matrix is of low rank, say $r < n$, we can represent the kernel matrix in terms of a decomposition $K = GG^T$, with G an $n \times r$ matrix.

4.1.2 Matrix Divergences

To measure the nearness between two matrices, we will use Bregman matrix divergences, which are generalizations of Bregman vector divergences. Let φ be a real–valued strictly convex function defined over a convex set $S = \text{dom}(\varphi) \subseteq \mathbb{R}^m$ such that φ is differentiable on the relative interior of S. The

Bregman vector divergence [8] with respect to φ is defined as

$$D_{\varphi}(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x}) - \varphi(\boldsymbol{y}) - (\boldsymbol{x} - \boldsymbol{y})^T \nabla \varphi(\boldsymbol{y}).$$

For example, if $\varphi(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$, then the resulting Bregman divergence becomes $D_{\varphi}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2^2$. Another example is $\varphi(\boldsymbol{x}) = \sum_i (x_i \log x_i - x_i)$, where the resulting Bregman divergence is the (unnormalized) relative entropy $D_{\varphi}(\boldsymbol{x}, \boldsymbol{y}) = KL(\boldsymbol{x}, \boldsymbol{y}) = \sum_i (x_i \log \frac{x_i}{y_i} - x_i + y_i)$. Bregman divergences generalize many properties of squared loss and relative entropy. See [12] for more details.

We can naturally extend this definition to real, symmetric $n \times n$ matrices, denoted by S^n. Given a strictly convex, differentiable function $\phi : S^n \to \mathbb{R}$, the Bregman matrix divergence is defined to be

$$D_{\phi}(X, Y) = \phi(X) - \phi(Y) - \operatorname{tr}((\nabla \phi(Y))^T (X - Y)),$$

where $\operatorname{tr}(A)$ denotes the trace of matrix A. Examples include $\phi(X) = \|X\|_F^2$, which leads to the well–known squared Frobenius norm $\|X - Y\|_F^2$. In this paper, we will extensively study two less well–known divergences. Let ϕ be the entropy of the eigenvalues of a positive definite matrix. Specifically, if X has eigenvalues $\lambda_1, ..., \lambda_n$, let $\phi(X) = \sum_i (\lambda_i \log \lambda_i - \lambda_i)$, which may be expressed as $\phi(X) = \operatorname{tr}(X \log X - X)$, where $\log X$ is the matrix logarithm. The resulting Bregman divergence is

$$D_{vN}(X, Y) = \operatorname{tr}(X \log X - X \log Y - X + Y), \tag{4.1}$$

and we call it the von Neumann divergence. This divergence is also called quantum relative entropy, and is used in quantum information theory [48]. Another important matrix divergence arises by taking the Burg entropy of the eigenvalues, i.e. $\phi(X) = -\sum_i \log \lambda_i$, or equivalently as $\phi(X) = -\log\det X$. The resulting Bregman divergence over positive definite matrices is

$$D_{\ell d}(X, Y) = \mathrm{tr}(XY^{-1}) - \log\det(XY^{-1}) - n, \tag{4.2}$$

and is commonly called the LogDet divergence. For now, we assume that X is positive definite for both divergences; later we will discuss extensions when X is positive semidefinite, i.e., low–rank.

For all three divergences introduced above, the generating convex function of the Bregman matrix divergence can be viewed as a composition $\phi(X) = (\varphi \circ \lambda)(X)$, where $\lambda(X)$ is the function that lists the eigenvalues in algebraically decreasing order, and φ is a strictly convex function defined over vectors [23]. In general, every such φ defines a Bregman matrix divergence over symmetric matrices via the eigenvalue mapping. For example, if $\varphi(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$, then the resulting composition $(\varphi \circ \lambda)(X)$ is the squared Frobenius norm. We call such divergences *spectral Bregman matrix divergences.*

We prove the following lemma in [42, page 6] which provides an alternative expression for a spectral matrix divergence $D_\phi(X, Y)$ based on the eigendecompositions of X and Y. This will prove to be useful when motivating extensions to the low–rank case.

Lemma 2. *Let the eigendecompositions of X and Y be $V\Lambda V^T$ and $U\Theta U^T$ respectively, and assume that φ is* separable, *i.e.* $\phi(X) = (\varphi\circ\lambda)(X) = \sum_i \varphi_i(\lambda_i)$. *Then*

$$D_\phi(X,Y) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 (\varphi_i(\lambda_i) - \varphi_j(\theta_j) - (\lambda_i - \theta_j)\nabla\varphi_j(\theta_j)). \tag{4.3}$$

Note that each of the divergences discussed earlier—the squared Frobenius divergence, the LogDet divergence, and the von Neumann divergence—arise from separable convex functions. Furthermore, in these three cases, the functions φ_i do not depend on i (so we denote $\varphi = \varphi_1 = \ldots = \varphi_n$) and the corollary below follows:

Corollary 4. *Given $X = V\Lambda V^T$ and $Y = U\Theta U^T$, the squared Frobenius, von Neumann and LogDet divergences satisfy:*

$$D_\phi(X,Y) = \sum_{i,j} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 D_\varphi(\lambda_i, \theta_j). \tag{4.4}$$

4.1.3 Bregman's Algorithm

We now can give a formal description of the problem that we study in this chapter. Given an input kernel matrix X_0, we attempt to solve the following for X:

$$\begin{aligned} \text{minimize}_X \quad & D_\phi(X, X_0) \\ \text{subject to} \quad & \operatorname{tr}(XA_i) \le b_i,\ 1 \le i \le c, \\ & \operatorname{rank}(X) \le r, \\ & X \succeq 0. \end{aligned} \tag{4.5}$$

The objective function D_ϕ is either the von Neumann or the LogDet divergence. Any of the linear inequality constraints $\mathrm{tr}(XA_i) \le b_i$ may be replaced with equalities. This problem is *non–convex* in general, due to the rank constraint. However, when the rank of X_0 does not exceed r, then this problem turns out to be *convex* when we use the von Neumann and LogDet matrix divergences (we will extend the definition of the divergences to rank–deficient matrices in Section 4.2). Convex optimization problems are easier to solve, because local minimums are also global minimums.

A notable advantage of using the von Neumann and LogDet divergences (in contrast to the squared Frobenius divergence) is that the algorithms used to solve the minimization problem implicitly maintain the positive semidefiniteness constraint, eliminating extra work which would be necessary otherwise.

We find the minimum in (4.5) by the iterative method of Bregman projections [8,12]. The idea is to enforce only one constraint per iteration and acquire the optimum as a limit. We also call the single constraint nearness problem a projection step, referring to the minimization property of orthogonal projections in Euclidean geometry.

In the case of inequality constraints, an appropriate correction has to follow the projection step. The method may also be viewed as a dual coordinate ascent procedure that optimizes the dual with respect to a single dual variable per iteration (with all other dual variables remaining fixed). Details of convergence can be found in [12].

For an equality constraint of the form $\mathrm{tr}(XA_i) = b_i$, the Bregman projection of the current iterate X_t may be computed by solving:

$$\begin{aligned} \text{minimize}_X \quad & D_\phi(X, X_t) \\ \text{subject to} \quad & \mathrm{tr}(XA_i) = b_i. \end{aligned} \tag{4.6}$$

We introduce the dual variable α_i, and form the Lagrangian $\mathcal{L}(X, \alpha_i) = D_\phi(X, X_t) + \alpha_i(b_i - \mathrm{tr}(XA_i))$. By setting the gradient of the Lagrangian (with respect to X and α_i) to zero, we can obtain the Bregman projection by solving the resulting system of equations simultaneously for α_i and X_{t+1}:

$$\begin{aligned} \nabla\phi(X_{t+1}) &= \nabla\phi(X_t) + \alpha_i A_i \qquad (4.7) \\ \mathrm{tr}(X_{t+1}A_i) &= b_i. \end{aligned}$$

For an inequality constraint of the form $\mathrm{tr}(XA_i) \leq b_i$, let $\nu_i \geq 0$ be the corresponding dual variable. To maintain non–negativity of the dual variable (which is necessary for satisfying the KKT conditions), we can solve (4.7) for α_i and perform the following updates:

$$\alpha_i' = \min(\nu_i, \alpha_i), \quad \nu_i \leftarrow \nu_i - \alpha_i'. \tag{4.8}$$

Clearly the update guarantees that $\nu_i \geq 0$. Finally, update X_{t+1} via

$$\nabla\phi(X_{t+1}) = \nabla\phi(X_t) + \alpha_i' A_i. \tag{4.9}$$

The main difficulty in using Bregman projections lies in efficiently solving the nonlinear system of equations given in (4.7).

In the case where $A_i = \boldsymbol{z}_i\boldsymbol{z}_i^T$, by calculating the gradient for the LogDet and von Neumann matrix divergences, respectively, (4.7) simplifies to:

$$X_{t+1} = \left(X_t^{-1} - \alpha_i \boldsymbol{z}_i\boldsymbol{z}_i^T\right)^{-1} \tag{4.10}$$

$$X_{t+1} = \exp(\log(X_t) + \alpha_i \boldsymbol{z}_i\boldsymbol{z}_i^T), \tag{4.11}$$

subject to $\boldsymbol{z}_i^T X_{t+1}\boldsymbol{z}_i = b_i$. As we will see, for the von Neumann and LogDet divergences, these projections can be computed very efficiently.

4.2 Bregman Divergences for Rank–deficient Matrices

As given in (4.1) and (4.2), the von Neumann and LogDet divergences are seemingly undefined for low–rank matrices. For the LogDet divergence, the convex generating function $\phi(X) = -\log\det X$ is infinite when X is singular—in other words, its effective domain is the set of positive definite matrices. For the von Neumann divergence the situation is somewhat better, since one can define $\phi(X) = \operatorname{tr}(X\log X - X)$ via continuity for rank–deficient matrices.

The key to using these divergences in the low–rank setting comes from restricting the convex function ϕ to the range spaces of the matrices. We fully motivate this approach using Corollary 4 in [42, page 10] and arrive to the following observations:

Observation 1. *The von Neumann divergence $D_{vN}(X, Y)$ is finite if and only if* $\operatorname{range}(X) \subseteq \operatorname{range}(Y)$.

Observation 2. *The LogDet divergence $D_{\ell d}(X, Y)$ is finite if and only if* $\operatorname{range}(X) = \operatorname{range}(Y)$.

Assuming the eigenvalues of X and Y are in non–increasing order, we arrive to the low–rank equivalent of (4.4) simply as:

$$D_\phi(X,Y) = \sum_{i,j\leq r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 D_\varphi(\lambda_i, \theta_j).$$

If we now revisit our optimization problem formulated in (4.5), where we aim to minimize $D_\phi(X, X_0)$, we see that the LogDet and von Neumann divergences naturally maintain rank constraints. For the LogDet divergence, the equality of the range spaces of X and Y implies that the rank of X is equal to the rank of Y. Thus, when minimizing $D_\phi(X, X_0)$, we maintain $\text{rank}(X) = \text{rank}(X_0)$, assuming that there is a feasible X with a finite $D_\phi(X, X_0)$. Similarly, for the von Neumann divergence, the property that $\text{range}(X) \subseteq \text{range}(X_0)$ implies $\text{rank}(X) \leq \text{rank}(X_0)$.

The rigorous treatment of the generalization to low–rank matrices is found in [42, page 11]. Here we omit some of the proofs.

Let W be an orthogonal $n \times r$ matrix, such that its columns span the range space of Y. We need the following simple and well–known lemma:

Lemma 3. *Let Y be a symmetric $n \times n$ matrix with* $\text{rank}(Y) \leq r$*, and let W be an $n \times r$ column orthogonal matrix ($W^T W = I$) with* $\text{range}(Y) \subseteq \text{range}(W)$. *Then $Y = WW^TYWW^T$.*

We are now ready to extend the domain of the von Neumann and LogDet divergences to low–rank matrices.

Definition 4. *Consider the positive semidefinite $n \times n$ matrices X and Y such that* $\operatorname{range}(X) \subseteq \operatorname{range}(Y)$ *when considering the von Neumann divergence, and* $\operatorname{range}(X) = \operatorname{range}(Y)$ *when considering the LogDet divergence. Let an $n \times r$ column orthogonal matrix W satisfy range$(Y) \subseteq$ range(W) and define:*

$$D_\phi(X,Y) = D_{\phi,W}(X,Y) = D_\phi(W^T X W, W^T Y W), \tag{4.12}$$

where D_ϕ is either the von Neumann or the LogDet divergence.

The first equality in (4.12) implicitly assumes that the right hand side is not dependent on the choice of W, for a proof we refer to [42, page 11].

Lemma 4. *In Definition 4, $D_{\phi,W}(X,Y)$ is independent of the choice of W.*

We now show that definition 4 is consistent with Corollary 4, demonstrating that our low–rank formulation agrees with the informal discussion given earlier.

Theorem 7. *Let the positive semidefinite matrices X and Y have eigendecompositions $X = V\Lambda V^T$, $Y = U\Theta U^T$ and let* $\operatorname{range}(X) \subseteq \operatorname{range}(Y)$. *Let the rank of Y equal r. Assuming that the eigenvalues of X and Y are sorted in non–increasing order, i.e., $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_r$ and $\theta_1 \geq \theta_2 \geq ... \geq \theta_r$, then*

$$D_{vN}(X,Y) = \sum_{i,j \leq r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 (\lambda_i \log \lambda_i - \lambda_i \log \theta_j - \lambda_i + \theta_j).$$

Proof. Denote the upper left $r \times r$ submatrices of Λ and Θ by Λ_r and Θ_r respectively, and the corresponding reduced eigenvector matrices for X and Y

by V_r and U_r. Picking W in (4.12) to equal U_r, we get:

$$D_{vN}(X,Y) = D_{vN}(U_r^T X U_r, U_r^T Y U_r) = D_{vN}((U_r^T V_r)\Lambda_r(V_r^T U_r), \Theta_r).$$

The arguments on the right hand side are $r \times r$ matrices and Θ_r is not rank–deficient. We can now apply Corollary 4 to get the result. $\square$

Theorem 8. *Let the positive semidefinite matrices X and Y have eigendecompositions $X = V\Lambda V^T$, $Y = U\Theta U^T$, and let* range(X) = range(Y) *and assume that the eigenvalues of X and Y are sorted in decreasing order. Then:*

$$D_{\ell d}(X,Y) = \sum_{i,j \le r} (\boldsymbol{v}_i^T \boldsymbol{u}_j)^2 \left(\frac{\lambda_i}{\theta_j} - \log \frac{\lambda_i}{\theta_j} \right) - r.$$

Proof. Similar to the proof of Theorem 7. Note that the range spaces must coincide in this case, because the determinant of XY^{-1} should not vanish for the restricted transformations, which agrees with the range(X) = range(Y) restriction. $\square$

We next show that the optimization problem and Bregman's projection algorithm for low–rank matrices can be cast as a full rank problem in a lower dimensional space, namely the range space. This equivalence implies that we do not have to re–derive the convergence proofs and other properties of Bregman's algorithm in the low–rank setting.

