Utrecht University

Master Thesis
Mathematical Sciences

Methods for eigenvalue optimization with application to semidefinite optimization

Author:
Emile Broeders

Supervisor:
Gerard Sleijpen

September, 2015

Abstract

This thesis deals with eigenvalue optimization problems and some methods which are applicable in solving those kind of problems. The goal is to develop a novel algorithm which can solve large instances of eigenvalue optimization problems.

In the second chapter is shown how semidefinite optimization problems can be turned into eigenvalue optimization problems through a rotation of basis. Although the relation between semidefinite optimization and eigenvalue optimization is not new, this approach where a semidefinite optimization problem is turned into a full eigenvalue optimization problem is.

In the third chapter SASPA (Subspace Accelerated Sensitive Pole Algorithm) is used to compute the intersection points of eigencurves. This is a novel application which allows to compute such intersection points with less computation time than through bisection methods. Moreover, SASPA also works for non-simple eigenvalues.

The fourth chapter deals with developing a novel algorithm to solve eigenvalue optimization problems based on gradients. This algorithm is able to handle large sparse problems within reasonable computation time.

Acknowledgement

I am thankful to my supervisor, Gerard Sleijpen, for encouraging and guiding me in the field of numerical mathematics. Due to him I was able to complete this thesis and conduct all research and development of methods included.

Lastly, I offer my regards to all those supported me in any way during the completion of my thesis.

Contents

Chapter 1

Introduction

Applied mathematics deals with all kind of optimization problems originating from different fields. For these kind of problems the objective is to minimize or maximize a function. For instance, the expected profit of a shop or the harvest of a farm. By developing algorithms that can solve these problems we allow companies, government institutions and other organizations to optimize their utilization of scare resources.

Convex optimization problems are problems with a specific structure. They are such that the convex sum of two possible solutions is also a solution to the problem. As an example: Say we want to maximize the profit of the harvest on one acre of land. One possible solution is to fill the land with wheat and another solution is to fill the land with corn. A convex sum of those solutions is for instance to plant wheat on one half of the field and corn on the other, which is another feasible solution to the problem.

The application of large convex optimization problems started in the second half of the twentieth century with the invention of modern computer architectures and development of the simplex algorithm by Dantzig in 1947. The simplex algorithm was developed by Dantzig as part of the United States Air Force team SCOOP (Scientific Computation of Optimum Programs) for optimization of various plans and schedules. The simplex algorithm is an algorithm to solve linear problems (to save words, when linear problems is short for linear convex optimization problems), which are a form of convex optimization problems, in an efficient manner.

Linear problems are such that if the objective value for a solution is x and we can double the solution, then the objective value for the double solution is $2x$. To illustrate this property: Say we expect to have a profit of 1000 euro on the harvest of one acre of corn field, then we expect a profit of 2000 euro on two acres. The simplex algorithm is an algorithm that returns the best solution to such a linear problem, where best means the solution which minimizes or maximizes the objective value of the problem.

In the years following Dantzig's publication of the simplex method, interest in convex optimization problems spurred. More efficient algorithms were

developed to solve linear problems and the framework of linear problems was generalized to cone optimization problems. The class of cone optimization problems consists of linear optimization, quadratic optimization, semidefinite optimization and many more exotic variants. The applications range from classic optimization, polynome fitting, estimation of statistical parameters to approximate NP-hard problems in polynomial time.

I will focus in this thesis on semidefinite optimization. These are optimization problems which can be seen as the unification of linear and quadratic problems. (Quadratic problems are optimization problems in which the objective is quadratic function.) Later will be shown how semidefinite problems are equivalent to eigenvalue optimization of a matrix. This will be used to develop a Newton-like method to solve large instances of semidefinite problems iterative.

In the next sections of this chapter all terms coined so far will be formalized. In section 1.1 a more mathematical description will be given of convex optimization, linear, quadratic and semidefinite problems. Then in section 1.2 will the most important properties of semidefinite optimization be given. In section 1.3 will optimality conditions (also known as Karush-Kuhn-Tucker conditions) and duality be explained.

1.1 Semidefinite optimization

Convex optimization problems are such that the objective is to minimize or maximize a convex function subjected to constraints. In the most general form, all these kind of problems can be written as:

Definition 1.1.1. *For a given convex space $\mathcal{F}$ and convex functions $f, g_1, \ldots, g_m : \mathcal{F} \to \mathbb{R}$, the following equation is known as a convex optimization problem (CP) with only equality constraints:*

$$\sup_{x \in \mathcal{F}} f(x) \text{ s.t. } g_i(x) = 0 \ (\forall i = 1 : m) \tag{1.1}$$

The function f is referred to as the objective and the functions $g_1, \ldots g_m$ are denoted as the constraints.

Moreover, there can also be additional inequality constraints such that:

$$\begin{aligned} &\sup_{x \in \mathcal{F}} f(x) \\ &\text{s.t. } g_i(x) = 0 \quad (\forall i = 1 : m) \\ &\qquad h_i(x) \geq 0 \quad (\forall i = 1 : n) \end{aligned} \tag{1.2}$$

Similar, $h_1, \ldots, h_k$ denote convex functions.

Definition 1.1.2. *The space $\{y \in \mathcal{F} \text{ s.t. } g_i(y) = 0 (\forall i)\}$ is called the feasible set, which is again convex and closed. The interior elements of the feasible set are said to be strict feasible elements.*

There are many variations possible on CP 1.1.1 which all vary in properties on f, g_i and $\mathcal{F}$. For example, probably the best known set of convex optimization problems are known as linear optimization problems (also known as linear programs). That is f and g_i linear and $\mathcal{F} = \mathbb{R}^n$ for some n:

Definition 1.1.3. *A linear program (LP) is an equation of the following form:*

$$\max_{x \in \mathbb{R}^n} c^{\mathrm{T}} x \text{ s.t. } Ax = b,\ x \geq 0 \tag{1.3}$$

With $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

In the definition 1.1.3, $x \geq 0$ should be read as each component of x is larger than or equals 0. Further in this thesis, each time a $\leq$ or $\geq$ is encountered, it should be read as a component wise (in-) equality.

The feasible set of a LP is the intersection of the positive quadrant of $\mathbb{R}^n$ and the affine subspace $Ax = b$.

Sometimes the restriction that $x \geq 0$ is lifted or modified to e.g. $-1 \leq x \leq 1$. Obvious the feasible set is then such that it incorporates the modified constraints.

It might be observed that in the definition of LP the supremum is changed into a maximum. That is because for linear programs the maximum is always attained. This can be quickly recognized: The feasible set is closed and embedded in $\mathbb{R}^n$, which is a complete space. Hence all limit solutions are feasible thus the supremum can be changed without loose in a maximum.

Linear problems can efficiently be solved by the simplex or interior point algorithms and through their general nature many real life optimization problems can be captured into a linear program. For instance in logistics and scheduling problems. Linear programs will be encountered later in this thesis and form part of the backbone of the final algorithm in chapter 4.

As two more examples of convex optimization problems:

Definition 1.1.4. *A quadratic program (QP) is of the form:*

$$\max_{x \in \mathbb{R}^n} x^{\mathrm{T}} Q x \text{ s.t. } Ax = b \tag{1.4}$$

With $Q \in \mathbb{R}^{n \times n}$, and $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Definition 1.1.5. *A quadratically constrained quadratic program (QCQP) is of the form:*

$$\sup_{x \in \mathbb{R}^n} x^{\mathrm{T}} Q x \text{ s.t. } x^{\mathrm{T}} P_i x = b_i \quad (\forall i = 1, \dots m) \tag{1.5}$$

With $Q \in \mathbb{R}^{n \times n}$, and $P_i \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^m$ for all $i = 1, \dots m$.

Problems like 1.1.4 and 1.1.5 arise naturally when dealing with eigenvalue problems. For instance: Finding the largest eigenvalue of a matrix, say M, is equivalent to solving:

$$\sup_{x \in \mathbb{R}^n} x^{\mathrm{T}} M x \text{ s.t. } x^{\mathrm{T}} x = 1 \tag{1.6}$$

This QCQP is better known as the Rayleigh quotient, which will be formalized in corollary 2.2.3.

As another example of CP, and the one which is the central theme in this thesis, is:

Definition 1.1.6. *A semidefinite optimization problem (SDP) is of the form:*

$$\sup_{X \in \mathcal{S}^n} \langle C, X \rangle \text{ s.t. } \langle A_i, X \rangle = b_i \ (\forall i = 1, \dots m),\ X \succeq 0 \tag{1.7}$$

In which $\mathcal{S}^n$ is the space of symmetric real matrices with trace inproduct, $C, A_1, \dots, A_m \in \mathcal{S}^n$ and $b_i \in \mathbb{R}$ for all $i = 1, \dots m$. Furthermore $X \succeq 0$ implies that X is constrained to be positive semidefinite.