Consider the optimization problem (4.5) for low–rank X_0, and denote a suitable orthogonal matrix as required in Definition 4 by W. The matrix of the eigenvectors of the reduced eigendecomposition of X_0 is a suitable choice. Consider the following matrix mapping:

$$M \longrightarrow \hat{M} = W^T M W.$$

By Lemma 3, the mapping is one–to–one on the set of symmetric matrices with range space contained in range(W). We now apply the mapping to all matrices in the optimization problem (4.5) to obtain:

$$\begin{aligned} \text{minimize} \quad & D_\phi(\hat{X}, \hat{X}_0) \\ \text{subject to} \quad & \operatorname{tr}(\hat{X}\hat{A}_i) \le b_i, \ 1 \le i \le c \\ & \operatorname{rank}(\hat{X}) \le r \\ & \hat{X} \succeq 0. \end{aligned} \tag{4.13}$$

The rank constraint is automatically satisfied when $\operatorname{rank}(X_0) = r$ and the problem is feasible. $\hat{X} \succeq 0$ if and only if $X \succeq 0$. By Definition 4, $D_\phi(\hat{X}, \hat{X}_0) = D_\phi(X, X_0)$. Finally, the next lemma verifies that the constraints are equivalent as well.

Lemma 5. *Given an orthogonal matrix W, such that it satisfies* range$(X) \subseteq$ range(W)*, it follows that* $\operatorname{tr}(\hat{X}\hat{A}_i) = \operatorname{tr}(XA_i)$*, where* $\hat{X} = W^T X W$*,* $\hat{A}_i = W^T A_i W$*.*

Proof. Let $\operatorname{rank}(X) = r$ and let the rank–r eigendecomposition of X be $V\Lambda V^T$. Since range$(X) \subseteq$ range(W), we may assume without the loss of generality that range(V) = range(W). We can write $W = VQ$ for some orthogonal Q. Then

$$\begin{aligned}\mathrm{tr}(\hat{X}\hat{A}_i) &= \mathrm{tr}((W^TXW)(W^TA_iW)) = \mathrm{tr}(Q^TV^TV\Lambda V^TVQQ^TV^TA_iVQ)\\ &= \mathrm{tr}(VQQ^T\Lambda QQ^TV^TA_i) = \mathrm{tr}(V\Lambda V^TA_i) = \mathrm{tr}(XA_i).\end{aligned}$$

□

If we assume that the optimization problem (4.5) has a (rank–deficient) solution with finite divergence measure, then the corresponding full–rank optimization problem (4.13) also has a solution. Conversely, by Lemma 3, for a solution $\hat{X}$ of (4.13), there is the unique solution of (4.5) $X = W\hat{X}W^T$ (with finite $D_\phi(X, X_0)$) that satisfies the range space restriction.

4.3 The von Neumann Matrix Update Algorithm

In this section we derive the explicit matrix updates for Bregman's algorithm in the low–rank setting. Recall (4.7), which we used to calculate the projection update and apply it to the mapped problem (4.13):

$$\begin{aligned}\nabla\phi(\hat{X}_{t+1}) &= \nabla\phi(\hat{X}_t) + \alpha\hat{A}_i\\ \mathrm{tr}(\hat{X}_{t+1}\hat{A}_i) &= b_i.\end{aligned}$$

In case of the von Neumann divergence this leads to the update $\hat{X}_{t+1} = \exp(\log(\hat{X}_t + \alpha\hat{A}_i)$. The arguments of Section 4.2 and induction on t implies that $X_{t+1} = W\hat{X}_{t+1}W^T$ (with W as in Definition 4), or explicitly:

$$X_{t+1} = W\exp(\log(W^TX_tW) + \alpha(W^TA_iW))W^T.$$

If we choose $W = V_t$ from the reduced eigendecomposition $X_t = V_t \Lambda_t V_t^T$, then the update is written as:

$$X_{t+1} = V_t \exp(\log(\Lambda_t) + \alpha V_t^T A_i V_t) V_t^T, \tag{4.14}$$

which we will use in the von Neumann kernel learning algorithm. Note that the limit of $\exp(\log(X_t + \epsilon I) + \alpha A_i)$ as ϵ approaches zero yields the same formula, which becomes clear if we apply a basis transformation that puts X_t in diagonal form.

The matrix A_i has often a quite special structure; we concentrate on the case when it is of rank–one: $A_i = \boldsymbol{z}_i^T \boldsymbol{z}_i$. The so–called distance constraints are a special case with $\boldsymbol{z}_i = \boldsymbol{e}_j - \boldsymbol{e}_k$.

We take the eigendecomposition of the exponent:

$$\log(\Lambda_t) + \alpha V_t^T \boldsymbol{z}_i \boldsymbol{z}_i^T V_t = U_t \Theta_{t+1} U_t^T,$$

and calculate the eigendecomposition of X_{t+1} by $V_{t+1} = V_t U_t$ and $\Lambda_{t+1} = \exp(\Theta_{t+1})$. This special eigenvalue problem (diagonal plus rank–one update) can be solved in $O(r^2)$ time; see [28], [31] and [20]. This means that the matrix multiplication $V_{t+1} = V_t U_t$ becomes the most expensive step in the computation, yielding $O(nr^2)$ complexity.

We reduce this cost by modifying the decomposition slightly. We let $X_t = V_t W_t \Lambda_t W_t^T V_t^T$ be the factorization of X_t, where W_t is an $r \times r$ orthogonal matrix, while V_t and Λ_t are defined as before. The matrix update becomes

$$X_{t+1} = V_t W_t \exp(\log \Lambda_t + \alpha W_t^T V_t^T \boldsymbol{z}_i \boldsymbol{z}_i^T V_t W_t) W_t^T V_t^T,$$

yielding the following formulas:

$$V_{t+1} = V_t,\ W_{t+1} = W_t U_t,\ \Lambda_{t+1} = \exp(\Theta_{t+1}),$$

where $\log \Lambda_t + \alpha W_t^T V_t^T \boldsymbol{z}_i \boldsymbol{z}_i^T V_t W_t = U_t \Theta_{t+1} U_t^T$. For a general rank–one constraint the product $V_t^T \boldsymbol{z}_i$ is calculated in $O(nr)$ time, but for distance constraints $O(r)$ operations are sufficient. The calculation of $W_t^T V_t^T \boldsymbol{z}_i$ and computing the eigendecomposition $U_t \Theta_{t+1} U_t^T$ will both take $O(r^2)$ time. The matrix product $W_t U_t$ appears to cost $O(r^3)$ time, but in fact a right multiplication by U_t^T can be approximated very accurately in $O(r^2 \log r)$ and even in $O(r^2)$ time using the fast multipole method—see [3] and [30].

Since we repeat the above update calculations until convergence, we can avoid calculating the logarithm of Λ_t at every step by maintaining $\Theta_t = \log \Lambda_t$ throughout the algorithm.

The above calculations assumed a known value for the projection parameter α. In reality, we have to determine this value such that $\operatorname{tr}(X_{t+1} A_i) = b_i$ holds. There is no closed formula available and therefore we find α as the zero of the function $f(\alpha) = \operatorname{tr}(X_{t+1} A_i) - b_i$ by using an iterative method (for example the secant method).

The algorithm for distance inequality constraints using the von Neumann divergence is given as Algorithm 4.

Algorithm 4: Learning a low–rank kernel matrix in von Neumann matrix divergence under distance constraints.

```
   Input : r: rank of desired kernel matrix, $\{A_i\}_{i=1}^c$: constraints,
           $V_0, \Lambda_0$: input kernel in factored form
1  Set $W = I_r$, $\Theta = \log \Lambda_0$, $i = 1$, and $\nu_j = 0\ \ \forall$ constraints $j$.
2  repeat
3      $\boldsymbol{v}^T = V_0(i_1, :) - V_0(i_2, :)$
4      $\boldsymbol{w} = W^T \boldsymbol{v}$
5      Call ZeroFinder$(f, b_i)$ to determine $\alpha$.
6      $\beta = \min(\nu_i, \alpha)$
7      $\nu_i \leftarrow \nu_i - \beta$.
8      Factor $\Theta + \beta \boldsymbol{w}\boldsymbol{w}^T = U \Theta U^T$.
9      $W \leftarrow WU$
10     $i = \mod(i+1, c)$
11 until convergence of $\nu$
12 $V = V_0 W \exp(\Theta/2)$
```

4.4 The LogDet Matrix Update Algorithm

In this section we derive the explicit matrix updates for Bregman's algorithm in the low–rank setting. Recall (4.7), which we used to calculate the projection update and apply it to the mapped problem (4.13):

$$\begin{aligned} \nabla\phi(\hat{X}_{t+1}) &= \nabla\phi(\hat{X}_t) + \alpha \hat{A}_i \\ \operatorname{tr}(\hat{X}_{t+1}\hat{A}_i) &= b_i. \end{aligned}$$

In case of the LogDet divergence this leads to the matrix update $\hat{X}_{t+1} = ((\hat{X}_t^{-1} - \alpha \hat{A}_i)^{-1}$. The arguments of Section 4.2 and induction on t implies that $X_{t+1} = W\hat{X}_{t+1}W^T$. (with W as in Definition 4), or explicitly:

$$X_{t+1} = W((W^T X_t W)^{-1} - \alpha(W^T A_i W))^{-1} W^T, \tag{4.15}$$

If we choose $W = V_t$ from the reduced eigendecomposition $X_t = V_t \Lambda_t V_t^T$, then the update is written as:

$$X_{t+1} = V_t((V_t^T X_t V_t)^{-1} - \alpha(V_t^T A_i V_t))^{-1} V_t^T, \tag{4.16}$$

using Lemma 3 and the fact that $\mathrm{range}(X_{t+1}) = \mathrm{range}(X_t)$. The right hand side of (4.16) can be calculated without the eigendecomposition and a closed formula exists for the projection parameter. Note that the projection parameter α had to be calculated by iterative means for the von Neumann divergence.

The matrix A_i has often a quite special structure; we concentrate on the case when it has rank–one: $A_i = \boldsymbol{z}_i^T \boldsymbol{z}_i$. The so–called distance constraints are a special case with $\boldsymbol{z}_i = \boldsymbol{e}_j - \boldsymbol{e}_k$. Equation 4.16 with a rank–one constraint calculates the inverse of a rank–one update, suggesting the application of the Sherman-Morrison inverse formula. Subsequent rewriting and simplification leads to the following equation for α (for details see [42, page 14]);

$$\boldsymbol{z}_i^T \left(X_t + \frac{\alpha X_t \boldsymbol{z}_i \boldsymbol{z}_i^T X_t}{1 - \alpha \boldsymbol{z}_i^T X_t \boldsymbol{z}_i} \right) \boldsymbol{z}_i = b_i. \tag{4.17}$$

Let $p = \boldsymbol{z}_i^T X_t \boldsymbol{z}_i$. Note that in the case of distance constraints, p is the distance between the two data points corresponding to constraint i. When $p \neq 0$, elementary arguments reveal that there is exactly one solution for α provided that $b_i \neq 0$. The unique solution, in this case, can be expressed as:

$$\alpha = \frac{1}{p} - \frac{1}{b_i}. \tag{4.18}$$

If we let

$$\beta = \alpha/(1 - \alpha p), \tag{4.19}$$

then our matrix update is given by

$$X_{t+1} = X_t + \beta X_t \boldsymbol{z}_i \boldsymbol{z}_i^T X_t. \tag{4.20}$$

When $p = 0$, (4.17) has a solution if and only if $b_i = 0$ since (4.17) implies that $b_i = p/(1 - \alpha p)$.

The following lemma confirms the expectation that we remain in the positive semidefinite cone and that the range space is unchanged; for the proof refer to [42, page 15].

Lemma 6. *Given a positive semidefinite matrix X_t, the matrix X_{t+1} from the update in (4.20) is positive semidefinite with* $\operatorname{range}(X_{t+1}) = \operatorname{range}(X_t)$*, assuming that (4.6) is feasible.*

A naive implementation of the update given in (4.20) costs $O(n^2)$ per iteration. However, we can achieve a more efficient update for low–rank matrices by working on a suitable factored form of the matrix X_t resulting in an $O(r^2)$ algorithm. Both the reduced eigendecomposition and the Cholesky factorization are possible candidates for the factorization; we prefer the latter because the resulting algorithm does not have to rely on iterative methods.

The positive semidefinite rank–r matrix X_t can be factored as GG^T, where G is an $n \times r$ matrix, and thus the update can be written as:

$$X_{t+1} = G(I + \beta G^T \boldsymbol{z}_i \boldsymbol{z}_i^T G)G^T.$$

The matrix $I + \beta \tilde{\boldsymbol{z}}_i \tilde{\boldsymbol{z}}_i^T$, where $\tilde{\boldsymbol{z}}_i = G^T \boldsymbol{z}_i$, is an $r \times r$ matrix. To update G for the next iteration, we factor this matrix as LL^T; then our new G is updated

Algorithm 5: Learning a low–rank kernel matrix in LogDet divergence under distance constraints.

```
Input  : $r$: rank of desired kernel matrix, $\{A_i\}_{i=1}^{c}$: constraints,
         $G_0$: input kernel factor matrix
Output: $G$: output low–rank kernel factor matrix
1  Set $B = I_r$, $i = 1$, and $\nu_j = 0 \;\; \forall$ constraints $j$.
2  repeat
3      $\boldsymbol{v}^T = G_0(i_1, :) - G_0(i_2, :)$
4      $\boldsymbol{w} = B^T \boldsymbol{v}$
5      $\alpha = \min\left(\nu_i, \frac{1}{\|\boldsymbol{w}\|_2^2} - \frac{1}{b_i}\right)$
6      $\nu_i \leftarrow \nu_i - \alpha$
7      $\beta = \alpha / (1 - \alpha \|\boldsymbol{w}\|_2^2)$
8      Call CholUpdateMult($\beta, \boldsymbol{w}, B$) to factor $I + \beta \boldsymbol{w}\boldsymbol{w}^T = LL^T$
       and update $B \leftarrow BL$.
9      Set $i \leftarrow \mod (i + 1, c)$.
10 until convergence of $\nu$
11 $G = G_0 B$
```

to GL. Since $I + \beta \tilde{\boldsymbol{z}}_i \tilde{\boldsymbol{z}}_i^T$ is a rank–one perturbation of the identity, this update can be done in $O(r^2)$ time using a standard Cholesky rank–one update routine.

To increase computational efficiency, we note that $G = G_0 B$, where B is the product of all the L matrices from every iteration and G_0 is the initial factor of X_0. Instead of updating G explicitly at each iteration, we simply update B to BL. The matrix $I + \beta G^T \boldsymbol{z}_i \boldsymbol{z}_i^T G$ is then $I + \beta B^T G_0^T \boldsymbol{z}_i \boldsymbol{z}_i^T G_0 B$. In the case of distance constraints, we can compute $G_0^T \boldsymbol{z}_i$ in $O(r)$ time as the difference of two rows of G_0. The multiplication update BL appears to have $O(r^3)$ complexity, dominating the run time. The kernel learning algorithm utilizing the matrix updates is presented as Algorithm 5.

In the next section we derive an algorithm that combines the Cholesky

Algorithm 6: CholUpdateMult(α, x, B). Right multiplication of a lower triangular $r \times r$ matrix B with the Cholesky factor of $I + \alpha \boldsymbol{x}\boldsymbol{x}^T$ in $O(r^2)$ time.

```
    Input  : $\alpha, \boldsymbol{x}, B$, with $I + \alpha \boldsymbol{x}\boldsymbol{x}^T \succeq 0$, $B$ is lower triangular
    Output: $B \leftarrow BL$, with $LL^T = I + \alpha \boldsymbol{x}\boldsymbol{x}^T$
 1  $\alpha_1 = \alpha$
 2  for $i = 1$ to $r$ do
 3      $t = 1 + \alpha_i x_i^2$
 4      $h_i = \sqrt{t}$
 5      $\alpha_{i+1} = \alpha_i / t$
 6      $t = B_{ii}$
 7      $s = 0$
 8      $B_{ii} = B_{ii} h_i$
 9      for $j = i - 1$ to $1$ do
10          $s = s + t x_{j+1}$
11          $t = B_{ij}$
12          $B_{ij} = (B_{ij} + \alpha_{j+1} x_j s) h_j$
13      end
14  end
```

factorization and the matrix multiplication with this factor into a single $O(r^2)$ routine and reducing the complexity of Algorithm 5 to $O(r^2)$.

4.5 Fast Multiplication with a Cholesky Update Factor

We efficiently combine the Cholesky factorization with the matrix multiplication in CholUpdateMult, as given by Algorithm 6. A simple analysis of this algorithm reveals that it requires $3r^2 + 2r$ floating point operations (flops). Thus, we can perform the Cholesky factorization followed by matrix multiplication in $O(r^2)$ time, as opposed to the usual $O(r^3)$ time needed by matrix multiplication. We devote this section to the development of this fast multiplication algorithm.

Recall the algorithm used for the Cholesky factorization of an $r \times r$ matrix A (see [20], page 78):

```
1 for $j = 1$ to $r$ do
2     $l_{jj} = (a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2)^{1/2}$
3     for $i = j + 1$ to $r$ do
4         $l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj}$
5     end
6 end
```

We will derive a corresponding algorithm in a manner similar to [20], while exploiting the special structure present in our problem. Let us denote the $I_r + \alpha_1 \boldsymbol{x}\boldsymbol{x}^T$ matrix by A ($\alpha_1 = \alpha$) and write it as a product of three block matrices:

$$A = \begin{bmatrix} \sqrt{1+\alpha_1 x_1^2} & 0 \\ \frac{\alpha_1 x_1}{\sqrt{1+\alpha_1 x_1^2}} \boldsymbol{x}_{2:r} & I_{r-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{A}_{22} \end{bmatrix} \begin{bmatrix} \sqrt{1+\alpha_1 x_1^2} & \frac{\alpha_1 x_1}{\sqrt{1+\alpha_1 x_1^2}} \boldsymbol{x}_{2:r}^T \\ 0 & I_{r-1} \end{bmatrix}.$$

It follows that $\tilde{A}_{22} + \frac{\alpha_1^2 x_1^2}{1+\alpha_1 x_1^2} \boldsymbol{x}_{2:r} \boldsymbol{x}_{2:r}^T = A_{22}$. Substitution of A_{22} by $I_{r-1} + \alpha_1 \boldsymbol{x}_{2:r} \boldsymbol{x}_{2:r}^T$ and simplification leads to:

$$\tilde{A}_{22} = I_{r-1} + \frac{\alpha_1}{1+\alpha_1 x_1^2} \boldsymbol{x}_{2:r} \boldsymbol{x}_{2:r}^T.$$

Introduce $\alpha_2 = \frac{\alpha_1}{1+\alpha_1 x_1^2}$ and proceed by induction. We extract the following algorithm which calculates L satisfying $LL^T = A$:

```
for $j = 1$ to $r$ do
    $t = 1 + \alpha_j x_j^2$
    $l_{jj} = \sqrt{t}$
    $\alpha_{j+1} = \alpha_j / t$
    $t = \alpha_j x_j / l_{jj}$
    for $i = j+1$ to $r$ do
        $l_{ij} = t x_i$
    end
end
```

The above algorithm uses $\frac{1}{2}r^2 + \frac{13}{2}r$ flops to calculate L, while $\frac{1}{2}r^3 + O(r^2)$ are needed for the general algorithm. However, we do not necessarily have to calculate L explicitly, since the parameters α_i together with $\boldsymbol{x}$ implicitly determine L. Notice that the cost of calculating $\alpha_1, \alpha_2, \ldots, \alpha_r$ is linear in r.

Next we show how to calculate $\boldsymbol{u}^T L$ for a given vector $\boldsymbol{u}$ without explicitly calculating L and arrive at an $O(r)$ algorithm for this vector–matrix multiplication. The coordinates of $\boldsymbol{v}^T = \boldsymbol{u}^T L$ are equal to:

$$\begin{aligned}
v_1 &= u_1\sqrt{1+\alpha_1 x_1^2} + \frac{\alpha_2}{\sqrt{1+\alpha_1 x_1^2}} x_1(u_2 x_2 + u_3 x_3 + \ldots u_r x_r) \\
v_2 &= u_2\sqrt{1+\alpha_2 x_2^2} + \frac{\alpha_3}{\sqrt{1+\alpha_2 x_2^2}} x_2(u_3 x_3 + \ldots u_r x_r) \\
&\vdots \\
v_{r-1} &= u_{r-1}\sqrt{1+\alpha_{r-1} x_{r-1}^2} + \frac{\alpha_r}{\sqrt{1+\alpha_{r-1} x_{r-1}^2}} x_{r-1} u_r x_r \\
v_r &= u_r\sqrt{1+\alpha_r x_r^2}.
\end{aligned}$$

We can avoid the recalculation of some intermediate results if we evaluate v_r first, followed by v_{r-1} down to v_1. This strategy leads to the following algorithm:

```
$v_r = u_r\sqrt{1+\alpha_r x_r^2}$
$s = 0$
for $j = r-1$ to $1$ do
    $s = s + u_{j+1}x_{j+1}$
    $t = \sqrt{1+\alpha_j x_j^2}$
    $v_j = u_j t + \alpha_j x_j s/t$
end
```

Exactly $11r-5$ flops are required by the above vector–matrix multiplication algorithm, and therefore we can readily multiply an $r \times r$ matrix by L using $11r^2 - 5r$ flops. Even fewer flops are sufficient to implement the matrix multiplication if we observe that the square root expression above is repeatedly calculated for each row, since it depends only on x_j and α_j. Additionally, when multiplying with a lower triangular matrix, the presence of zeros allows further simplifications. Taking these considerations into account we arrive at the previously presented Algorithm 6, which uses exactly $3r^2 + 2r$ flops.

Chapter 5

A Zero-Finding Involving the Matrix Exponential

5.1 Background and motivation

In the machine learning application of Chapter 4, see also [38, 41, 42], a matrix nearness problem depends on finding the zero of the function

$$f(\alpha) = \boldsymbol{z}^T e^{\log X + \alpha \boldsymbol{z}\boldsymbol{z}^T} \boldsymbol{z} - b \tag{5.1}$$

where the $n \times n$ symmetric positive definite matrix X, vector $\boldsymbol{z}$ and the scalar $b > 0$ are given parameters; the exponentiation and logarithm used are matrix functions. The zero-finding computation arises during the construction of a positive definite matrix that satisfies linear constraints while minimizing a distance measure called the von Neumann matrix divergence, see also [42]. In the machine learning application the constraints are extracted from observations, and the constructed positive definite matrix is used to carry out data analysis tasks such as clustering, classification or nearest neighbor search [18, 39]. In another application, one aims to find the nearest correlation matrix (positive semidefinite matrix with diagonal elements equal to one) to a given initial matrix. In [33], the nearness is measured using the Frobenius norm; however, other measures, such as the von Neumann matrix divergence, are also feasible [23].

The underlying matrix nearness problem can be solved by Bregman's iterative algorithm which consists of matrix updates that depend on finding the zero of $f(\alpha)$. In this section we present an efficient zero-finding algorithm that exploits the structure of the function. If the cost of evaluating the derivative is similar to the cost of evaluating the function itself, inverse quadratic interpolation (which needs no derivative computations) is expected to be faster than Newton's method, see [40] and [49][p. 55]. In our problem, the evaluation of $f'(\alpha)$, once $f(\alpha)$ has been computed, costs less than the computation of $f(\alpha)$ alone and therefore the cost of the derivative computations is offset by the faster convergence of Newton's method.

The lack of commutativity of matrix multiplication makes the derivative computation non-trivial. Our algorithm operates on the eigendecomposition of the matrix and arranges the computations of $f(\alpha)$ and $f'(\alpha)$ efficiently. We also take advantage of the not widely used improvement to Newton's method described in [40]. In numerical experiments we compare our algorithm to zero-finders which do not need computation of the derivative.

5.1.1 Derivative of the Matrix Exponential

The formula for the derivative of the matrix exponential is not as simple as that for the exponential function defined on the reals. The difficulty stems from the non-commutativity of matrix multiplication. We start with some basic properties of the matrix derivative and then review the formula for the derivative of the matrix exponential.

We consider smooth matrix functions of one variable denoted by $M(x) : \mathbb{R} \to \mathbb{R}^{n\times n}$; these can also be thought of as $\mathbb{R} \to \mathbb{R}$ functions arranged in an $n\times n$ matrix. The derivative matrix $M'(x)$ is formed by taking the derivatives of the matrix elements. Our first observation is about the trace of the derivative. By definition:

$$\operatorname{tr}(M(x))' = \operatorname{tr}(M'(x)). \tag{5.2}$$

We turn to multiplication next. The lack of commutativity does not yet indicate any difficulties:

$$(M(x)N(x))' = M'(x)N(x) + M(x)N'(x). \tag{5.3}$$

We are seeking $\operatorname{tr}\left(e^{M(\alpha)}A\right)'$ as the function $f(\alpha)$ defined in (5.1) is of this form with $M(\alpha) = \log X + \alpha \boldsymbol{z}\boldsymbol{z}^T$ and $A = \boldsymbol{z}\boldsymbol{z}^T$. But in order to demonstrate the issues caused by non-commutativity we take a short diversion by looking at the slightly simpler example of $\operatorname{tr}(e^M)'$. From here on, when there is no chance of confusion, we may omit the variable from our formulae.