The trace inproduct is defined as:

$$\langle A, B \rangle := \mathrm{Tr}(A^{\mathrm{T}} B) = \mathrm{Tr}(A \cdot B) = \mathrm{Tr}(B \cdot A) \tag{1.8}$$

The space $\mathcal{S}^n$ with trace inproduct will be more elaborated introduced in section 1.2. For now it suffices to have an intuitive understanding to see the importance of SDP.

As shorthand, the SDP is often written with linear operator $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$:

$$\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m, X \mapsto \begin{pmatrix} \langle A_1, X \rangle \\ \vdots \\ \langle A_m, X \rangle \end{pmatrix} \tag{1.9}$$

(As an interesting side note, it follows from Riesz representation theorem that any linear operator $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$ is of the form 1.9. Hence SDPs can be written as:

$$\sup_{X \in \mathcal{S}^n} \langle C, X \rangle \text{ s.t. } \mathcal{A}(X) = b,\ X \succeq 0 \tag{1.10}$$

With $C \in \mathcal{S}^n$, $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$ and $b \in \mathbb{R}^m$.)

To demonstrate a feature that makes SDPs powerful: Both LPs and QCQP can easily be cast into SDP. First for a given linear problem: Let $\tilde{C} = \mathrm{diag}(c)$, that is $\tilde{C} \in \mathcal{S}^n$ with on the diagonal vector c and everywhere else 0. Let $\tilde{A}_i = \mathrm{diag}(a_i)$, symmetric matrix with on the diagonal row i of matrix A. Then the SDP version for a given LP is:

$$\begin{aligned} &\max_{X \in \mathcal{S}^n} \langle \tilde{C}, X \rangle \\ &\text{s.t. } \langle \tilde{A}_i, X \rangle = b_i \quad (\forall i = 1, \dots, m) \\ &\qquad X \succeq 0 \end{aligned} \tag{1.11}$$

Stated without proof: The optimum $X^* \in \mathcal{S}^n$ for the SDP is the optimum $x^* \in \mathbb{R}^n$ of the LP on the diagonal: $X^* = \mathrm{diag}(x^*)$.

Second, also QCQP can easily be converted to SDP. Without variable substitutions:

$$\begin{aligned} &\sup_{X \in \mathcal{S}^n} \langle Q, X \rangle \\ &\text{s.t. } \langle P_i, X \rangle = b_i \quad (\forall i = 1, \dots m) \\ &\qquad X \succeq 0 \end{aligned} \tag{1.12}$$

Again, without proof: The optimum $X^* \in \mathcal{S}^n$ for the SDP is the outer product of the optimum $x^* \in \mathbb{R}^n$ of the QCQP: $X^* = x^* {x^*}^{\mathrm{T}}$.

As an example, consider quadratic problem 1.6. The SDP formulation is:

$$\sup_{X \in \mathcal{S}^n} \langle M, X \rangle \text{ s.t. } \langle I_n, X \rangle = 1,\ X \succeq 0 \tag{1.13}$$

With I_n the n by n identity matrix.

Due that the semidefinite formulation can be used easily to combine linear and quadratic aspects among other properties which will be explained in the next section, SDP's are applicable for formulating many problems. Problems where SDP are used range from graph optimization, combinatorial optimization, polynome fitting and many more. Some selected examples are discussed in section 1.4.

1.2 Properties of Frobenius symmetric matrices space

Throughout this thesis many properties of semidefinite matrices are used. It will be shown how many of these properties are used to apply semidefinite optimization in a wide variety of applications in a natural manner.

First will a few matrix decomposition's be stated. After that will a few properties of the trace inproduct be shown. Finally a few basic operators on semidefinite matrices will be shown.

If you feel confident in your knowledge of linear algebra, then you can opt for skipping this section. In this section will mostly elementary results be discussed for easing the explanation of the more advanced results in later sections.

1.2.1 Matrix decomposition's

For a start, some equivalent definitions for positive semidefinite matrices:

Definition 1.2.1. *A matrix $X \in \mathcal{S}^n$ is positive semidefinite if either (hence both) of the following properties hold:*

1. *For all $u \in \mathbb{R}^n$ this relation holds: $u^{\mathrm{T}} X u \geq 0$.*
2. *The smallest eigenvalue of X is non-negative.*

If $X \in \mathcal{S}^n$ is positive semidefinite, then it is said that the inequality $X \succeq 0$ holds and $X \in \mathcal{S}^n_{\succeq 0}$.

The spectral decomposition theorem is very important. It states that any real symmetric matrix can be decomposed in terms of its eigenvalues and eigenvectors:

Theorem 1.2.2 (Spectral decomposition theorem)**.** *Any real symmetric matrix $X \in \mathcal{S}^n$ can be uniquely decomposed as:*

$$X = \sum_{i=1}^{n} \lambda_i u_i u_i^{\mathrm{T}} \tag{1.14}$$

Where $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ are the eigenvalues of X counted according to multiplicity and $u_1, \ldots, u_n \in \mathbb{R}^n$ the corresponding normalized eigenvectors.

This is equivalent to:

$$X = UDU^{\mathrm{T}} \tag{1.15}$$

Where U is orthonormal with the eigenvectors on the columns ($U = (u_1|u_2|\ldots|u_n)$) and D diagonal with on the diagonal the corresponding eigenvalues.

Corollary 1.2.3 (Gram decomposition)**.** *Any matrix $X \in \mathcal{S}^n_{\succeq 0}$ can be decomposed as the sum of at most n orthogonal vectors:*

$$X = \sum_{i=1}^{n} x_i x_i^{\mathrm{T}} \tag{1.16}$$

Where $x_i = \sqrt{\lambda_i} \cdot u_i$, the product of the square root of the ith eigenvalue and ith eigenvector of X.

Also the Cholesky for any real symmetric positive semidefinite matrix is important. With abuse of language, it gives a sense of the square root of a real symmetric positive semidefinite matrix.

Theorem 1.2.4 (Cholesky decomposition)**.** *Any real symmetric positive semidefinite matrix $X \in \mathcal{S}^n_{\succeq 0}$ can be decomposed as the product of real lower triangle matrix L and L^{T}:*

$$X = LL^{\mathrm{T}} \tag{1.17}$$

If $X \succ 0$ (that is positive definite), then, and only then, is that decomposition unique.

Closely related is the LDL-decomposition:

$$X = LDL^{\mathrm{T}} \tag{1.18}$$

With L again lower triangular but with all ones on the main diagonal and D a diagonal matrix.

The reason the LDL-decomposition is sometimes preferred over the LL-decomposition is that for the LDL-decomposition no square roots have to be calculated. Hence its computation is slightly more efficient.

1.2.2 Trace product

Definition 1.2.5 (Trace of a matrix)**.** *The trace is defined as the sum of the elements on the main diagonal of a matrix:*

$$\mathrm{Tr}(X) := X_{1,1} + X_{2,2} + \cdots + X_{n,n} = \sum_{i=1}^{n} X_{i,i} \tag{1.19}$$

Where $(X_{i,j})_{i,j} \in \mathcal{S}^n$*.*

From the definition given above, it should be clear that the trace operator on $\mathcal{S}^n$ is linear:

$$\mathrm{Tr}(aA) = a\,\mathrm{Tr}(A) \qquad (a \in \mathbb{R}, A \in \mathcal{S}^n) \tag{1.20}$$

And:

$$\mathrm{Tr}(A + B) = \mathrm{Tr}(A) + \mathrm{Tr}(B) \qquad (A, B \in \mathcal{S}^n) \tag{1.21}$$

Another important property is that the trace is invariant under cyclic permutations:

$$\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA) \qquad (A, B, C \in \mathcal{S}^n) \tag{1.22}$$

It can be easily deduced from the cyclic property that the trace of a symmetric real matrix equals the sum of its eigenvalues:

Lemma 1.2.6. *The trace of matrix* $X \in \mathcal{S}^n$ *equals the sum of its eigenvalues counted according to its multiplicity:*

$$\mathrm{Tr}(X) = \sum_{i=1}^{n} \lambda_i \tag{1.23}$$

Proof. Using that the eigendecomposition of X is UDU^{T}, with U orthogonal and on the diagonal of D the eigenvalues of X:

$$\mathrm{Tr}(X) = \mathrm{Tr}(UDU^{\mathrm{T}}) = \mathrm{Tr}(U^{\mathrm{T}}UD) = \mathrm{Tr}(D) = \sum_{i=1}^{n} \lambda_i \tag{1.24}$$

□

Definition 1.2.7. *The trace inproduct on* $\mathcal{S}^n$ *is defined as:*

$$\langle A, B \rangle = \mathrm{Tr}(AB) \tag{1.25}$$

It follows from the linearity and cyclic properties of the trace operator on $\mathcal{S}^n$*, that the trace inproduct is bilinear and symmetric.*