We can express the matrix derivative of the kth power as follows: $(M^k)' = \sum_{i=0}^{k-1} M^i M' M^{k-1-i}$. Note that the summation cannot be collapsed when M and M' do not commute. However, if we take the trace on both sides then the summation can be collapsed since $\operatorname{tr}(AB) = \operatorname{tr}(BA)$ and $\operatorname{tr}(\sum_i A_i) = \sum_i \operatorname{tr}(A_i)$ (the latter also holds for infinite sums when one of the sides converges):

$$\operatorname{tr}\left(M^k\right)' = k \operatorname{tr}\left(M^{k-1}M'\right). \tag{5.4}$$

By (5.4) and the power series expansion of the exponential function $\mathrm{tr}(e^M)'$ we get:

$$\begin{aligned}\mathrm{tr}\left(e^M\right)' &= \mathrm{tr}\left(\sum_{k=0}^{\infty}\frac{M^k}{k!}\right)' = \sum_{k=0}^{\infty}\frac{\mathrm{tr}\left(M^k\right)'}{k!} \\ &= \sum_{k=1}^{\infty}\frac{\mathrm{tr}\left(M^{k-1}M'\right)}{(k-1)!} = \mathrm{tr}\left(e^M M'\right).\end{aligned}$$

The above argument does not imply that the derivative of e^M equals to $e^M M'$ and it also does not readily extend to $\mathrm{tr}\left(e^{M(\alpha)}A\right)'$. In order to tackle this latter expression, we apply (5.2) and (5.3) to get $\mathrm{tr}(e^{M(\alpha)}A)' = \mathrm{tr}((e^{M(\alpha)})'A)$ and then we use the formula for $(e^M)'$ from [50][p. 15, Theorem 5]:

$$(e^M)' = e^M h(\mathrm{ad}_M)M', \tag{5.5}$$

where the *commutator* operator $\mathrm{ad}_A : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ satisfies $\mathrm{ad}_A B = AB - BA$, and

$$h(t) = \begin{cases} \frac{1-e^{-t}}{t}, & t \neq 0 \\ 1 & t = 0. \end{cases} \tag{5.6}$$

The analytical function h can be extended to act on linear operators (transformations) via its Taylor series and by the Jordan canonical form; for a detailed treatment we refer the reader to [34][Chapter 1, Definition 1.2][1]. The extension applied to the operator ad_M maps matrices to matrices and appears on

[1]The space of linear transformations over an n-dimensional vector space can be identified with, and therefore is equivalent to the space of $n \times n$ matrices denoted by M_n. A linear operator, like ad, that acts on M_n can be represented by an $n^2 \times n^2$ matrix, because the underlying linear space, M_n has dimension n^2.

the right hand side of (5.5) operating on M'. The Taylor expansion of $h(t)$ around 0 is:

$$h(t) = 1 - \frac{t}{2!} + \frac{t^2}{3!} - \frac{t^3}{4!} + \ldots = \sum_{i=0}^{+\infty} \frac{(-t)^i}{(i+1)!},$$

so one may write (5.5) in a more verbose way as:

$$(e^M)' = e^M \sum_{i=0}^{+\infty} \frac{1}{(i+1)!} (-\operatorname{ad}_M)^i M'.$$

5.2 Algorithms

We propose to solve $f(\alpha) = 0$ using Newton's method and the method described by Jarratt in [40]. The latter zero-finder uses a rational interpolating function of the form

$$y = \frac{x - a}{bx^2 + cx + d} \tag{5.7}$$

fitted to the function and derivative values from *two* previous iterations. For completeness, we outline Jarratt's method in Algorithm 7. When the cost of the interpolation itself is negligible, Jarrat's method needs the same computational work as Newton's method, but it yields faster convergence. Despite this fact, this zero-finder has not gained sufficient attention. The *(asymptotic) efficiency index*[2] in the sense of Ostrowski [49][Chapter 3, Section 11] is $\sqrt{1+\sqrt{3}} \approx 1.653$, if we assume that the computational cost to evaluate $f(\alpha)$ and $f'(\alpha)$ are the same. The efficiency index for Newton's method under the same assumption is only $\sqrt{2} \approx 1.414$. In comparison, inverse quadratic

[2]A similar concept is the *order of convergence per function evaluation.*

Algorithm 7: Zero-finding based on P. Jarratt's method, see [40].

```
  Input  : Subroutines to evaluate $f$ and $f'$, initial guess $\alpha_0$.
  Output: Sequence of approximation to the solution of $f(\alpha) = 0$.
1 Compute $f_0 = f(\alpha_0)$, $f'_0 = f'(\alpha_0)$.
2 $\alpha_1 = \alpha_0 - f_0/f'_0$. (Initial Newton step.)
3 for $i = 2, 3, \ldots$ do
4     Compute $f_{i-1} = f(\alpha_{i-1})$ and $f'_{i-1} = f'(\alpha_{i-1})$.
5     Set $\alpha_i = \alpha_{i-1} - \frac{(\alpha_{i-1}-\alpha_{i-2})f_{i-1}[f_{i-2}(f_{i-1}-f_{i-2})-(\alpha_{i-1}-\alpha_{i-2})f_{i-1}f'_{i-2}]}{2f_{i-1}f_{i-2}(f_{i-1}-f_{i-2})-(\alpha_{i-1}-\alpha_{i-2})(f_{i-1}^2 f'_{i-2}+f_{i-2}^2 f'_{i-1})}$.
6 end
```

interpolation, which is the workhorse of Brent's method [9] requires no derivative computations and has asymptotic efficiency index of 1.839. Newton's and Jarratt's method can perform better when the derivative computation costs less than the function evaluation and this is often the case when the objective function is built from exp, sin, cos, see also [40]. In such circumstances, the efficiency index for Newton's and Jarratt's methods may approach the order of convergence, 2 and $1 + \sqrt{3} \approx 2.732$ respectively.

We show how to efficiently carry out and arrange the computations of $f(\alpha)$ and $f'(\alpha)$ in Section 5.3. An additional improvement exploiting the shape of the objective function is discussed in Section 5.4. We end this section by a lemma that establishes that f is strictly monotone, which implies that $f(\alpha) = 0$ has a unique solution. The proof is very similar to Lemma 7 of [1]; the fact that $\boldsymbol{z}\boldsymbol{z}^T$ has rank one allows some simplifications. We also establish convexity.

Lemma 7. *If M is symmetric and $\boldsymbol{z} \neq 0$ then $f(\alpha) + b = \boldsymbol{z}^T e^{M+\alpha \boldsymbol{z}\boldsymbol{z}^T} \boldsymbol{z}$ is* *strictly monotone increasing and strictly convex.*

Proof. First, we note that it is sufficient to show that the first and second derivatives are positive at any given α_0. Consider the function $\overline{f}(\alpha) = f(\alpha + \alpha_0)$, a shift by α_0. Since $\overline{f}(\alpha) = \boldsymbol{z}^T e^{\overline{M}+\alpha \boldsymbol{z}\boldsymbol{z}^T}\boldsymbol{z} - b$ where $\overline{M} = M + \alpha_0 \boldsymbol{z}\boldsymbol{z}^T$ is also a symmetric matrix, we can conclude that it is sufficient to prove that the first and second derivatives are positive at $\alpha = 0$.

Second, we show that we can assume that M is positive definite. Otherwise, pick a β that is large enough so that $\widehat{M} = M + \beta I$ is positive definite. Since $e^{M+\alpha \boldsymbol{z}\boldsymbol{z}^T} = e^{-\beta} e^{\widehat{M}+\alpha \boldsymbol{z}\boldsymbol{z}^T}$, we conclude that the sign of the derivatives is the same for $\widehat{M}$ and M.

In order to establish the claim in the case of a positive definite M and $\alpha = 0$, we inspect the coefficients in the power series expansion of $\boldsymbol{z}^T e^{M+\alpha \boldsymbol{z}\boldsymbol{z}^T}\boldsymbol{z}$ around zero. We note that f is analytical, which can be seen by bounding the terms of the expansion. According to the power series expansion of exp we have:

$$\begin{aligned}
\boldsymbol{z}^T e^{M+\alpha \boldsymbol{z}\boldsymbol{z}^T}\boldsymbol{z} &= \boldsymbol{z}^T \sum_{k=0}^{\infty} \frac{(M+\alpha \boldsymbol{z}\boldsymbol{z}^T)^k}{k!}\boldsymbol{z} \\
&= \sum_{k=0}^{\infty} \frac{\boldsymbol{z}^T M^k \boldsymbol{z}}{k!} + \alpha \sum_{k=1}^{\infty} \frac{1}{k!} \sum_{i=0}^{k-1} \boldsymbol{z}^T M^i \boldsymbol{z}\boldsymbol{z}^T M^{k-1-i}\boldsymbol{z} \\
&+ \alpha^2 \sum_{k=2}^{\infty} \frac{1}{k!} \sum_{i+j\le k-2} \boldsymbol{z}^T M^i \boldsymbol{z}\boldsymbol{z}^T M^j \boldsymbol{z}\boldsymbol{z}^T M^{k-2-i-j}\boldsymbol{z} + \cdots
\end{aligned}$$

For a positive definite M and integer i we have $\boldsymbol{z}^T M^i \boldsymbol{z} > 0$, implying that the coefficient of α^l is positive for all $l \geq 0$. □

5.3 Evaluation of f and its derivative

We now show that $f(\alpha)$ can be computed at a cost of $2n^2+O(n)$ floating point operations (flops), in addition to the flops that are needed to compute the eigendecomposition of a diagonal plus rank-one matrix. This eigendecomposition is expected to dominate the total cost of the function evaluation. In order to compute $f'(\alpha)$ as well, we need $3n^2 + O(n)$ additional flops. We note that n floating point exponentiations (which are significantly more costly than additions and multiplications) are also necessary to get $f(\alpha)$, however the computational cost is still dominated by the $O(n^2)$ additions/multiplications. No additional floating point exponentiations are needed to compute $f'(\alpha)$.

We assume that we maintain the eigendecomposition of each iterate of Bregman's algorithm as is done in the machine learning application, see [42]. We do not count the initial cost of computing this eigendecomposition. In some applications the factors form the input to the whole procedure and the updated factors are the output. Even if the factors have to be produced, or the matrix assembled upon return, these steps need to be carried out only once and the cost is amortized over the iterative steps of Bregman's algorithm.

In the presence of the eigendecomposition $X = V\Lambda V^T$, we can express $f(\alpha)$ as follows:

$$\begin{aligned} f(\alpha) &= \boldsymbol{z}^T e^{\log(V\Lambda V^T)+\alpha \boldsymbol{z}\boldsymbol{z}^T} \boldsymbol{z} - b = \boldsymbol{z}^T V e^{\log\Lambda + \alpha V^T \boldsymbol{z}\boldsymbol{z}^T V} V^T \boldsymbol{z} - b \qquad (5.8)\\ &= \boldsymbol{v}^T e^{\log\Lambda + \alpha \boldsymbol{v}\boldsymbol{v}^T} \boldsymbol{v} - b, \end{aligned}$$

where $\boldsymbol{v} = V^T\boldsymbol{z}$. We begin the evaluation by solving a diagonal plus rank-one eigendecomposition

$$\log\Lambda + \alpha \boldsymbol{v}\boldsymbol{v}^T = U\Theta U^T,\ \Theta = \operatorname{diag}(\theta) \tag{5.9}$$

which can be done in $O(n^2)$ time [32]. Next, we form $\boldsymbol{u} = U^T\boldsymbol{v}$ and get:

$$\begin{aligned} f(\alpha) &= \boldsymbol{v}^T e^{U\Theta U^T}\boldsymbol{v} - b = \boldsymbol{v}^T U e^{\Theta} U^T \boldsymbol{v} - b = \boldsymbol{u}^T e^{\Theta}\boldsymbol{u} - b \qquad (5.10) \\ &= (\boldsymbol{u} \circ e^{\theta})^T \boldsymbol{u} - b, \end{aligned}$$

where $\circ$ denotes the Hadamard product. We move on to the efficient computation of $f'(\alpha)$. The expression in (5.8) can be written in the form $\operatorname{tr}(e^{M(\alpha)}A)$ with $A = \boldsymbol{v}\boldsymbol{v}^T$ and $M(\alpha) = \log\Lambda + \alpha\boldsymbol{v}\boldsymbol{v}^T$. According to (5.2), (5.3) and (5.5) the derivative at α equals:

$$\begin{aligned} f'(\alpha) &= \operatorname{tr}\left((e^{M(\alpha)})' A\right) \qquad (5.11) \\ &= \operatorname{tr}\left(e^{\log\Lambda + \alpha\boldsymbol{v}\boldsymbol{v}^T} \cdot \left(h\left(\operatorname{ad}_{\log\Lambda+\alpha\boldsymbol{v}\boldsymbol{v}^T}\right)\boldsymbol{v}\boldsymbol{v}^T\right)\cdot \boldsymbol{v}\boldsymbol{v}^T\right). \end{aligned}$$

In order to compute the expression $h\left(\operatorname{ad}_{\log\Lambda+\alpha\boldsymbol{v}\boldsymbol{v}^T}\right)\boldsymbol{v}\boldsymbol{v}^T$, we reduce the problem to the diagonal case and then use the spectral decomposition of the operator in question.

Lemma 8. *Let $U \in \mathbb{R}^{n\times n}$ orthogonal and let Θ and B be arbitrary matrices. Then the following holds:*

$$\operatorname{ad}_{U\Theta U^T} B = U \operatorname{ad}_{\Theta}(U^T B U) U^T.$$

Proof. By the definition of the ad operator and $UU^T = I$, the right hand side above may be rewritten as:

$$U(\Theta U^T B U - U^T B U \Theta)U^T = U\Theta U^T B - B U\Theta U^T = \operatorname{ad}_{U\Theta U^T} B.$$

□

An analytical function can be extended to the operator space using the Jordan canonical form [34] (Chapter 1, Definition 1.2). Lemma 9 below generalizes the above result to analytical functions of the operator ad_Θ:

Lemma 9. *Let U, Θ and B be as in Lemma 8 and let g be analytical. The following holds:*

$$g(\mathrm{ad}_{U\Theta U^T})B = Ug(\mathrm{ad}_\Theta)(U^T BU)U^T.$$

Proof. Since g is analytical, it is sufficient to show that for any nonnegative integer k:

$$\mathrm{ad}^k_{U\Theta U^T} B = U\,\mathrm{ad}^k_\Theta(U^T BU)U^T.$$

For $k = 0$ the statement is immediate and we proceed by induction on k. Assume that the statement holds for $k \geq 1$, then apply Lemma 8 and the definition of ad to finish the proof:

$$\begin{aligned}
\mathrm{ad}^k_{U\Theta U^T} B &= \mathrm{ad}_{U\Theta U^T}(\mathrm{ad}^{k-1}_{U\Theta U^T} B) = \mathrm{ad}_{U\Theta U^T}(U\,\mathrm{ad}^{k-1}_\Theta(U^T BU)U^T) \\
&= U\,\mathrm{ad}_\Theta(U^T U\,\mathrm{ad}^{k-1}_\Theta(U^T BU)U^T U)U^T = U\,\mathrm{ad}^k_\Theta(U^T BU)U^T.
\end{aligned}$$

□

Our next step is to calculate $g(\mathrm{ad}_\Theta)$ using the spectral theorem. By the definition of the adjoint, one can easily show that if X is symmetric, then ad_X is self-adjoint, and so in our case we can use the eigendecomposition of ad_Θ to calculate $g(\mathrm{ad}_\Theta)$. The following argument mimics Lemma 8 of [50, Chapter 1], which gives the eigenvectors of ad_X; here we only need to deal with diagonal matrices. The definition of ad and the elementary calculation

$$\mathrm{ad}_\Theta \, \boldsymbol{e}_i \boldsymbol{e}_j^T = \Theta \boldsymbol{e}_i \boldsymbol{e}_j^T - \boldsymbol{e}_i \boldsymbol{e}_j^T \Theta = (\theta_i - \theta_j) \boldsymbol{e}_i \boldsymbol{e}_j^T \tag{5.12}$$

shows that the eigenvectors of ad_Θ are the n^2 matrices of the form $\boldsymbol{e}_i \boldsymbol{e}_j^T$ with eigenvalues $\theta_i - \theta_j$ respectively, where $\Theta = \mathrm{diag}(\theta)$.