An important property of the trace inproduct on $\mathcal{S}^n_{\succeq 0}$ is that it is always non-negative:

Lemma 1.2.8. *For all $A, B \in \mathcal{S}^n_{\succeq 0}$ (symmetric positive semidefinite matrix) holds the relation:*

$$\langle A, B \rangle \geq 0 \tag{1.26}$$

Proof. Let $LL^{\mathrm{T}} = A$ and $MM^{\mathrm{T}} = B$ both be Cholesky decomposition's. Then:

$$\langle LL^{\mathrm{T}}, MM^{\mathrm{T}} \rangle = \mathrm{Tr}(LL^{\mathrm{T}}MM^{\mathrm{T}}) = \mathrm{Tr}(M^{\mathrm{T}}LL^{\mathrm{T}}M) = \mathrm{Tr}(C) \tag{1.27}$$

In which $C = (M^{\mathrm{T}}L)(M^{\mathrm{T}}L)^{\mathrm{T}}$. Clearly $C \succeq 0$ since its Cholesky decomposition exists. □

1.2.3 Basic operations

The semidefinite property is invariant under permutations:

Lemma 1.2.9. *Let $X \in \mathcal{S}$ and $P \in \mathbb{R}^{n \times n}$ be non-singular. Then:*

$$X \succeq 0 \Leftrightarrow P^{\mathrm{T}}XP \succeq 0 \tag{1.28}$$

Proof. Assume $X \succeq 0$. Let $X = UDU^{\mathrm{T}}$ be the eigendecomposition of X. Then $P^{-\mathrm{T}}U$ forms a basis for the eigenvectors of $P^{\mathrm{T}}XP$, hence the eigenvalues corresponding to the i column of $P^{\mathrm{T}}XP$ is $D_{i,i}$ plus the square root of the norm of that column. Thus definitely non-negative and both sides are positive semidefinite.

For the other way around: observe that $X = P^{-\mathrm{T}}(P^{\mathrm{T}}XP)P^{-1}$. □

The following definition is widely used:

Definition 1.2.10 (Schur complement)**.** *Consider matrix $X \in \mathcal{S}^n$ in block form:*

$$X = \begin{pmatrix} A & B \\ B^{\mathrm{T}} & C \end{pmatrix} \tag{1.29}$$

With $A \in \mathcal{S}^n$ non-singular. Then the matrix $C - B^{\mathrm{T}}A^{-1}B$ is called the Schur complement of A in X.

Theorem 1.2.11. *Let $X \in \mathcal{S}^n$ be in block form as in equation 1.2.10. X is non-singular if and only if A and $C - B^{\mathrm{T}}A^{-1}B$ are both non-singular.*

Proof. Define:

$$P = \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} \tag{1.30}$$

Then $X = P^{\mathrm{T}} \begin{pmatrix} A & 0 \\ 0 & C - B^{\mathrm{T}}A^{-1}B \end{pmatrix} P$. Clearly P is non-singular, thus X is only singular iff $\begin{pmatrix} A & 0 \\ 0 & C - B^{\mathrm{T}}A^{-1}B \end{pmatrix}$ is singular, which is only if either A is singular or $C - B^{\mathrm{T}}A^{-1}B$ is singular. □

Corollary 1.2.12. *Let $X \in \mathcal{S}^n$ be in block form as in equation 1.2.10. Then:*

$$X \succeq 0 \Leftrightarrow A \succeq 0 \text{ and } C - B^{\mathrm{T}}A^{-1}B \succeq 0 \tag{1.31}$$

Proof. Note that as in theorem 1.2.11: $X = P^{\mathrm{T}} \begin{pmatrix} A & 0 \\ 0 & C - B^{\mathrm{T}}A^{-1}B \end{pmatrix} P$. The result follows direct from lemma 1.2.9. □

1.3 Duality and optimality conditions

Duality theory allows us to derive optimality conditions for problems such as 1.1.3, 1.1.4, and 1.1.6. Moreover, duality theory allows to recast the problem into a corresponding dual problem which might be easier to compute than the original 'primal' problem.

To derive the dual problem, consider first the Lagrangian:

Definition 1.3.1 (Lagrangian)**.** *The Lagrangian of a convex problem (i.e. 1.1.1), with only equality constraints is defined as:*

$$\mathcal{L} : \mathcal{F} \times \mathbb{R}^m \to \mathbb{R}, (x, y) \mapsto f(x) + \sum_i^m y_i g_i(x) \tag{1.32}$$

If there are in addition inequality constraints:

$$\mathcal{L} : \mathcal{F} \times \mathbb{R}^m \times \mathbb{R}^k_+ \to \mathbb{R}, (x, y, z) \mapsto f(x) + \sum_i^m y_i g_i(x) + \sum_i^k z_i h_i(x) \tag{1.33}$$

The variables (y, z) are known as Lagrange multipliers.

Note that the Lagrangians allow to write convex problems as an one liner:

$$p^* = \max_{x \in \mathcal{F}} \min_{y \in \mathbb{R}^m, z \in \mathbb{R}^k_+} \mathcal{L}(x, y, z) \tag{1.34}$$

Equation 1.34 is known as the primal problem of a convex optimization problem. The dual is obtained by interchanging the max–min:

$$d^* = \min_{y \in \mathbb{R}^m, z \in \mathbb{R}^k_+} \max_{x \in \mathcal{F}} \mathcal{L}(x, y, z) \tag{1.35}$$

From now on p^* and d^* will denote the optimum objective value of the primal and the dual problem. Similar x^* and y^* are used to denote the argument for which the optimum is attained.

Example 1.3.2. *The Lagrangian of 1.1.3 is:*

$$\mathcal{L}(x, y, z) = c^{\mathrm{T}}x + y^{\mathrm{T}}(b - Ax) + z^{\mathrm{T}}x \tag{1.36}$$

This can be rewritten as:

$$\mathcal{L}(x, y, z) = y^{\mathrm{T}} b + (c^{\mathrm{T}} - y^{\mathrm{T}} A + z^{\mathrm{T}})x \tag{1.37}$$

Hence the dual of 1.1.3 is:

$$\begin{aligned} \min_{y \in \mathbb{R}^m, z \in \mathbb{R}^n} \; & b^{\mathrm{T}} y \\ \text{s. t. } & c - A^{\mathrm{T}} y + z = 0 \\ & z \geq 0 \end{aligned} \tag{1.38}$$

The more familiar dual form is obtained by removing z from the problem:

$$\begin{aligned} \min_{y \in \mathbb{R}^m} \; & b^{\mathrm{T}} y \\ \text{s. t. } & c - A^{\mathrm{T}} y \leq 0 \end{aligned} \tag{1.39}$$

It is possible to derive a dual program for a semidefinite program as well through a similar procedure as in the previous example:

Example 1.3.3. *The Lagrangian of 1.1.6 is:*

$$\mathcal{L}(X, y, Z) = \langle C, X \rangle + y^{\mathrm{T}}(b - \mathcal{A}(X)) + \langle Z, X \rangle \tag{1.40}$$

This can be rewritten as:

$$\mathcal{L}(x, y, z) = y^{\mathrm{T}} b + \langle C - \mathcal{A}^{\mathrm{T}}(y) + Z, X \rangle \tag{1.41}$$

In which $\mathcal{A}^{\mathrm{T}}(y) = \sum_i y_i A_i$. Note that by this definition:

$$\langle \mathcal{A}(X), y \rangle = \langle X, \mathcal{A}^{\mathrm{T}}(y) \rangle \tag{1.42}$$

The dual of 1.1.6 is:

$$\begin{aligned} \inf_{y \in \mathbb{R}^m, Z \in \mathcal{S}^n} \; & b^{\mathrm{T}} y \\ \text{s. t. } & C - \mathcal{A}^{\mathrm{T}}(y) + Z = 0 \\ & Z \succeq 0 \end{aligned} \tag{1.43}$$

And by removing Z:

$$\begin{aligned} \inf_{y \in \mathbb{R}^m} \; & b^{\mathrm{T}} y \\ \text{s. t. } & C - \mathcal{A}^{\mathrm{T}}(y) \preceq 0 \end{aligned} \tag{1.44}$$

Already from observing that in the primal problem the objective is maximized, and in the dual the objective is minimized, it follows that the primal objective is less than the dual objective. This is proven for semidefinite optimization problems:

Theorem 1.3.4 (Weak duality for SDP)**.** *For any feasible primal solution $X \in \mathcal{S}^n$ (not necessary optimal) and any feasible dual solution $y \in \mathbb{R}^m$ (also not necessary optimal) holds the following inequality:*

$$\langle C, X \rangle \leq b^{\mathrm{T}} y \tag{1.45}$$

Proof. Note that any primal feasible solution $X \in \mathcal{S}^n$ implies $X \succeq 0$ and any dual feasible solution $y \in \mathbb{R}^m$ implies $C - \mathcal{A}^{\mathrm{T}}(y) \preceq 0$. Hence:

$$\langle C, X \rangle - y^{\mathrm{T}} b = \langle C, X \rangle - y^{\mathrm{T}} \mathcal{A}(X) = \langle C - \mathcal{A}^{\mathrm{T}}(y), X \rangle \leq 0 \tag{1.46}$$

□

The quantity $|\langle C - \mathcal{A}^{\mathrm{T}}(y), X \rangle|$ at optimum $X \in \mathcal{S}^n$ and $y \in R^m$ is known as the duality gap. When it is said that there is no duality gap, then it is implied that the duality gap equals zero. Consequently, if a given primal and dual solution have no duality gap, then the solutions are optimal.

As a side note, if the primal problem is unbounded, i.e. $p^* = \infty$, then by weak duality $d^* \geq \infty$, hence infeasible. Similar if the dual is unbounded, then the primal is infeasible.

In many cases strong duality holds, i.e. there is no duality gap. This stated here without proof:

Theorem 1.3.5 (Strong duality)**.** *If the primal problem is bounded, that is the optimum is finite i.e. $p^* < \infty$, and strictly feasible, then the dual problem attains its infimum and there is no duality gap.*

Proof. See [5]. □

Corollary 1.3.6. *If the dual problem is bounded i.e. $d^* > -\infty$, and strictly feasible, then the primal problem attains its supremum and there is no duality gap.*

By strong duality it follows that for optimum (X^*, y^*) the following equality holds:

$$\langle C - \mathcal{A}^{\mathrm{T}}(y^*), X^* \rangle = 0 \tag{1.47}$$

This equation is known as the complementary slackness condition.

If the dual solution y^* is known, and strong duality holds, then calculating X^* is equivalent to solving a lower dimensional problem than the original primal by noting that the space spanned by $C - \mathcal{A}^{\mathrm{T}}(y)$ should be in the null-space of X^*:

$C - \mathcal{A}^{\mathrm{T}}(y) \preceq 0$, a Cholesky decomposition $LL^{\mathrm{T}} = \mathcal{A}^{\mathrm{T}}(y) - C$ exists. By complementary slackness:

$$\begin{aligned} 0 &= \langle LL^{\mathrm{T}}, X^* \rangle \\ &= \langle I, L^{\mathrm{T}} X^* L \rangle \end{aligned} \tag{1.48}$$

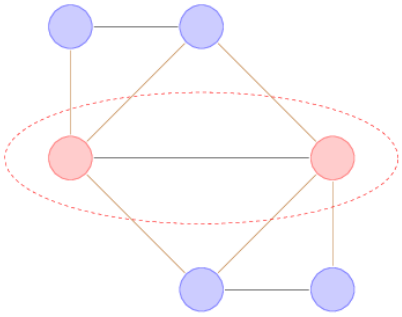

Figure 1.1: An example of a non-unique max-cut

And also $X^* \succeq 0$, hence the columns of L have to be in the null-space of X^*. Let Q has as columns an orthogonal basis perpendicular to the columns of L. Obvious Q has less columns than n; say it has k columns which form a basis. Then by the variable substitution $X = QUQ^{\mathrm{T}}$, problem 1.1.6 is reduced to the space $\mathcal{S}^k$.

In practice k is a small number and thus problem 1.1.6 can be solved quite fast.

1.4 Some examples

In this section four examples of semidefinite optimization problems are given. These include from graph theory: The famous Goemans-Williamson cut for solving max-cut instances [12]. The Lovász number, which gives an upper bound to the clique number and a lower bound to the chromatic number [17]. One from linear algebra: calculating ideal GMRES and Arnoldi polynomes for symmetric matrices (based on [13]). And one based on phase retrieval via matrix completion (based on [6]).

1.4.1 Max-cut problem

The max-cut problem for graphs is defined as the problem of finding the largest possible cut for a graph. That is, one wants to separate all nodes of a graph into two disjoint sets such that the number of edges between the two sets is as large as possible.

The corresponding decision problem, that is whether for a given graph G there exists a cut of at least k with k arbitrary large, is NP-complete. This problem is one of Karp's original NP-complete problems [15]. Hence it is probably impossible to derive an exact algorithm to solve the problem which runs within a reasonable time. However Goemans and Williamson accomplished to derive an 0.87856-approximation algorithm (that is, the algorithm always delivers an answer that at least is 0.87856 times the optimal value) which runs

in polynomial time using semidefinite optimization. The resulting algorithm is known as the Goemans-Williamson max-cut algorithm.

In the original paper in which Goemans and Williamson describe their approximation algorithm based on semidefinite optimization, they considered G directed and weighted [12]. However because the problem is stated here only as an example of semidefinite optimization and those details make it unnecessary more complex, those details are omitted. The algorithm described in the rest of this subsection is the specific case of the Goemans-Williamson max-cut for undirected uniform weighted graphs.

In the remainder of this subsection it is assumed that $G = (V, E)$ (graph G with nodes / vertexes V and edges E) is undirected. That is if $(i, j) \in E$ then also $(j, i) \in E$.

Definition 1.4.1. *A cut on $G = (V, E)$ is a partition of V into two sets $S, \bar{S}$ such that $S \cap \bar{S} = \phi$ and $S \cup \bar{S} = V$. The size of the cut is defined as the number of edges starting in S and ending in $\bar{S}$, that is: $C(G, S) := |\{(i, j) \in E \text{ s.t. } i \in S, j \in \bar{S}\}|$.*

An example of a max-cut is given in figure 1.1.

The main idea of the Goemans-Williamson algorithm is to associate with every vertex $i \in V$ a binary variable $x_i \in \{1, -1\}$ which denotes whether $i \in S$ or $i \in \bar{S}$. Then for two vertices $i, j \in V$:

$$\frac{1 - x_i x_j}{2} = \begin{cases} 0 & \text{if } i, j \in S \text{ or } i, j \in \bar{S} \\ 1 & \text{if } i \in S, j \in \bar{S} \text{ or } i \in \bar{S}, j \in S \end{cases} \tag{1.49}$$

Hence the value for a given cut is:

$$C(G, S) = \frac{1}{2} \sum_{(i,j) \in E} \frac{1 - x_i x_j}{2} \tag{1.50}$$

(The fraction $\frac{1}{2}$ at the beginning arises because $(i, j) \in E$ implies $(j, i) \in E$, hence all edges are counted twice.)

And the max cut is thus:

$$C_{\max}(G) = \max_{x \in \mathbb{R}^{|V|}} \frac{1}{2} \sum_{(i,j) \in E} \frac{1 - x_i x_j}{2} \qquad \text{s.t. } x_i \in \{-1, +1\} \qquad (\forall i = 1, \ldots, |V|) \tag{1.51}$$

Therefore equation 1.51 has to be solved, however the constraint that all x_i is a binary variable makes it difficult. Hence consider the relaxation to write it

as a semidefinite program:

$$\begin{aligned} C_{\mathrm{GW}}(G) = \max_{X \in \mathcal{S}^{|V|}} & \frac{1}{4} \sum_{(i,j) \in E} (1 - X_{i,j}) \\ \text{s. t. } & X_{i,i} = 1 \ \ (\forall i \in V) \\ & X \succeq 0 \end{aligned} \tag{1.52}$$

In the ideal case the $X^* \in \mathcal{S}^n$ that maximizes problem 1.52 can be decomposed as $X^* = xx^{\mathrm{T}}$ with $x_i = \pm 1$ for all i. However, because C_{GW} is a relaxation it is to be expected to find a better solution than X^* which does not permit such a decomposition. The great accomplishment of Goemans-Williamson is that they showed:

Theorem 1.4.2. *Given a undirected graph G with uniform weights, the following inequalities hold:*

$$C_{\mathrm{GW}}(G) \geq C_{\max}(G) \geq 0.87856 C_{\mathrm{GW}}(G) \tag{1.53}$$

The full proof can be found in [12]. The proof is more algorithmic and based on the following steps:

1. Solve X for equation 1.52.
2. Perform a Cholesky decomposition $X = VV^{\mathrm{T}}$ such that $X_{i,j} = v_i^{\mathrm{T}} v_j$ (columns of V^{T})
3. Select a random unit vector r and define a cut $S = \{i \text{ s. t. } r^{\mathrm{T}} v_i \geq 0\}$
4. Check if $C(G, S) \geq 0.87856 C_{\mathrm{GW}}(G)$, if not, go back to step 3

The number 0.87856 appears in the construction of S (step 3) as the result of some sort of randomized rounding the 'fractional' solution to fit the original problem statement.