Lemma 10. *Let* $\Theta = \mathrm{diag}(\theta)$ *be diagonal, and* g *analytical. For any* B *we have:*

$$g(\mathrm{ad}_\Theta) B = \sum_{i,j} g(\theta_i - \theta_j)(\boldsymbol{e}_i^T B \boldsymbol{e}_j) \boldsymbol{e}_i \boldsymbol{e}_j^T . \tag{5.13}$$

Proof. Repeated application of (5.12) establishes that for any nonnegative integer k:

$$\mathrm{ad}_\Theta^k \, B = \mathrm{ad}_\Theta^k \sum_{ij} (\boldsymbol{e}_i^T B \boldsymbol{e}_j) \boldsymbol{e}_i \boldsymbol{e}_j^T = \sum_{ij} (\theta_i - \theta_j)^k (\boldsymbol{e}_i^T B \boldsymbol{e}_j) \boldsymbol{e}_i \boldsymbol{e}_j^T .$$

The proof is completed by appealing to the analytical property of g. □

We note that the right hand side of (5.13) can be expressed as the Hadamard product of B and the matrix which has its (i, j) element equal to $g(\theta_i - \theta_j)$. According to Lemma 9 and equation (5.9), we have for any analytical g,

$$g\left(\mathrm{ad}_{\log \Lambda + \alpha \boldsymbol{v}\boldsymbol{v}^T}\right) \boldsymbol{v}\boldsymbol{v}^T = g(\mathrm{ad}_{U\Theta U^T}) \boldsymbol{v}\boldsymbol{v}^T = U g(\mathrm{ad}_\Theta)(U^T \boldsymbol{v}\boldsymbol{v}^T U) U^T .$$

Recall from Section 5.3 that we introduced $\boldsymbol{u} = U^T\boldsymbol{v}$ and that $\Theta = \text{diag}(\theta)$. Now we define matrix H to have (i, j) element equal to $h(\theta_i - \theta_j)$, where h is as in (5.6) and so finally from (5.11) and Lemma 10 we have:

$$\begin{aligned} f'(\alpha) &= \boldsymbol{v}^T e^{U\Theta U^T} U(H \circ \boldsymbol{u}\boldsymbol{u}^T)U^T\boldsymbol{v} && (5.14)\\ &= \boldsymbol{v}^T U e^{\Theta} U^T U(H \circ \boldsymbol{u}\boldsymbol{u}^T)\boldsymbol{u} = (\boldsymbol{u} \circ e^{\theta})^T(H \circ \boldsymbol{u}\boldsymbol{u}^T)\boldsymbol{u}. \end{aligned}$$

An alternative derivation for $f'(\alpha)$ based on the Daleckii–Krein theorem is also possible, see [5][p. 60, p.154].

Note that the computation of the eigenvalues and the vector $\boldsymbol{u}$ is also part of the computations needed to evaluate f at α, see (5.10). Therefore no additional eigendecompositions are necessary to compute the derivative. The direct computation of elements of the matrix H would require n^2 floating point exponentiations. Fortunately, we do not need to compute H explicitly, but instead we may expand the right hand side of (5.14) to get:

$$\begin{aligned} f'(\alpha) &= \sum_{i,j=1}^{n} u_i^2 u_j^2 e^{\theta_i} h(\theta_i - \theta_j) && (5.15)\\ &= 2 \sum_{\substack{1 \le i < j \le n \\ \theta_i \ne \theta_j}} u_i^2 u_j^2 \frac{e^{\theta_i} - e^{\theta_j}}{\theta_i - \theta_j} + 2 \sum_{\substack{1 \le i < j \le n \\ \theta_i = \theta_j}} u_i^2 u_j^2 e^{\theta_i} + \sum_{i=1}^{n} u_i^4 e^{\theta_i}. \end{aligned}$$

The above form exploits symmetry and allows the reuse of the e^{θ_i} terms available from the computation of $f(\alpha)$. We need $2.5n^2$ floating point additions, subtractions and multiplications and $0.5n^2$ floating point divisions. The cost of floating point divisions on modern architectures is between 2.5 to 3 times that of floating point addition. It is important to note that exponentiation is a much more expensive operation; its cost is about ten times that of a division. We summarize the computational steps required to compute $f(\alpha)$ and $f'(\alpha)$ in Algorithm 8.

Algorithm 8: Computations needed to evaluate f and f'.

```
Input  : Matrix $X$ with its $V\Lambda V^T$ eigendecomposition; vector $\boldsymbol{z}$;
         scalar $\alpha$.
Output: $f(\alpha)$, $f'(\alpha)$, see (5.1).              $\overset{*}{\pm}$        /        exp
1 $\boldsymbol{v} = V^T\boldsymbol{z}$                                   $2n^2$
2 Factor $\log\Lambda + \alpha\boldsymbol{v}\boldsymbol{v}^T = U\,\mathrm{diag}(\theta)U^T$        $\ell n^2$
3 $\boldsymbol{u} = U^T\boldsymbol{v}$                                   $2n^2$
4 $\boldsymbol{x} = \boldsymbol{u}\circ e^{\theta}$                                  $n$                  $n$
5 $f(\alpha) = \boldsymbol{x}^T\boldsymbol{u} - b$                             $2n$
6 $f'(\alpha) = \boldsymbol{x}^T(H\circ\boldsymbol{u}\boldsymbol{u}^T)\boldsymbol{u}$ see (5.14), (5.15)     $2.5n^2$  $0.5n^2$
```

The repeated computation of $f(\alpha)$ takes $(2+\ell)n^2+O(n)$ floating point operations (flops) where $\ell n^2+O(n)$ flops are needed for the eigendecomposition of a diagonal plus rank-one matrix[3]. Note that only steps 2 to 6 in Algorithm 8 have to be done repeatedly while finding the zero, so we did not include the matrix-vector multiplication in step 1 in the flop count for computing $f(\alpha)$. When we are computing $f'(\alpha)$, we are reusing intermediate results from the computation of $f(\alpha)$ and therefore we need only about $2.5n^2$ additional floating point additions/multiplications and $0.5n^2$ divisions. We expect the total computational cost to be dominated by the eigendecomposition.

The above discussion of the operation counts did not consider the issue of numerical accuracy. The difference quotient term of $(e^{\theta_i} - e^{\theta_j})/(\theta_i - \theta_j)$

[3]We observed the value of ℓ to typically fall between 25 and 50.

in (5.15) may suffer from catastrophic cancellation when θ_i and θ_j are not well separated. Our solution is to use an alternate formula when $x = (\theta_i - \theta_j)/2$ is sufficiently small:

$$\frac{e^{\theta_i} - e^{\theta_j}}{\theta_i - \theta_j} = e^{\theta_i/2} e^{\theta_j/2} \frac{\sinh(x)}{x} = e^{\theta_i/2} e^{\theta_j/2} \left(1 + \frac{x^2}{3!} + \frac{x^4}{5!} + \frac{x^6}{7!} + R(x)\right).$$

As indicated by the above equation we approximate $\sinh x$ using its Taylor expansion, which converges rapidly for small x. The native floating point instruction computing sinh produces accurate results, but if it were used for all (θ_i, θ_j) pairs, then we would pay a substantial performance penalty[4]. When $|x| \geq 0.1$, we use the original form that appears in (5.15), otherwise we use the above Taylor approximation. Elementary calculations using the Lagrange form of the remainder reveal that $|R(x)|$ is less than the machine epsilon when $|x| < 0.1$. Our implementation uses six floating point multiplications and three additions and no divisions[5] which should be compared to the two subtractions and a division in the original difference quotient formula. We observed no adverse effect on performance.

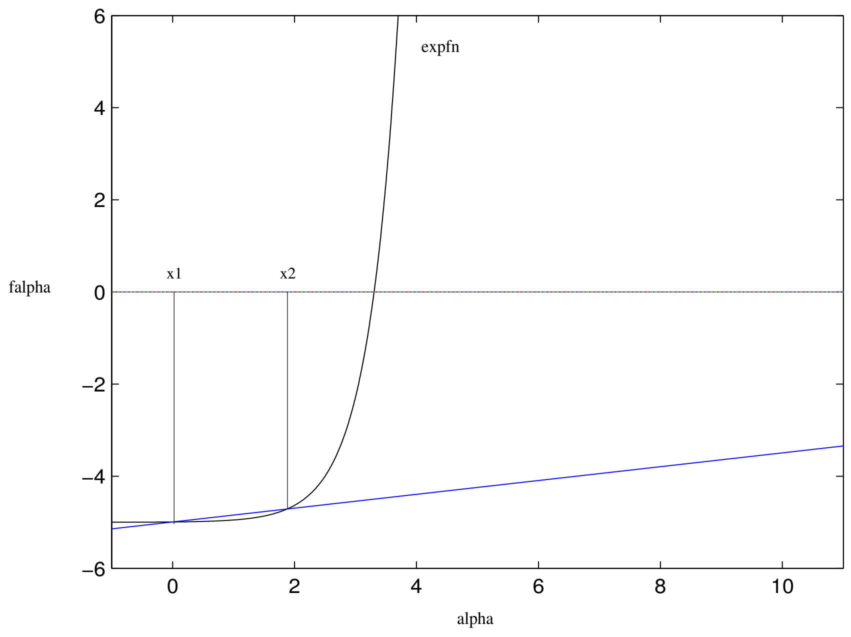
5.4 Logarithmic prescaling

All the zero-finding algorithms discussed use interpolation to fit simple functions to find the next approximation. Newton's method as well as the secant method use straight lines, the inverse quadratic interpolation method

[4]We found that the computation of sinh using a floating point instruction is 35 times longer than a multiplication.

[5]Constant divisions are turned into multiplications.

Figure 5.1: Overflow and underflow resulting in $\alpha_2 = \alpha_4$ (no progress) for the secant method.



uses the inverse of a quadratic function, as its name suggests, and Jarrat's method uses a function of the form given by (5.7).

When the graph of the objective function has a known specific shape, it may be advantageous, or even necessary, to fit a different function. We note that convexity of f, established in Lemma 7, implies convergence for the secant method, regardless of initial guesses. However, in floating point arithmetic, the presence of overflow, underflow and rounding error may result in lack of convergence. Figure 5.1 depicts the situation where the secant method does not make progress: the function value at α_3 is so large (not shown on the figure) that the computation of α_4 suffers from underflow, resulting in $\alpha_4 = \alpha_2$. This issue affects the other three zero-finding algorithms as well.

A similar problem occurs during the solution of the secular equation used to compute the eigendecomposition of a rank-one update to a diagonal matrix. The solution there is to fit a rational function which has the same asymptotes as the objective function [10, 45]. In our case, better convergence can be attained by fitting parameterized exponential functions. Doing so helps with the overflow/underflow problem depicted in Figure 5.1 and speeds up convergence.

We implement this idea of fitting a nonlinear function using a slightly different approach than what is found in [10, 45]. The main advantage of our solution is that we do not need to derive the (parameterized) fitting function, making it is easier to apply when a function such as (5.7) is used for interpolation. We apply a transformation to the function $f(\alpha)$ that yields a transformed function, $g(\alpha)$, and we use the zero-finders on $g(\alpha)$ in their original form. Our transformation applies a logarithmic prescaling; we introduce:

$$g(\alpha) = \log(f(\alpha) + b) - \log b \tag{5.16}$$

and observe that g is monotone and has the same zero as f. For the Newton-type methods we also need the derivative: $g'(\alpha) = f'(\alpha)/(f(\alpha)+b)$. Note that the additional computations are negligible.

5.5 Experimental results

We compare Newton's method and Jarrat's method, both of which employ the use of derivatives, to the secant method and inverse quadratic interpolation which are zero-finding algorithms that do not require calculation of the derivative.

We implemented the algorithms in C++ as MATLAB [47] and OCTAVE [27] compatible MEX files which call the Fortran DLAED4 function from LAPACK [2] for the diagonal plus rank-one eigendecompositions. We implemented the correction for accurate eigenvectors according to [32], and also implemented deflation in C++; we utilized fast linear algebra routines from BLAS [6]. In all algorithm versions we accepted an approximation as the zero when the function value was not larger then $n \cdot eps$ for an $n \times n$ matrix. We tested the performance of the algorithms in three sets of experiments. We revisited the protein data experiment (GYRB) from [41, 42]; we carried out a "synthetic" correlation matrix experiment motivated by [33]; and in the third experiment we find the zero of a slightly modified version of (5.1) as a result of the use of the so called "slack variables" in the hand written digits recognition (MNIST) experiment in [42].

We compare running times and the number of eigendecompositions (the most expensive step) executed by the zero-finding methods. We used a computer with an INTEL X3460 CPU running at 2.8GHz utilizing 8MB of cache. We ran the algorithms in single threaded mode (including the BLAS and LAPACK subroutines) with no other programs running.

Table 5.1: Running times and number of (rank-one update to a diagonal matrix) eigendecompositions executed by the various algorithms when solving the protein data classification problem using 1000 constraints. The middle column indicates the relative performance when compared to the secant method applied to f. Function g is defined by (5.16).

Protein data classification

Applied to	Method	run-time (sec)	run-time ratio compared to secant on f	number of eigendecomp.
	secant	7.49	1.00	43,781
f	inv. quad. int.	6.82	0.91	39,733
	Newton	5.63	0.75	30,148
	Jarratt	4.73	0.63	24,994
	secant	6.62	0.88	38,380
g	inv. quad. int.	6.20	0.83	35,941
	Newton	4.86	0.65	25,557
	Jarratt	4.49	0.60	23,523

The first experiment reproduces a result from [41, 42], where the objective is to find a 52×52 kernel matrix for protein data classification. The task is formulated as a matrix nearness problem using the von Neumann matrix divergence, $D_{vN}(X, Y) = \operatorname{tr}(X \log X - X \log Y - X + Y)$, as the nearness measure. We extract 1000 linear inequality constraints from the training data and use Bregman's iterative process starting from the identity matrix; for additional details we refer the reader to [41, 42]. Table 5.1 presents running times of the different zero-finders and the number of eigendecompositions needed. The methods using derivatives are seen to have better performance due to fewer eigendecompositions.