In order to derive the dual of program 1.52 first rewrite it to a standard semidefinite problem. Note that the the sum in the objective can be rewritten as: $\sum_{(i,j) \in E} (1 - X_{i,j}) = |E| - \langle X, A_G \rangle = \langle X, L_G \rangle$; that is A_G as the adjacency matrix and L_G the Laplacian matrix which are defined as:

$$\begin{aligned} (A_G)_{i,j} &= \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \\ (L_G)_{i,j} &= \begin{cases} -1 & \text{if } (i,j) \in E \\ \deg(i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{1.54}$$

(Here $\deg(i)$ is the degree of vertex i, i.e. $\deg(i) = |\{(i,j) \in E\}|$.)

And the constraint $X_{i,i} = 1$ can be reworded using $E_{i,i} = e_i e_i^{\mathrm{T}}$: $X_{i,i} = \langle X, E_{i,i} \rangle = 1$.

This makes it easy to write the primal and dual pair:

$$\begin{aligned} C_{\mathrm{GW}}(G) = \max_{X \in \mathcal{S}^{|V|}} & \frac{1}{4} \langle L_G, X \rangle \\ \text{s. t. } & \langle X, E_{i,i} \rangle = 1 \ \ (\forall i \in V) \\ & X \succeq 0 \end{aligned} \tag{1.55}$$

$$\begin{aligned} C^*_{\mathrm{GW}}(G) = \min_{y \in \mathbb{R}^{|V|}} & \sum_i y_i \\ \text{s. t. } & \frac{1}{4} L_G - \sum_i y_i E_{i,i} \preceq 0 \end{aligned} \tag{1.56}$$

1.4.2 Lovász theta number

The Lovász theta number was introduced in 1979 as an upper bound on the Shannon capacity of a graph [17]. Lovász theta number is lower bounded by the clique number of the complement graph, and upper bounded by the chromatic number of its complement [16]. Hence it is an important number for the analysis of graphs.

The clique number and the chromatic number for undirected graphs are NP-hard to compute, hence the Lovász number is interesting because it can be computed in polynomial time by using semidefinite optimization. Moreover, at the time of writing there is no algorithm yet known to compute the Shannon capacity of a graph and even for small graphs remains the Shannon capacity unknown. However, it can be easily be seen as lower bounded by the clique number. Thus if the clique number and the Lovász theta number coincide then the Shannon capacity is equal to those numbers. Lovász used this to calculate the Shannon capacity of the pentagon graph (cyclic, 5-vertices) to be $\sqrt{5}$, which was still unknown at the time [17].

In this section the Lovász theta number is presented as a semidefinite program, in the original article this is just one of the representations presented. In this section the many other representations will be omitted and stick to the semidefinite program because that is the one actually used later in chapter 5.

Definition 1.4.3. *In an undirected graph $G = (V, E)$, a clique $C \subset V$ is a set of vertices that are all adjacent to each other. A maximal clique is a clique that cannot be extended by adding another vertex.*

The clique number of a graph is the size of its largest clique, which is an upper bound to the size of all possible maximal cliques. The clique number for graph G is denoted as $\omega(G)$

Definition 1.4.4. *A coloring of a graph $G = (V, E)$ is a labeling of the vertexes V such that no two adjacent vertices are labeled the same. A k-coloring is a*

partition of V into k sets, $C_1, \dots C_k$, such that each set is independent, i.e. no two vertices in C_i are adjacent.

The chromatic number of a graph is a lower bound on the labels of a coloring, that is the smallest number of labels necessary to color the graph. The chromatic number for graph G is denoted as $\chi(G)$.

Definition 1.4.5. *The complement of a graph $G = (V, E)$ is $\bar{G} = (V, \bar{E})$. That is, in $\bar{G}$ two vertices are adjacent if and only if those two vertices are not-adjacent in G.*

Definition 1.4.6. *The Lovász number for a graph $G = (V, E)$ is defined as [17]:*

$$\begin{aligned} \theta(G) = \max_{B \in \mathcal{S}^{|V|}} & \langle J, B \rangle \\ \text{s. t. } & \langle B, I \rangle = 1 \\ & \langle B, E_{ij} \rangle = 0 \quad (\forall i, j \in V \text{ adjacent}) \\ & B \succeq 0 \end{aligned} \tag{1.57}$$

In which J is the all ones matrix and E_{ij} is the matrix: $e_i e_j^{\mathrm{T}} + e_j e_i^{\mathrm{T}}$.

The famous sandwich theorem:

Theorem 1.4.7. *For any undirected graph $G = (V, E)$ holds:*

$$\omega(G) \leq \theta(\bar{G}) \leq \chi(G) \tag{1.58}$$

Proof. First $\omega(G) \leq \theta(\bar{G})$:

Let $x \in \mathbb{R}^{|V|}$ be the characteristic vector for a maximal clique C in G (i.e. if $i \in C$, $x_i = 1$ else $x_i = 0$) of size $\omega(G)$. Then a feasible solution to 1.57 for $\bar{G}$ is:

$$B' = \frac{1}{\omega(G)} x x^{\mathrm{T}} \tag{1.59}$$

Moreover, $\langle J, B' \rangle = \omega(G)$, hence $\theta(\bar{G}) \geq \omega(G)$.

Second $\theta(\bar{G}) \leq \chi(G)$:

Let B be any feasible matrix for equation 1.57; then $\langle J, B \rangle \leq \chi(G)$: Let $C_1, \dots, C_k$ be a k-coloring such that $k = \chi(G)$. Let $y_1, \dots, y_k$ be the characteristic vectors of $C_1, \dots, C_k$. Moreover, define:

$$Y = \sum_i^k (k y_i - e)(k y_i - e)^{\mathrm{T}} = k^2 \sum_i^k y_i y_i^{\mathrm{T}} - kJ \tag{1.60}$$

(e is the all ones vector.)

Note that $Y \succeq 0$, hence $\langle B, Y \rangle \geq 0$. Also:

$$\begin{aligned} \langle B, Y \rangle &= \langle B, k^2 \sum_i y_i y_i^{\mathrm{T}} - kJ \rangle \\ &= k^2 \langle B, I \rangle - k \langle B, J \rangle \\ &= k^2 - k \langle B, J \rangle \geq 0 \end{aligned} \tag{1.61}$$

Hence for any feasible B of equation 1.57: $\langle J, B \rangle \leq \chi(G)$. □

The dual of 1.57 is given by:

$$\begin{aligned} &\min_{t \in \mathbb{R}, y \in \mathbb{R}^{|E|}} t \\ &\quad \text{s. t. } J - \mathcal{E}^{\mathrm{T}}(y) - tI \preceq 0 \end{aligned} \tag{1.62}$$

In which $\mathcal{E}^{\mathrm{T}}(y) = \sum_{(i,j) \in E} E_{ij} y_{ij}$.

Clearly this is already an eigenvalue optimization problem because t is minimal if and only if its equal to the maximum eigenvalue:

$$\min_{y \in \mathbb{R}^{|E|}} \lambda_{\max}(J - \mathcal{E}^{\mathrm{T}}(y)) \tag{1.63}$$

1.4.3 Ideal GMRES and Arnoldi polynomes

The generalized minimal residual method (GMRES) is an iterative Krylov subspace method for numerical solving $Ax = b$ introduced by Saad and Schultz in 1986 [29]. Since its development there has been active research to convergence bounds of the algorithm and is still today considered a difficult open problem [31].

During GMRES the solution to $Ax = b$ is solved through a least squares restricted to a Krylov subspace. In each iteration the subspace is extended with one orthogonal vector, until the solution of the least squares is an exact solution. The Krylov subspace which is constructed during the algorithm up to the kth iteration is:

$$\mathcal{K}_k(A, Ab) = \mathrm{span}\{Ab, A^2 b, \ldots, A^k b\} \tag{1.64}$$

Let x_k be the k iterative step of GMRES and $r_k := Ax_k - b$. Then in GMRES:

$$\|r_k\|_2 = \min_{q \in P_k} \|q(A) r_0\|_2 \tag{1.65}$$

In which P_k is the set of all polynomials of degree at most k with for all $q \in P_k$, $q(0) = 1$.

Note that the equation above is a least squares problem in disguise, $\{Aq(A)b : q \in P_k\}$ spans exactly the Krylov subspace $\mathcal{K}_k(A, Ab)$.

An easy upper bound on convergence of GMRES is:

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \min_{q \in P_k} \|q(A)\|_2 \tag{1.66}$$

For a more thorough description of the algorithm and implementation details, see [29].

A similar iterative algorithm is Arnoldi iteration (developed by Arnoldi in 1951 [2]) which is used to compute eigenvalues of matrix A.

During Arnoldi iteration the matrix A is projected on the Krylov subspace $\mathcal{K}_k(A, b)$ (b a random unit vector). That is, let Q_k be a basis for $\mathcal{K}_k(A, b)$ and define $H_k = Q_k^{\mathrm{T}} A Q_k$. Then as k increases, and the Krylov subspace keeps expending, the matrix H_k will become more similar to A. Then the Ritz values of H_k converge to the eigenvalues of A.

It can be shown that the characteristic polynome of H_k is a monic polynome of degree k that minimizes $\|q(A)b\|_2$:

$$\min_{q \in P^k} \|q(A)b\|_2 \tag{1.67}$$

In which P^k is the set of all monic polynomials of degree k [13]. For a more inclusive description see [2].

As with GMRES, an easy upper bound on convergence is given by:

$$\min_{q \in P^k} \|q(A)\|_2 \tag{1.68}$$

The $q \in P_k$ which minimizes equation 1.66 is known as the ideal GMRES polynome and the $q \in P^k$ which minimizes equation 1.68 is known as the ideal Arnoldi polynome.

The motivation of calculating the ideal GMRES and Arnoldi polynomes is that they give an upper bound on convergence and can also be applied to testing different preconditioning strategies. The computation of the ideal GMRES/Arnoldi polynome can be formulated as a semidefinite optimization program through the following identity:

Lemma 1.4.8. *The matrix 2-norm for matrix $A \in \mathcal{S}^n$ is the largest absolute eigenvalue of A. That is:*

$$\|A\|_2 = \max_i |\lambda_i| \tag{1.69}$$

Where λ_i is a distinct labeled eigenvalue of A. Furthermore the following equality holds:

$$\|A\|_2 = \lambda_{\max}\left(\begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix}\right) \tag{1.70}$$

Proof. Let $Z = \begin{pmatrix} 0 & A \\ A & 0 \end{pmatrix}$, denote the set of eigenvalues of Z as $\lambda(Z)$ and the eigenvalues of A as $\lambda(A)$. Then $\lambda(Z) = \pm\lambda(A)$. To see that: Label all eigenvectors of A as q_i such that $q_i^{\mathrm{T}} A q_i = \lambda_i$ and $q_i^{\mathrm{T}} q_j = \delta_{i,j}$ (i.e. 0 if $i \neq j$ and 1 if $i = j$). Such a decomposition is possible by the spectral decomposition theorem (theorem 1.2.2).

Then $(q_i, \pm q_i)^{\mathrm{T}}$ forms two eigenvectors of Z with eigenvalues $\pm\lambda_i$. Hence because A has n eigenvalues and this way $2n$ eigenvalues for Z (which is of size $2n$) are created, it follows that Z cannot have any other eigenvalue.

Therefore the largest eigenvalue of Z is the largest absolute eigenvalue of A. □

The computation of the ideal GMRES polynome of degree k is:

$$\begin{aligned} &\min_{t\in\mathbb{R},\, y\in\mathbb{R}^k} t \\ &\text{s. t. } \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} + \sum_{1\leq i\leq k} y_i \begin{pmatrix} 0 & A^i \\ (A^i)^{\mathrm{T}} & 0 \end{pmatrix} - t \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \preceq 0 \end{aligned} \tag{1.71}$$

And similar the ideal Arnoldi polynome of degree k:

$$\begin{aligned} &\min_{t\in\mathbb{R},\, y\in\mathbb{R}^k} t \\ &\text{s. t. } \begin{pmatrix} 0 & A^k \\ (A^k)^{\mathrm{T}} & 0 \end{pmatrix} + \sum_{1\leq i\leq k} y_i \begin{pmatrix} 0 & A^{i-1} \\ (A^{i-1})^{\mathrm{T}} & 0 \end{pmatrix} - t \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \preceq 0 \end{aligned} \tag{1.72}$$

1.4.4 Phase retrieval

This subsection is largely based on [6].

Phase retrieval is the problem of reconstructing a signal based on the magnitude of its Fourier transform. This problem arises in many applications due to that detectors often can only detect the squared signal.

Historically one of the first applications of phase retrieval is X-ray crystallography. In X-ray crystallography are X-rays aimed at a crystal and the angle of dispersion and the intensity is measured. From those measurements a 3D-image can be reconstructed with the density of the electrons within the crystal.

Moreover, different applications of Phase Retrieval include MRI, astronomical imaging and microscopy.

To fix the problem statement, consider a discrete one dimensional signal of size n: $x_0, \ldots, x_{n-1} \in \mathbb{R}$ (notice the indexing starts with 0). Its Fourier transform is given by:

$$\tilde{x}(\omega) = \frac{1}{\sqrt{n}} \sum_t x_t \exp(-i2\pi\omega t/n), \quad \omega \in \Omega \tag{1.73}$$

The set Ω is a grid of sampled frequencies. The usual sampled frequencies are $\{0, \ldots, n-1\}$.

The problem of Phase Retrieval is reconstructing x from the magnitude measurements $|\tilde{x}(\omega)|$, $\omega \in \Omega$. Note that the problem is ill-posed because there is no unique solution. If $x \in \mathbb{R}^n$ is a solution the problem, then as an example, $-x$ is also a solution. In general there are infinite many solutions. However from a practical point, all these solutions are acceptable ambiguities.

Observe that $\tilde{x}(\omega)$ can be written as $a_\omega^* x$ in which $a_\omega = \frac{1}{\sqrt{n}} \times (1, \exp(-i2\pi\omega 1/n), \ldots)^*$. Hence:

$$\begin{aligned} |\tilde{x}(\omega)|^2 &= |\langle a_\omega, x \rangle|^2 \\ &= \langle a_\omega a_\omega^*, x x^* \rangle \end{aligned} \tag{1.74}$$

Denote $A_\omega = a_\omega a_\omega^*$.

As for the Max-Cut, the quadratic constraint $\langle A_\omega, x x^* \rangle$ can be relaxed by optimizing over the space of positive semidefinite symmetric matrices and substitute $X = x x^*$.

The SDP problem is a feasibility problem:

$$\begin{aligned} \text{find}_{X \in \mathcal{S}^n} \; & X \\ \text{s.t. } & \langle A_\omega, X \rangle = b_\omega \;\; (\omega \in \Omega) \\ & X \succeq 0 \\ & \text{rank}(X) = 1 \end{aligned} \tag{1.75}$$

($b_\omega = |\tilde{x}(\omega)|^2$)

The constraint on the rank of X is a non-convex constraint. A possibility to lift the constraint is by replacing it by a trace constraint: $\langle I, X \rangle = 1$. Moreover, as a further relaxation it can be incorporated into the objective such that the problem can be solved by standard SDP solving algorithms:

$$\begin{aligned} \min_{X \in \mathcal{S}^n} & \langle I, X \rangle \\ \text{s.t. } & \langle A_\omega, X \rangle = b_\omega \;\; (\omega \in \Omega) \\ & X \succeq 0 \end{aligned} \tag{1.76}$$

Through these relaxations the resulting SDP offers a large degree of freedom to deal with noise and other errors made by abstracting the problem. That can be for instance be done by replacing the equality constraint with an upper and lower bound, or by using some maximum likelihood principle as was done in [6].

After obtaining an optimum $X \in \mathcal{S}^n$ of sufficient low rank, the original vector $x \in \mathbb{R}^n$ can be found through performing a Cholesky decomposition.

If X is not of sufficient low rank, then it is possible to promote in equation 1.76 solutions of lower rank through solving for a sequence of weighted trace

norms. That works as follows: Choose $\varepsilon > 0$ and $W_0 = I$. Next solve for $k = 0$:

$$\begin{aligned} \min_{X_k \in \mathcal{S}^n} & \langle W_k, X_k \rangle \\ \text{s. t. } & \langle A_\omega, X_k \rangle = b_\omega \ \ (\omega \in \Omega) \\ & X_k \succeq 0 \end{aligned} \tag{1.77}$$

Next, update $W_{k+1} = (X_k + \varepsilon I)^{-1}$ and solve 1.77 for $k = k + 1$. Repeat until X_k is of sufficient low rank or k reaches a threshold $k_{\max}$.

As described in [10], this scheme can be viewed as an iterative algorithm to solve:

$$\begin{aligned} \min_{X \in \mathcal{S}^n} & \log(\det(X + \varepsilon I)) \\ \text{s. t. } & \langle A_\omega, X \rangle = b_\omega \ \ (\omega \in \Omega) \\ & X \succeq 0 \end{aligned} \tag{1.78}$$

Hence, the iterative method is known as the log det heuristic.

The dual of 1.76 is:

$$\begin{aligned} - \min_{y \in \mathbb{R}^{|\Omega|}} & b^{\mathrm{T}} y \\ \text{s. t. } & -I - \mathcal{A}^{\mathrm{T}}(y) \preceq 0 \end{aligned} \tag{1.79}$$

Or by substituting $y' = -y$:

$$\begin{aligned} \max_{y \in \mathbb{R}^{|\Omega|}} & b^{\mathrm{T}} y \\ \text{s. t. } & I - \mathcal{A}^{\mathrm{T}}(y) \succeq 0 \end{aligned} \tag{1.80}$$

Note that $y = 0$ is a strict feasible dual solution. Hence if the problem is bounded and the optimum attained, then strong duality holds.