In the second experiment the objective is to find the nearest correlation

Table 5.2: Running times and number of (rank-one update to a diagonal matrix) eigendecompositions executed by the various algorithms when solving the correlation matrix problem. The middle column indicates the relative performance when comparing to the secant method applied to f. Function g is defined by (5.16).

Nearest correlation matrix

Applied to	Method	run-time (sec)	run-time ratio compared to secant on f	number of eigendecomp.
f	secant	201.3	1.00	9,255
	inv. quad. int.	190.1	0.94	8,568
	Newton	172.3	0.86	6,824
	Jarratt	145.5	0.72	5,321
g	secant	182.0	0.90	8,082
	inv. quad. int.	169.9	0.84	7,371
	Newton	141.8	0.70	5,094
	Jarratt	136.6	0.68	4,741

matrix X to a given positive definite starting matrix Y:

$$\text{minimize } D_{vN}(X, Y), \text{ subject to } X_{ii} = 1,\ i \in \{1, \ldots, n\},\ X \succ 0.$$

We generated Y to be a random symmetric matrix with eigenvalues uniformly distributed in $(0, 1)$. The results in Table 5.2 are averaged from ten runs using 500×500 randomly generated matrices. We observe again that the use of the derivative improves performance when compared to non-derivative based zero-finding methods.

In the third experiment we executed Bregman's algorithm using the MNIST data set consisting of images of handwritten digits encoded as 164-dimensional vectors. For details on this experiment we refer the reader to [42].

The zero-finding problem is a slightly modified version of (5.1) due to the use of slack variables. Here, we only give a short summary. Instead of enforcing the constraints, we penalize deviation from the desired conditions using the relative entropy $KL(\boldsymbol{x}, \boldsymbol{y}) = \sum_i (x_i \log(x_i/y_i) - x_i + y_i)$, the vector divergence from which the von Neumann matrix divergence is generalized:

$$\begin{aligned}&\text{minimize}_{X,\boldsymbol{b}}\ D_{vN}(X,Y) + \gamma KL(\boldsymbol{b}, \boldsymbol{b}_0),\\ &\text{subject to}\ \ \operatorname{tr}(XA_i) \le \boldsymbol{e}_i^T \boldsymbol{b},\ i \in \{1,\dots,c\},\ X \succ 0.\end{aligned}$$

The objective function measures the distance from the starting matrix Y as well as the amount by which the constraints are relaxed. The $\gamma > 0$ parameter controls how much "slack" we permit; in essence it is used to find the balance between over- and under-constraining the optimization problem.

The resulting zero-finding problem is a slightly modified version of (5.1):

$$\boldsymbol{z}^T e^{\log X + \alpha \boldsymbol{z}\boldsymbol{z}^T} \boldsymbol{z} + e^{\alpha/\gamma} - b = 0.$$

The derivative computation and other discussions of Section 5.2 apply after minor modifications.

In Table 5.3 we present the MNIST handwritten digits recognition experiment results for four zero-finding methods. We only show the versions using the logarithmic prescaling, because without that improvement the algorithms greatly suffer from the overflow/underflow problem discussed in Section 5.4, which would force the use of the bisection (or some other, but still inefficient) method for many iterations. Due to the modified objective function,

Table 5.3: Running times and number of (rank-one update to a diagonal matrix) eigendecompositions executed by the algorithms when solving the *MNIST* handwritten digits recognition problem. The algorithms were applied to function g as defined by (5.16).

MNIST handwritten digits recognition

Method	run-time (sec)	number of rank-one eigendecompositions
secant	281.7	444,385
inv. quad. int.	274.7	432,411
Newton	175.0	241,637
Jarratt	175.0	241,641

for which the logarithmic prescaling works very well, the number of iterations executed by the zero-finders is quite low (never more than four for Newton and Jarrat's method). The inverse quadratic interpolation provides its first approximation only in the fourth iteration and Jarratt's method in the third. Simply put, the faster convergence has no time to set in for inverse quadratic interpolation and Jarrat's method. As a result, the quadratic interpolation method yields only a slight benefit over the secant method and Jarratt's method does not yield any improvement over Newton's method. Newton's method requires nearly half the number of eigendecompositions when compared to inverse quadratic interpolation, while the running time improvement is 36%.

Chapter 6

Improvement of the Secular Equation Solver

Finding the eigenvalues of a diagonal plus rank-one matrix amounts to computing the zeros of the so-called secular function. The central idea of the state-of-the-art LAPACK implementation written by Ren-Cang Li is based on a modified Newton's method where a suitable rational approximation is used in place of straight lines. In this chapter, we describe an improvement to the secular equation solver that speeds up convergence by building on a long forgotten zero finding method by P. Jarratt. Numerical experiments of Section 5.5 demonstrate the advantages of our approach.

6.1 Background

The state-of the art LAPACK [2] implementation of the divide and conquer eigensolver [10, 32, 45] relies on the finding of the zeros of the secular equation:

$$f(x) = \frac{1}{\rho} + \sum_{k=1}^{n} \frac{z_k^2}{d_k - x}. \tag{6.1}$$

The zeros are interlaced between the poles of f located at d_k and the iterative algorithm computes them independent of each other, making the approach highly parallelizable. The algorithm is motivated by Newton's method. However, instead of fitting straight lines, rational approximating functions are used with up to three poles, two of which always coincide with the nearest poles of f, see [45] for details.

The improvement is based on the somewhat forgotten algorithm of P. Jarratt [40], dating back to 1966. He described how to use two prior iterations to speed up Newton's algorithm without the need for extra function evaluations to achieve a convergence order of $1 + \sqrt{3}$. We note however, that the *(asymptotic) efficiency index* (or the similar *order of convergence per function evaluation)* in the sense of Ostrowski [49][Chapter 3, Section 11] is a better measure of actual performance. For example, when the cost of the computation of the derivative equals to that of the function, the order of convergence per function evaluation is $\sqrt{2}$ for Newton's method. In comparison, inverse quadratic interpolation, the workhorse of Brent's method [9], which does not need the computation of the derivative has asymptotic efficiency index of $1.839 > \sqrt{2}$. The algorithm due to Jarratt can perform well when the additional cost of the derivative computation is less than the cost of the function evaluation. It is also necessary that the cost of the rational approximation employed is sufficiently small in comparison to the function evaluation. This was the case for the zero-finding problem discussed in Chapter 5, see also [56]. The algorithm finding the zero of the function involving the matrix exponential takes advantage of the accelerated convergence of Jarrat's method. We emphasize that Jarratt's method is always an improvement over Newton's method when the cost of the rational approximation is small compared to the function evaluation. Only when comparing the performance to inverse quadratic interpolation one needs to consider the cost of the derivative computations.

6.2 Jarratt's method

First we recall the zero finder due to P. Jarratt that we have also used in Chapter 5.2, full details are in [40]. This iterative zero finder uses a rational interpolating function of the form

$$y = \frac{x-a}{bx^2+cx+d} \tag{6.2}$$

fitted to the function and derivative values from *two* previous iterations. We denote the function by g, the last two approximations to its zero by x_{i-2} and x_{i-1}, and we also introduce the $g_{i-1} = g(x_{i-1})$, $g_{i-2} = g(x_{i-2})$ and $g'_{i-1} = g'(x_{i-1})$, $g'_{i-2} = g'(x_{i-2})$ shorthands. The new approximation to the zero denoted by x_i can be computed as follows:

$$x_i = x_{i-1} - \frac{(x_{i-1}-x_{i-2})g_{i-1}[g_{i-2}(g_{i-1}-g_{i-2}) - (x_{i-1}-x_{i-2})g_{i-1}g'_{i-2}]}{2g_{i-1}g_{i-2}(g_{i-1}-g_{i-2}) - (x_{i-1}-x_{i-2})(g_{i-1}^2 g'_{i-2} + g_{i-2}^2 g'_{i-1})}. \tag{6.3}$$

We outlined Jarratt's method as Algorithm 7 in Section 5. In its original form, Jarratt's method can simply replace Newton's method; however in the case of the secular function the fitting of straight lines is already replaced by a rational approximation. In the next section we formulate the secular equation solver in a way that will make the connection to Newton's method clear and the application of Jarrat's method straightforward.

6.3 Reparameterization

We formulate the fitting of a rational function in place of straight lines in a way which provides a quite convenient view for our exposition and also for the algorithm implementation. We can transform an arbitrary zero finding problem $f(x) = 0$ to

$$g(x) = h^{-1}(f(x)) - h^{-1}(0) = 0 \tag{6.4}$$

where the function h is strictly monotone and continuous. If we assume that h is sufficiently smooth, then the function g inherits the necessary smoothness of f and we can apply a given zero finder algorithm to g in place of f.

In case of the secular function (6.1) we select h to be the two or three pole approximation between the poles at d_i and d_{i+1} as described in [45]. The approximation h is constructed such that:

$$h(x_{i-1}) = f(x_{i-1}), \quad h'(x_{i-1}) = f'(x_{i-1}),$$

where x_{i-1} is the previous approximation to the zero. The derivative of g defined by (6.4) is:

$$g'(x) = \frac{f'(x)}{h'(h^{-1}(f(x)))}.$$

We compute $g(x_{i-1})$ and $g'(x_{i-1})$ as follows:

$$\begin{aligned} g(x_{i-1}) &= h^{-1}(f(x_{i-1}) - h^{-1}(0) = h^{-1}(h(x_{i-1})) - h^{-1}(0) = x_{i-1} - h^{-1}(0) \\ g'(x_{i-1}) &= \frac{f'(x_{i-1})}{h'(h^{-1}(f(x_{i-1})))} = \frac{f'(x_{i-1})}{h'(x_{i-1})} = 1. \end{aligned}$$

The Newton iteration applied to g yields: $x_i = x_{i-1} - g(x_{i-1})/g'(x_{i-1}) = h^{-1}(0)$, in other words, the next approximation x_i is the zero of the approximating function h just as called for when solving the secular equation in [45].

Now, that we have established that the zero finding algorithm in question can be viewed as a classical Newton iteration step applied to the function g, it will be straightforward to apply Jarratt's method as well. We describe the details in the next section.

It is worth pointing out the use of a different h approximation in each iteration. This is slightly different from the approach we used in Section 5 where we employed a single reparametrization function.

6.4 Algorithm

We are modifying the secular equation solver described in [45] by improving the approximation by using Jarratt's method [40] and as a result we reduce the number of iterations needed. We continue to use the definitions introduced in Section 6.3.

We will apply Jarratt's rational approximation to the function g defined by (6.4). We already have $g(x_{i-1}) = x_{i-1} - h^{-1}(0)$ and $g'(x_{i-1}) = 1$; we need to compute $g(x_{i-2})$ and $g'(x_{i-2})$ as well in order to successfully apply step 5 of Algorithm 7. By definition $g(x_{i-2}) = h^{-1}(f(x_{i-2})) - h^{-1}(0)$, and therefore we solve the $h(x) - f(x_{i-2}) = 0$ zero finding problem in addition to the $h(x) = 0$ (which was already needed to compute $g(x_{i-1})$). We introduce η and μ as the solutions of these equations:

$$h(\eta) = 0, \quad h(\mu) - f(x_{i-2}) = 0. \tag{6.5}$$

We note that since h is a two or three pole approximation, the above zero finding tasks are easier to solve (and cost much less computationally) than working with f. Definitions and elementary calculations yield:

$$g(x_{i-2}) = \mu - \eta,\ g'(x_{i-2}) = \frac{f'(x_{i-2})}{h'(\beta)},\ g(x_{i-1}) = x_{i-1} - \eta,\ g'(x_{i-1}) = 1. \tag{6.6}$$

We present the resulting algorithm as Algorithm 9.

Algorithm 9: High level proposed secular equation solver.

```
   Input  : Secular equation ($f$ in (6.1)) parameters: $z_k$, $d_k$, $\rho$.
            Index of zero sought: $k$.
   Output: Approximations $x_1, x_2, \dots$ to the zero of $f$ in the interval
            $(d_k, d_{k+1})$.
1  Compute the initial guess $x_1$.
2  Evaluate $f(x_1)$ and $f'(x_1)$.
3  Compute the $h$ approximation with two or three poles, such that
   $h(x_1) = f(x_1)$ and $h'(x_1) = f'(x_1)$ holds.
4  Solve $h(x_2) = 0$.
5  for $i = 3, 4, \dots$ do
6      Compute the $h$ approximation with two or three poles, such
       that $h(x_{i-1}) = f(x_{i-1})$ and $h'(x_{i-1}) = f'(x_{i-1})$ holds.
7      Solve $h(\eta) = 0$.
8      Solve $h(\mu) - f(x_{i-2}) = 0$.
9      Compute $h'(\mu)$.
10     Set $g_{i-2} = \mu - \eta$, $g'_{i-2} = f'(x_{i-2})/h'(\mu)$.
11     Set $g_{i-1} = x_{i-1} - \eta$, $g'_{i-1} = 1$.
12     Compute $x_i$ using (6.3).
13 end
```

6.5 Numerical accuracy and implementation

We already encountered one issue related to numerical accuracy that we did not yet discuss explicitly. Recall the computation of the next approximation to the zero, described by Equation (6.3):

$$x_i = x_{i-1} - \frac{(x_{i-1} - x_{i-2})g_{i-1}[g_{i-2}(g_{i-1} - g_{i-2}) - (x_{i-1} - x_{i-2})g_{i-1}g'_{i-2}]}{2g_{i-1}g_{i-2}(g_{i-1} - g_{i-2}) - (x_{i-1} - x_{i-2})(g^2_{i-1}g'_{i-2} + g^2_{i-2}g'_{i-1})}.$$

We compute a *small* correction to x_{i-1} in the second term above and therefore even when the number of useful digits in the computed correction term is relatively small, it still allows the computation of x_i to high accuracy.

In step 9 of Algorithm 9 we have to be careful in evaluating $h'(\mu)$. Note, that the parameters of function h are to be computed so that h and h' matches f and f' at the two prior points. However, the actual computed parameters suffer from errors due to the floating point representation. If we substitute into the computed h we find that the results do not equal the prescribed values of f and f'. The reparametrization that we introduced in Section 6.3 does not rely on h satisfying the equations determining the approximation. One possible solution is to recompute $h'(x_{i-1})$ even though in exact arithmetic this value must be 1.