Solving the dual might not seem worthwhile because of the necessity of the primal solution for recovering the signal. However, as remarked at the end in section 1.3, if strong duality holds and the dual solution y^* is known, then the primal problem can be shown to be equivalent to a low dimensional problem.

Moreover, by solving the primal as well as the dual in the log det heuristic, a near optimum feasible solution for the next iteration is already obtained. This will be elaborated on in section 2.1.2.

Chapter 2

Eigenvalue optimization

In this chapter it will be shown how many semidefinite optimization problems can be recast into an eigenvalue optimization problem. That are problems in which the objective is to maximize or minimize the eigenvalue of a matrix through linear perturbations. The section after derives some important properties of eigenvalue problems which will be used in solving the problems.

2.1 Equivalence of semidefinite problems

It is possible to derive for most semidefinite problems an eigenvalue optimization problem. That is a problem in which the eigenvalues of a matrix operator are minimized or maximized. To do so, first perform a variable rotation, second apply a Cholesky decomposition and third conclude that the corresponding problem can be interpreted as en eigenvalue optimization problem.

Through the rest of this section the following problem will be recast into an eigenvalue optimization problem:

$$\begin{aligned} &\inf_{y\in\mathbb{R}^m} b^{\mathrm{T}}y \\ &\text{s.t. } C-\mathcal{A}^{\mathrm{T}}(y)\preceq 0 \\ &\qquad Py\geq 0 \end{aligned} \tag{2.1}$$

In which Py is the vector with components of y that correspond to inequalities. This equation is similar to equation 1.44.

We want to recast problem 2.1 into a problem of the following form:

$$\begin{aligned} &\inf_{z\in\mathbb{R}^k} \lambda_{\max}(\tilde{C}-\tilde{\mathcal{A}}^{\mathrm{T}}(z)) \\ &\text{s.t. } \tilde{P}^{\mathrm{T}}z\geq 0 \end{aligned} \tag{2.2}$$

The reason being is that eigenvalue problems such as 2.2 can efficiently be solved by an eigenvalue optimize algorithm, which will be the subject for the rest of this thesis.

To do so apply a variable transformation: Define orthogonal square matrix Q of full rank such that $z = Q^{\mathrm{T}}y$ and $b^{\mathrm{T}}y = \|b\|_2 z_1$ holds for all y. This implies that:

$$\begin{aligned} Q &= (\ q_1 \mid q_2 \mid q_3 \mid \ \ldots \ \mid q_m\ ) \\ q_1 &= \hat{b} \end{aligned} \tag{2.3}$$

In which $\hat{b} = \frac{b}{\|b\|_2}$. Note that $QQ^{\mathrm{T}} = Q^{\mathrm{T}}Q = I$, the identity.

By using the transformation $y = Qz$, the SDP problem is modified in the following manner:

$$\begin{aligned} &\inf_{z \in \mathbb{R}^m} \|b\|_2 z_1 \\ &\text{s. t. } C - \mathcal{A}^{\mathrm{T}}(Qz) \preceq 0 \\ &\qquad PQz \geq 0 \end{aligned} \tag{2.4}$$

Note that $\mathcal{A}^{\mathrm{T}}(Qz)$ can be simplified:

Lemma 2.1.1. *A transposed linear matrix operator under rotation of its argument is still a transposed linear matrix operator. That is:*

$$\mathcal{A}^{\mathrm{T}}(Qz) = \mathcal{A}_Q^{\mathrm{T}}(z) = \sum_j z_j \, \mathcal{A}^{\mathrm{T}}(q_j) \tag{2.5}$$

In which $q_1, \ldots, q_m \in \mathbb{R}^m$ denote the column vectors of Q.

Proof.

$$\mathcal{A}^{\mathrm{T}}(Qz) = \sum_i (Qz)_i A_i = \sum_i (\sum_j Q_{i,j} z_j) A_i \tag{2.6}$$

(In here $Q_{i,j}$ means the element in Q on the ith row, jth column.)

By swapping the sums:

$$\sum_i (\sum_j Q_{i,j} z_j) A_i = \sum_j z_j (\sum_i Q_{i,j} A_i) \tag{2.7}$$

This formulation is recognizable as a linear operator in z much alike as $\mathcal{A}^{\mathrm{T}}$, which is a motivation to denote this formula as the operator $\mathcal{A}_Q^{\mathrm{T}}$ acting on z. Additionally, for clean notation, express $\sum_i Q_{i,j} A_i$ as $\mathcal{A}^{\mathrm{T}}(q_j)$. □

Second, next comes the crucial part in which the SDP problem is recast into an eigenvalue optimization problem.

Lemma 2.1.2. *Consider the SDP problem 2.4. If $\mathcal{A}^{\mathrm{T}}(b)$ is positive definite, then the SDP constraint can be altered into an eigenvalue constraint on z_1:*

$$z_1 \geq \lambda_{\max}(L^{-1}CL^{-\mathrm{T}} - L^{-1}(\sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j))L^{-\mathrm{T}}) \tag{2.8}$$

In which $LL^{\mathrm{T}} = \mathcal{A}^{\mathrm{T}}(b)$ is the Cholesky decomposition

Proof. Note that because $\mathcal{A}^{\mathrm{T}}(b)$ is positive definite, L, L^{T}, L^{-1} and $L^{-\mathrm{T}}$ are all non-singular, hence:

$$\begin{aligned} C - \mathcal{A}^{\mathrm{T}}(Qz) &= C - z_1 \mathcal{A}^{\mathrm{T}}(b) - \sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j) && (2.9)\\ &= C - z_1 LL^{\mathrm{T}} - \sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j) && (2.10)\\ &= L(L^{-1}CL^{-\mathrm{T}} - z_1 I - L^{-1}(\sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j))L^{-\mathrm{T}})L^{\mathrm{T}} && (2.11) \end{aligned}$$

Thus the constraint $C - \mathcal{A}^{\mathrm{T}}(Qz) \preceq 0$ can be written as:

$$L^{-1}CL^{-\mathrm{T}} - L^{-1}(\sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j))L^{-\mathrm{T}} - z_1 I \preceq 0 \tag{2.12}$$

(See theorem 1.2.9 and note that neither L^{-1} nor $L^{-\mathrm{T}}$ are singular).
Because both L and L^{T} are positive definite.
Which is equivalent to stating that $z_1 \geq \lambda_{\max}(L^{-1}CL^{-\mathrm{T}} - L^{-1}(\sum_{j\geq 2} z_j \mathcal{A}^{\mathrm{T}}(q_j)L^{-\mathrm{T}}))$.
□

Third, and finally, check if the inequality constraint in the SDP problem is indifferent to z_1:

Lemma 2.1.3. *Consider the SDP problem 2.4. If $Pb = 0$, then $PQz \geq 0$ is independent to z_1.*

Proof.

$$PQz = z_1 Pb + PQ\,(0,\ z_2, \ldots, z_m)^{\mathrm{T}} = PQ\,(0,\ z_2, \ldots, z_m)^{\mathrm{T}} \tag{2.13}$$

□

And finally, combining all of this into one theorem:

Theorem 2.1.4. *Consider the SDP problem 2.1. If $\mathcal{A}^{\mathrm{T}}(b) \succ 0$ and $Pb = 0$, then the SDP problem is equivalent to the following eigenvalue optimization problem:*

$$\begin{aligned} &\inf_{z\in\mathbb{R}^{m-1}} \|b\|_2 \lambda_{\max}(L^{-1}CL^{-\top} - \sum_j z_j L^{-1} \mathcal{A}^{\mathrm{T}}(q'_j)L^{-\mathrm{T}}) \\ &\quad \text{s. t. } PQ'z \geq 0 \end{aligned} \tag{2.14}$$

In which $q'_1, \ldots, q'_{m-1}, b$ form a full rank orthogonal basis, $Q' = (\, q_1 \mid \ldots \mid q_{m-1} \,)$, and $LL^{\mathrm{T}} = \mathcal{A}^{\mathrm{T}}(b)$ is a Cholesky decomposition.

An implementation based on theorem 2.1.4 is included in appendix A.1.

2.1.1 Perturbing semidefinite optimization problems

This subsection uses results from section 2.2. If unfamiliar with analysis on eigenvalues then it is advised to read that section first.

When trying to use theorem 2.1.4 and $\mathcal{A}^{\mathrm{T}}(b) \succeq 0$ but $\mathcal{A}^{\mathrm{T}}(b)$ is not of full rank, then the trick with the Cholesky decomposition in lemma 2.1.2 is no longer valid. Therefore it is tempting to add a small perturbation such that $\widetilde{\mathcal{A}}^{\mathrm{T}}(\widetilde{b})$ is positive definite. In this section one such possible perturbation is explored.