6.6 Experiments

We compared our secular equation solver to the state-of-the-art implementation that can be found in LAPACK. Our goal is to demonstrate that our improved secular equation solver computes the eigenvalues to the same acuracy as the state-of-the-art implementation and that a small improvement in execution speed can be expected. We executed our algorith in standalone mode as well as integrated into the Divide and Conquer (D&C) eigensolver. Our goal is to compare our secular equation algorithm to the state-of-the-art implementation. It is not our intention to compare the symmetric eigensolvers including D&C, for such a study we refer to [22].

We used a computer with an INTEL X3460 CPU running at 2.8GHz utilizing 8MB of cache. We ran the algorithms in single threaded mode, and we report CPU time.

In the first experiment we look at eight problem instances which challenge the secular equation zero-finders by featuring clustered zeros. These examples were provided by Ren-Cang Li [46]. The data originates from stress tests and actual (problematic or failing) runs of the divide and conquer algorithm originated from problem reports. Subsequent analysis resulted in algorithm modifications for the secular equation solver. We present the results in Table 6.1. Our improved algorithm implementation reduces the maximum number of iterations (approximation steps) needed to find the zeros in seven out of the eight examples as well as the total number of iterations executed. The lack of improvement in the eighth example can be explained by the smaller problem size and the fact that a relatively low number of approximation steps are needed for both algorithms.

In the next set of experiments we look at four large matrix examples that we extracted from the STETESTER software package [21], see also [22]. We

Table 6.1: Total number of iterations and maximum iteration for the Ren-Cang Li examples.

Eight clustered eigenvalue tests									
max # of iterations	OLD	8	6	7	7	7	8	6	5
	NEW	7	5	6	6	6	7	5	5
total # of iterations	OLD	113	194	245	1805	337	134	173	86
	NEW	110	186	232	1687	334	132	165	86
problem size		30	40	50	364	100	40	40	23

Table 6.2: Running times and maximum relative errors in the eigenvalues computed for selected large symmetric tridiagonal matrices.

Four large eigenvalue tests								
	Clement		NASA		BCSSTKM13		Bennighof	
	time (sec)	relative error	time (sec)	relative error	time (sec)	relative error	time (sec)	relative error
OLD	8.36	3.8e-13	2.00	2.0e-13	5.49	8.2e-11	76.40	4.9e-08
NEW	8.05	4.0e-13	1.95	2.0e-13	5.37	6.0e-11	73.61	4.9e-08

incorporated our improved secular equation solver into the divide-and-conquer eigensolver and computed the eigenvalues of the symmetric tridiagonal matrices. In Table 6.2 we feature the four matrices: Clement which has known exact eigenvalues and three large matrices that arose from aerospace engineering, see [22] and references therein. The Clement matrix in the experiment has dimension $n = 10000$, its eigenvalues are well separated, namely they are $\pm 1, \pm 2, \ldots \pm 10000$. Deflation is limited for this matrix [22]. The matrices NASA, BCSSTKM13 and Bennighof have dimensions 2146, 8012 and 24873 respectively, they arose in aerospace engineering applications. We ob-

Table 6.3: Iteration and floating point counts and CPU times when using quadruple precision IEEE arithmetic for selected tridiagonal matrices.

Quadruple precision tests									
	Clement			NASA			BCSSTKM13		
	iter. $\times 10^3$	flops $\times 10^6$	time (sec)	iter. $\times 10^3$	flops $\times 10^6$	time (sec)	iter. $\times 10^3$	flops $\times 10^6$	time (sec)
OLD	475	4114	886.5	325	2895	624.6	375	3948	856.8
NEW	423	3705	817.1	296	2694	587.8	346	3641	804.6

serve that our solver computes the eigenvalues to the same accuracy while it executes 3.5% faster.

In the third set of experiments we look at iteration and floating point operation counts when using *quadruple precision*. Higher precision requirements result in more iteration per zero when solving the secular equation and therefore we expect a larger improvement. In Table 6.3 we observe that the iteration count and the number of floating point operations is reduced by more than 7.5%. The quadruple precision IEEE arithmetic is not usually supported in hardware, but instead emulated in software, resulting in much longer execution times. We provide the execution times for completeness, while we stress that the presence of the software emulation layer is not fully considered. We run this experiment using an INTEL T7700 CPU running at 2.4GHz.

6.7 Convergence speed

We present an alternative proof for the order of convergence for Jarratt's method [40]. Our first lemma establishes the progress the algorithm makes in each iteration, provided that the starting point is sufficiently close to the approximated zero.

Lemma 11. *Assume that f is at least three times continuously differentiable in a neighborhood of a simple root α of f. Let x and y be approximations to the root satisfying $2|y - \alpha| \leq |x - \alpha| < \delta$. If δ is small enough, then the next approximation that we denote by z as it is calculated by Algorithm 7 satisfies:*

$$|z - \alpha| < C|x - \alpha||y - \alpha|^2$$

where $C > 0$ depends on δ, but not on x or y.

Proof. Without loss of generality we assume that $\alpha = 0$ is the simple root of f we are approximating. This restriction simplifies the calculations, but could be removed in a straightforward manner. Our goal is to bound $|z|$ in terms of the previous iterates x and y, provided that δ is small enough. We will not explicitly calculate a suitable value for δ, but it will be clear from our argument that the value exists, and could be calculated if needed.

The formula for z can be written as N/D instead of the sum of y and a correction term which was preferred for numerical computations:

$$\begin{aligned} N &= (x+y)f_x f_y(f_y - f_x) - (y-x)(xf_y^2 f'_x + yf_x^2 f'_y) \\ D &= 2f_x f_y(f_y - f_x) - (y-x)(f_y^2 f'_x + f_x^2 f'_y). \end{aligned}$$

In order to prove the desired upper bound for the next approximation z, we will bound the numerator N and the denominator D separately. We want to express them using the Taylor series expansion of both f and its derivative f' around $\alpha = 0$.

Taylor's theorem applied to f and its derivative (note that $f(0) = 0$) can be used to express $f(x)$, $f(y)$, $f'(x)$ and $f'(y)$ as follows:

$$f(x) = f'(0)x + \frac{1}{2}f''(0)x^2 + \frac{1}{6}f'''(\xi_x)x^3 \tag{6.7}$$
$$f(y) = f'(0)y + \frac{1}{2}f''(0)y^2 + \frac{1}{6}f'''(\xi_y)y^3 \tag{6.8}$$
$$f'(x) = f'(0) + f''(0)x + \frac{1}{2}f'''(\eta_x)x^2 \tag{6.9}$$
$$f'(y) = f'(0) + f''(0)y + \frac{1}{2}f'''(\eta_y)y^2 \tag{6.10}$$

We substitute these into the formulas for N and D and group the terms according to the powers of x and y. A significant amount of algebraic manipulations are necessary to reach the desired form; fortunately computer algebra packages can automate this tedious task.

We used the MAXIMA computer algebra system to carry out the simplifications. We issued the following commands:

```
fx  : T1*x + 1/2*T2*x^2 + 1/6*T3xix*x^3;
fy  : T1*y + 1/2*T2*y^2 + 1/6*T3xiy*y^3;
fdx : T1   +    T2*x    + 1/2*T3etax*x^2;
fdy : T1   +    T2*y    + 1/2*T3etay*y^2;
```

```
N   : (x+y)*fx*fy*(fy-fx)-
      (y-x)*(x*fy^2*fdx+y*fx^2*fdy);
D   : (2*fx*fy*(fy-fx)-
      (y-x)*(fy^2*fdx+fx^2*fdy));

collectterms(expandwrt(N, x, y), x, y);
collectterms(expandwrt(D, x, y), x, y);
```

The $T1$, $T2$, $T3xix$, $T3xiy$, $T3etax$, $T3etay$ variables in the above MAXIMA commands correspond to $f'(x)$, $f''(x)$, $f'''(\xi_x)$, $f'''(\xi_y)$, $f'''(\eta_x)$, $f'''(\eta_y)$ from equations (6.7)-(6.10); the remaining variable correspondences should be self explanatory.

The numerator can be bound to be small if all the terms occur with large powers of x and y; therefore we need to look for the terms with small exponents. The terms having the smallest x, y exponents have order six; they dominate all the other terms when δ and hence x, y are small:

$$x^4y^2f'(0)^2\tfrac{f'''(\eta_x)-f'''(\xi_x)}{2} \tag{6.11}$$

$$x^2y^4f'(0)^2\tfrac{f'''(\xi_y)-f'''(\eta_y)}{2} \tag{6.12}$$

$$x^3y^3f'(0)^2\left(\tfrac{f'''(\xi_x)-f'''(\xi_y)}{6}+\tfrac{f'''(\eta_y)-f'''(\eta_x)}{2}\right) \tag{6.13}$$

We denote by $M = M_\delta$ the absolute supremum of the continuous f''' on $[-\delta, \delta]$. Elementary calculations exploiting $|y| \le |x|/2$ yield that the absolute sum of the dominating terms is bound by $\frac{23}{12}|f'(0)|^2Mx^4y^2$; therefore if δ is small enough, we have:

$$|N| \leq 2|f'(0)|^2M|x|^4|y|^2.$$

Note that $f'(0) \neq 0$ because we are approximating a simple root.

Similarly, the terms in D are dominated by $(x-y)^3f'(0)^3$ when δ is small enough. Since $|(x-y)^3| \geq |x|^3/8$, we derive that:

$$|D| \geq \frac{|f'(0)|^3|x|^3}{9}.$$

Finally, we bound $|z|$ as required:

$$|z| = \frac{|N|}{|D|} \leq \frac{18|f'(0)|^2M|x|^4|y|^2}{|f'(0)|^3|x|^3} = \frac{18M}{|f'(0)|}|x||y|^2.$$

□

When the function f is a little smoother, we can prove a stronger result.

Lemma 12. *Assume that f is at least four times continuously differentiable in a neighborhood of a simple root α of f. Using the same notations as in Lemma 11 the next approximation z satisfies:*

$$|z-\alpha| < C|x-\alpha|^2|y-\alpha|^2.$$

Proof. We improve the bound established for the dominating terms of the numerator N from the proof of Lemma 11.

Consider $f'''(\eta_x) - f'''(\xi_x)$ from equation (6.11) and express this difference using the mean–value theorem:

$$f'''(\eta_x) - f'''(\xi_x) = f^{(4)}(\nu_x)(\eta_x - \xi_x),$$

where ν_x is in the interval with endpoints ξ_x and η_x. Since $|\xi_x|, |\eta_x| < |x|$, we deduce that $|f'''(\eta_x) - f'''(\xi_x)| \leq 2S|x|$, where $S = S_\delta$ denotes the absolute supremum of $f^{(4)}$ on $[-\delta, \delta]$. Apply the same argument to the other dominating terms to sharpen the bound on N as follows:

$$|N| \leq 4|f'(0)|^2 S|x|^5|y|^2.$$

Using the bound on $|D|$ from the proof of Lemma 11 yields:

$$|z| = \frac{|N|}{|D|} \leq \frac{36|f'(0)|^2 S|x|^5|y|^2}{|f'(0)|^3|x|^3} = \frac{36S}{|f'(0)|}|x|^2|y|^2. \tag{6.14}$$

□

We expect Jarratt's iteration to be at least quadratically convergent. In the following sequel we prove that the order of the convergence is at least $1+\sqrt{3}$ in a neighborhood of a simple root of a sufficiently smooth function. First, we have a short diversion regarding the definition of *order of convergence*. We start out with a simple definition that can be applied to certain sequences only. We say, that the order of convergence of the sequence $x_i \to \alpha$ $(i = 0, 1, 2, \ldots)$ is $\kappa > 1$ if the limit

$$\lim_{i\to\infty} \frac{|x_i - \alpha|}{|x_{i-1} - \alpha|^\kappa} \tag{6.15}$$

is finite and non-zero. For example, 2^{-2^i} converges to zero with order 2 as i approaches infinity. However, the apparently more rapidly converging sequence with elements $2^{-2^i}/(i+1)$ does not have an order of convergence in the above sense. For $1 < \kappa \leq 2$ the limit in question is zero, while for $\kappa > 2$ the limit is infinity, but it is never a finite non-zero value. Furthermore, sequences with variable convergence "speed" like the one with elements $2^{-4^{\lfloor i/2 \rfloor}}$ also evade the simple definition. The counterexamples motivate the following extension.

Definition 5. *We say that the sequence x_i converges to α with at least order $\kappa > 1$, if there exists a nonnegative sequence ϵ_i such that for all i indices $|x_i - \alpha| < \epsilon_i$, and*

$$\lim_{i\to\infty} \frac{\epsilon_i}{\epsilon_{i-1}^{\kappa}} = \kappa.$$

The convergence order of Jarratt's method is easier to determine when the limit in (6.15) exists, in other words when the simple definition is sufficient. Application of the next result—after shifting the root to zero—leads to the proof that Jarratt's method has convergence order at least $1 + \sqrt{3}$ under the assumptions of Lemma 12 and the existence of the limit in (6.15).

Lemma 13. *Assume that the sequence x_i ($i = 0, 1, 2, \ldots$) converges to zero, and it satisfies $|x_i| \leq C x_{i-2}^2 x_{i-1}^2$ for $i \geq 2$. If for some $\kappa > 1$ the limit*

$$\mu = \lim_{i\to\infty} \frac{|x_i|}{|x_{i-1}|^{\kappa}} \tag{6.16}$$

is finite and non-zero, then $\kappa \geq 1 + \sqrt{3}$.

Proof. We may assume for simplicity that $|x_i| < 1$ holds for all indices. We first prove that:

$$\lim_{i\to\infty} \frac{\log|x_i|}{\log|x_{i-1}|} = \kappa. \tag{6.17}$$

Indeed, $\log \mu$ can be expressed as $\lim_{i\to\infty}(\log|x_i| - \kappa \log|x_{i-1}|)$ and a division by $\log|x_{i-1}|$ which converges to $-\infty$ leads to:

$$0 = \lim_{i\to\infty}\left(\frac{\log|x_i|}{\log|x_{i-1}|} - \kappa\right),$$

and this is equivalent to (6.17). Next, apply the logarithm function to both sides of the inequality $|x_i| \leq Cx_{i-2}^2 x_{i-1}^2$, and divide by the negative $\log|x_{i-1}|$:

$$\frac{\log|x_i|}{\log|x_{i-1}|} \geq \frac{\log C}{\log|x_{i-1}|} + 2 + \frac{\log|x_{i-2}|}{\log|x_{i-1}|}.$$

Take the limit of each term as i approaches infinity and arrive to:

$$\kappa \geq 2 + \frac{2}{\kappa}$$

which is equivalent to $\kappa \geq 1 + \sqrt{3}$ since κ is positive. $\square$

Note that the existence of the limit in (6.17) does not imply that the μ limit in (6.16) is non-zero. For practical purposes the sequence appearing in (6.17) is useful, because it eliminates the need to guess the value of κ. The following lemma is the next step towards establishing the order of convergence in the sense of Definition 5.

Lemma 14. *Assume that the sequence x_i ($i = 0, 1, 2, \ldots$) satisfies $|x_i| \leq Cx_{i-2}^2 x_{i-1}^2$ for $i \geq 2$. We can bound the ith iterate ($i \geq 2$) in terms of x_0 and x_1 by*

$$|x_i| \leq C^{G_i} x_0^{2H_{i-1}} x_1^{H_i}$$

where G_i, H_i are defined recursively by the equations and starting values:

$$G_i = 2G_{i-2} + 2G_{i-1} + 1 \quad G_0 = 0 \quad G_1 = 0$$

$$H_i = 2H_{i-2} + 2H_{i-1} \quad H_0 = 0 \quad H_1 = 1.$$

Proof. The proof uses induction on i. For $i = 2$ the claim coincides with $|x_2| \leq Cx_0^2 x_1^2$, which is an instance of the assumed inequality. We establish the induction step by writing ($i \geq 3$):

$$\begin{aligned} |x_i| &\leq Cx_{i-2}^2 x_{i-1}^2 \leq \\ &\quad C\left(C^{G_{i-2}} x_0^{2H_{i-3}} x_1^{H_{i-2}}\right)^2 \left(C^{G_{i-1}} x_0^{2H_{i-2}} x_1^{H_{i-1}}\right)^2. \end{aligned}$$

Collect the exponents with the same base and notice that the defining equations for G_i, H_i yield the required formulas. □

We seek closed formulas for both G_i and H_i, in particular we hope that the growth rate of these sequences will allow us to prove the convergence order of Jarrat's algorithm. The method used for calculating the closed formulas is classical, maybe with the exception of the idea of reducing the calculation of G_i to that of H_i. We include the proofs for completeness.

Lemma 15. *A closed formula for the sequence H_i defined in Lemma 14 is the following:*

$$H_i = \frac{1}{2\sqrt{3}}\left((1+\sqrt{3})^i - (1-\sqrt{3})\right)^i.$$

Proof. Sequences obeying the recursion that H_i satisfies are closed for linear combinations. We will determine two geometrical series which satisfy the recursion and then we will form H_i as a linear combination.

Denote the ith element of the geometric series by q^i. The following must hold:

$$q^i = 2q^{i-2} + 2q^{i-1}.$$

All these equations are equivalent to $q^2 - 2q - 2 = 0$, called the *characteristic equation*. The roots provide the $(1+\sqrt{3})^i$ and $(1-\sqrt{3})^i$ geometrical series and we determine the linear combination producing H_i by considering the H_0, H_1 elements only. Elementary calculation finishes the proof. $\square$

Lemma 16. *The sequences G_i, H_i as defined above satisfy*[1]*:*

$$G_i = \sum_{j=0}^{i-1} H_j$$

and as a consequence a closed formula for G_i is:

$$G_i = \frac{1}{6}\left((1+\sqrt{3})^i + (1-\sqrt{3})^i - 2\right).$$

Proof. We will use induction on i. When $i = 0, 1$, the formula is satisfied based on the definition of the sequences. Let $i \geq 2$ and assume the that the formula is true for $i-1$ and $i-2$. Use the defining equations in addition and write:

$$\begin{aligned} G_i &= 2G_{i-2} + 2G_{i-1} + 1 = 2\sum_{j=0}^{i-3} H_j + 2\sum_{k=0}^{i-2} H_k + 1 \\ &= 1 + H_0 + \sum_{j=0}^{i-3}(2H_j + 2H_{j+1}) = H_1 + \sum_{j=0}^{i-3} H_{j+2} \\ &= \sum_{j=0}^{i-1} H_j. \end{aligned}$$

The closed formula for G_i follows from the closed formula for H_j $(j < i)$ provided by Lemma 15. $\square$

[1]The empty sum is zero by convention.

Finally, we can state:

Theorem 9. *Consider the sequence x_i produced by the Jarratt's method defined by Algorithm 7 that converges to a simple root α of the function f. We assume that f is at least four times continuously differentiable in a neighborhood of α. It follows that the order of the convergence is at least $\kappa = 1 + \sqrt{3}$.*

Proof. To simplify our arguments we shift the root to zero and assume that the whole approximating sequence is inside the neighborhood provided by Lemma 12. We apply Lemma 14 to get:

$$|x_i| \leq C^{G_i} x_0^{2H_{i-1}} x_1^{H_i}.$$

Define $\epsilon_i = C^{G_i} x_0^{2H_{i-1}} x_1^{H_i}$. To satisfy the requirements of Definition 5 we are left to prove that $\lim_{i\to\infty} \epsilon_i/\epsilon_{i-1}^{\kappa}$ is a finite non-zero value. Expand the $\epsilon_i/\epsilon_{i-1}^{\kappa}$ expression using Lemmas 15 and 16 collect the exponents. The largest terms conveniently cancel out; for example, the exponent of x_0 becomes:

$$\frac{1}{\sqrt{3}} \left(\frac{2}{\kappa}\right)^{i-1} \left(\kappa - \frac{2}{\kappa}\right).$$

As the index i approaches infinity, the above exponent converges to zero, a consequence of $\kappa > 2$. The exponent of x_1 also converges to zero by a similar argument, while the exponent of C can be seen to converge to $1/\sqrt{3}$. So indeed, $\lim_{i\to\infty} \epsilon_i/\epsilon_{i-1}^{\kappa}$ is a finite non-zero value, finishing the proof. $\square$

Finally, we can outline a proof establishing that our secular equation solver algorithm converges with order (at least) $1 + \sqrt{3}$. We recall the reparametrization discussed in Section 6.3 and that the function h is an approximation to f near the zero. We view the secular equation solver algorithm as an application of Jarratt's method to the $g(x) = h^{-1}(f(x)) - h^{-1}(0)$ function. Even though the function h is different for each step, it is approximating f around the zero and therefore a bound similar to (6.14) can be established (with a different constant) and that proves that the order of convergence is at least as much as for Jarratt's method.

Bibliography

[1] C.-G. Ambrozie. Finding positive matrices subject to linear restrictions. *Linear Algebra and its Applications*, 426:716–728, 2007.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[3] J. Barnes and P. Hut. A hierarchical $O(n \log n)$ force calculation algorithm. *Nature*, 324:446–449, 1986.

[4] R. B. Bendel and M. R. Mickey. Population correlation matrices for sampling experiments. *Commun. Statist. Simul. Comp.*, B7(2):163–182, 1978.

[5] Rajendra Bhatia. *Positive Definite Matrices*. Princeton, 2006.

[6] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (blas). *ACM Trans. Math. Soft.*, 28:135–151, 2002.

[7] Bernhard G. Bodmann and Vern I. Paulsen. Frames, graphs and erasures. *Linear Algebra and its Applications*, 404:118–146, July 2005.

[8] L. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Comp. Mathematics and Mathematical Physics*, 7:200–217, 1967.

[9] R. P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, 1973.

[10] J. R. Bunch, Ch. P. Nielsen, and D. C. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerical Mathematics*, 31:31–48, 1978.

[11] P. J. Cameron and J. H. van Lint. *Designs, Graphs, Codes, and their Links.* Cambridge University Press, 1991.

[12] Y. Censor and S. Zenios. *Parallel Optimization.* Oxford University Press, 1997.

[13] N. N. Chan and Kim-Hung Li. Diagonal elements and eigenvalues of a real symmetric matrix. *J. Math. Anal. Appl.*, 91:562–566, 1983.

[14] O. Christensen. *An Introduction to Frames and Riesz Bases.* Birkhauser, Boston, MA, 2003.

[15] J. H. Conway, R. H. Hardin, and N. J. A. Sloane. Packing lines, planes, etc.: Packings in Grassmannian spaces. *Experimental mathematics*, 5(2):139–159, 1996.

[16] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36:177–195, 1981.

[17] P. I. Davies and N. J. Higham. Numerically stable generation of correlation matrices and their factors. *BIT*, 40(4):640–651, 2000.

[18] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information–theoretic metric learning. In *Proc. 24th International Conference on Machine Learning*, 2007.

[19] P. Delsarte, J. M. Goethals, and J. J. Seidel. Spherical codes and designs. *Geometriae Dedicata*, 67(3):363–388, 1977.

[20] James D. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[21] James W. Demmel, Osni A. Marques, Beresford N. Parlett, and Christof Vömel. A testing infrastructure for lapack's symmetric eigensolvers, 2007.

[22] James W. Demmel, Osni A. Marques, Beresford N. Parlett, and Christof Vömel. Performance and accuracy of lapack's symmetric tridiagonal eigensolvers. *SIAM J. Sci. Comput.*, 30:1508–1526, 2008.

[23] I. S. Dhillon and J. Tropp. Matrix nearness problems using bregman divergences. *SIAM Journal on Matrix Analysis and Applications*, 2007.

[24] Inderjit S. Dhillon, Jr. Robert W. Heath, Mátyás A. Sustik, and Joel A. Tropp. Generalized finite algorithms for constructing hermitian matrices with prescribed diagonal and spectrum. *SIAM Journal on Matrix Analysis and Applications*, 27(1):61–71, 2005.

[25] J. J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenproblem. *SIAM J. Sci. Statist. Comp.*, 8:139–154, 1987.

[26] R. J. Duffin and A. C. Schaeffer. A class of nonharmonic fourier series. *Transactions of the American Mathematical Society*, 72:341–366, 1952.

[27] John W. Eaton. *GNU Octave Manual.* Network Theory Limited, 2002.

[28] G. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15:318–334, 1973.

[29] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.

[30] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.

[31] M. Gu and S. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15:1266–1276, 1994.

[32] M. Gu and S. C. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal on Matrix Analysis and Applications*, 15:1266–1276, October 1994.

[33] N. J. Higham. Computing the nearest correlation matrix. *IMA Journal of Numerical Analysis*, 22:329–343, 2002.

[34] N. J. Higham. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[35] R. B. Holmes. On random correlation matrices. *SIAM Journal on Matrix Analysis and Applications*, 12(2):239–272, April 1991.

[36] R. B. Holmes and V. I. Paulsen. Optimal frames for erasures. *Linear Algebra and its Applications*, 377:31–51, 2004.

[37] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

[38] P. Jain, B. Kulis, J. Davis, and I. S. Dhillon. Metric and kernel learning using a linear transformation. *Journal of Machine Learning Research(JMLR)*, 2012. to appear.

[39] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[40] P. Jarratt. A rational iteration function for solving equations. *The Computer Journal*, 9:304–307, 1966.

[41] B. Kulis, M. A. Sustik, and I. S. Dhillon. Learning low-rank kernel matrices. In *Proc. 23rd International Conference on Machine Learning (ICML)*, 2006.

[42] B. Kulis, M. A. Sustik, and I. S. Dhillon. Low-rank kernel learning with Bregman matrix divergences. *Journal of Machine Learning Research*, 10:341–376, 2009.

[43] S. Lang. *Algebra*. Springer Verlag, 3rd, revised edition, 2002.

[44] P.W.H Lemmens and J.J. Seidel. Equiangular lines. *Journal of Algebra*, 24:494–512, 1973.

[45] R.-C. Li. Solving secular equations stably and efficiently. Technical Report UCB/CSD-94-851, EECS Department, University of California, Berkeley, Dec 1994.

[46] Ren-Cang Li. personal communication, 2012.

[47] MATLAB. *version 7.12.0.635 (R2011a)*. The MathWorks Inc., Natick, Massachusetts, 2011.

[48] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[49] A. M. Ostrowski. *Solution of equations in Euclidean and Banach spaces*. Academic Press, 1973.

[50] Wulf Rossman. *Lie Groups an Introduction Through Linear Groups*. Oxford University Press, 2002.

[51] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[52] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimation. *SIAM J. Numer. Anal.*, 17(30):403–409, 1980.

[53] T. Strohmer and R. W. Heath. Grassmannian frames with applications to coding and communication. *Applied and computational harmonic analysis*, 14(3):257–275, May 2003.

[54] M. Sustik, J. A. Tropp, I. S. Dhillon, and R. W. Heath. On the existence of equiangular tight frames. Department. of Computer Sciences TR04–32, University of Texas at Austin, August 2004.

[55] M. A. Sustik, J. A. Tropp, I. S. Dhillon, and R. W. Heath Jr. On the existence of equiangular tight frames. *Linear Algebra and its Applications*, 426:619–635, October 2007.

[56] Mátyás A. Sustik and Inderjit S. Dhillon. On a zero-finding problem involving the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, x(y):u–v, 2012.

[57] J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50:2231–2242, October 2004.

[58] J. A. Tropp, I. S. Dhillon, and Jr. R. W. Heath. Finite-step algorithms for constructing optimal CMDA signature sequences. *IEEE Trans. Inform. Th.*, November 2004.

[59] J. H. van Lint and J. J. Seidel. Equilateral point sets in elliptic geometry. *Proc. Nederl. Akad. Wetensch. Series A*, 69:335–348, 1966. (Reprinted in Indagationes Mathematicae 28:335–348, 1966).

[60] P. Viswanath and V. Anantharam. Optimal sequences and sum capacity of synchronous CDMA systems. *IEEE Trans. Inform. Th.*, 45(6):1984–1991, Sept. 1999.

[61] L. Washington. *Introduction to Cyclotomic Fields.* Number 83 in Graduate Texts in Mathematics. American Mathematical Society, Providence, 2nd edition, 1997.

[62] L. R. Welch. Lower bounds on the maximum cross-correlation of signals. *IEEE Trans. Inform. Th.*, 20:397–399, 1974.