As illustration of the perturbation being examined: Assuming that $\mathcal{A}^{\mathrm{T}}(b)$ is positive semidefinite with rank $n-1$. Let v be such that $\mathcal{A}^{\mathrm{T}}(b)v = 0$, then adding ϵvv^{T} with $\epsilon > 0$ to $\mathcal{A}^{\mathrm{T}}(b)$ is enough to cast it positive definite. The proposed perturbation is extending b with one component ϵ and $\mathcal{A}^{\mathrm{T}}$ with the matrix vv^{T}. This idea, and the extension to a larger dimensional null space, is examined in the next two lemmas and theorems.

Lemma 2.1.5. *For $V \in \mathcal{S}^n$ with $\|V\|_F = \langle V, V \rangle = 1$ and $\varepsilon > 0$: Let $\widetilde{X} = X + \varepsilon V$ and $\langle V, \widetilde{X} \rangle \geq \epsilon$. If $X \succeq 0$ then and only then $\widetilde{X} \succeq 0$.*

Proof. Note that if $X \succeq 0$, then $\widetilde{X}$ is the sum of two positive semidefinite matrices, hence also positive semidefinite.

For the other direction, $\widetilde{X} \succeq 0$ is equivalent to that for all $Y \in \mathcal{S}^n_{\succeq 0}$ the relation $\langle \widetilde{X}, Y \rangle \geq 0$ holds. Let $Y_1 = \langle Y, V \rangle V$ and $Y_2 = Y - \langle Y, V \rangle V$ for any Y. Note that both Y_1 and Y_2 are positive semidefinite. Then $\langle \widetilde{X}, Y \rangle \geq 0$ and:

$$\begin{aligned} \langle X, Y \rangle &= \langle \widetilde{X} - \varepsilon V, Y \rangle \\ &= \langle \widetilde{X}, Y \rangle + \langle -\varepsilon V, Y \rangle \\ &= \langle \widetilde{X}, Y_1 + Y_2 \rangle - \varepsilon(\langle V, Y \rangle) \\ &= \langle \widetilde{X}, V \rangle \langle Y, V \rangle + \langle \widetilde{X}, Y_2 \rangle - \varepsilon \langle V, Y \rangle \\ &\geq \varepsilon \langle Y, V \rangle + \langle \widetilde{X}, Y_2 \rangle - \varepsilon \langle V, Y \rangle \\ &\geq \langle \widetilde{X}, Y_2 \rangle \geq 0 \end{aligned} \tag{2.15}$$

Hence for all $Y \in \mathcal{S}^n_{\succeq 0}$, $\langle X, Y \rangle \geq 0$ thus $X \succeq 0$. □

Lemma 2.1.6. *Consider problem 2.1 with $V \in \mathcal{S}^n_{\succeq 0}$. Then the following per-*

turbed primal problem has the same optimal as the unperturbed problem:

$$\begin{aligned} \sup_{\widetilde{X}\in\mathcal{S}^n} \; & \langle C, \widetilde{X} - \varepsilon V\rangle \\ \text{s.t. } & \mathcal{A}(\widetilde{X} - \varepsilon V) = b \\ & \langle V, \widetilde{X}\rangle \geq \varepsilon \\ & \widetilde{X} \succeq 0 \end{aligned} \tag{2.16}$$

Proof. Note that if the original primal problem is unbounded, then the perturbed problem is also unbounded and the statement holds trivial. Hence, assume that the original primal problem is bounded.

To continue, observe that the perturbed primal problem is equivalent to transforming $\widetilde{X} = X + \varepsilon V$ for optimal $\widetilde{X}$ and X; because with this transformation the objective of the primal problem is equivalent to the objective of the perturbed primal problem, as well as the first equality constraint.

Hence the only difference is in the semidefinite constraint, but because $\langle V, \widetilde{X}\rangle \geq \varepsilon$ it follows that if $\widetilde{X} \succeq 0$ then and only then $X \succeq 0$, as proven in lemma 2.1.5. Therefore the constraint is in the original problem equivalent to the perturbed constraint. □

The next theorem examines the case where the rank of $\mathcal{A}^{\mathrm{T}}(b)$ is $n-1$. That is, if the dimension of the null space is 1. This case is intuitive easier to follow because the derivative of the corresponding eigenvalue is well defined. The multidimensional case is written down in theorem 2.1.8.

Theorem 2.1.7. *Consider the perturbed primal 2.16, with $\mathcal{A}^{\mathrm{T}}(b)$ of rank $n-1$ and $v \in \mathbb{R}^n$ normalized such that $\mathcal{A}^{\mathrm{T}}(b)v = 0$ and $V = vv^{\mathrm{T}}$. Denote $\widetilde{\mathcal{A}}$ and $\widetilde{b}$ as the corresponding perturbed values appearing in the dual:*

$$\begin{aligned} \inf_{y\in\mathbb{R}^m} \; & \widetilde{b}^{\mathrm{T}} y \\ \text{s.t. } & C - \widetilde{\mathcal{A}}^{\mathrm{T}}(y) \preceq 0 \\ & Py \geq 0 \end{aligned} \tag{2.17}$$

Then $\widetilde{A}^{\mathrm{T}}(\widetilde{b}) = \mathcal{A}^{\mathrm{T}}(b) + \varepsilon(\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}) + vv^{\mathrm{T}})$, and $\varepsilon > 0$ can be chosen such that $\widetilde{\mathcal{A}}^{\mathrm{T}}(\widetilde{b})$ is of full rank.

Furthermore, if $\epsilon < \delta$ for a sufficient small δ, then $\widetilde{A}^{\mathrm{T}}(\widetilde{b}) \succ 0$ (is positive definite).

Proof. Following duality theory:

$$\widetilde{b} = \left[\, (b + \varepsilon\, \mathcal{A}(vv^{\mathrm{T}}))^{\mathrm{T}} \,|\, \varepsilon \,\right]^{\mathrm{T}} \tag{2.18}$$

$$\widetilde{\mathcal{A}}^{\mathrm{T}}((y_1, \ldots, y_m, y_{m+1})^{\mathrm{T}}) = \mathcal{A}^{\mathrm{T}}((y_1, \ldots, y_m)^{\mathrm{T}}) + y_{m+1} vv^{\mathrm{T}} \tag{2.19}$$

$$P = e_{m+1} \tag{2.20}$$

Following those relations:

$$\begin{aligned}\widetilde{\mathcal{A}}^{\mathrm{T}}(\widetilde{b}) &= \mathcal{A}^{\mathrm{T}}(b + \varepsilon\,\mathcal{A}(vv^{\mathrm{T}})) + \varepsilon vv^{\mathrm{T}} \\ &= \mathcal{A}^{\mathrm{T}}(b) + \varepsilon\,\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}) + \varepsilon vv^{\mathrm{T}} \\ &= \mathcal{A}^{\mathrm{T}}(b) + \varepsilon(\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}) + vv^{\mathrm{T}})\end{aligned} \tag{2.21}$$

Next use the continuity theorem 2.2.4 and differentiability theorem 2.2.7 to show that for small $z > 0$, $f(z)$ is of full rank:

Define:

$$\begin{aligned}f : \mathbb{R} \to \mathcal{S}^n,\ z \mapsto \mathcal{A}^{\mathrm{T}}(b) + zE \\ E = \mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}) + vv^{\mathrm{T}}\end{aligned} \tag{2.22}$$

Because $\lambda_{n-m} > 0$ for all $1 \leq m \leq n-1$, it follows by the continuity theorem 2.2.4 that $\lambda_{n-m}(f(z)) > 0$ for small $z > 0$.

Further, observe that $\frac{\mathbf{d}\,\lambda_n f(z)}{\mathbf{d}\,z} > 1$, using theorem 2.2.7:

$$\begin{aligned}\frac{\mathbf{d}\,\lambda_n f(z)}{\mathbf{d}\,z} &= v^{\mathrm{T}} E v \\ &= v^{\mathrm{T}}(\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}) + vv^{\mathrm{T}})v \\ &= v^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}})v + vv^{\mathrm{T}}v \\ &= \langle \mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(vv^{\mathrm{T}}), vv^{\mathrm{T}} \rangle + 1 \\ &= \langle \mathcal{A}(vv^{\mathrm{T}}), \mathcal{A}(vv^{\mathrm{T}}) \rangle + 1 \\ &\geq 1\end{aligned} \tag{2.23}$$

Hence also $\lambda_n(f(z)) > 0$ for $z > 0$ small. Thus there exists a $\delta > 0$ such that for all $0 < \varepsilon < \delta$, $f(\varepsilon)$ is of full rank. □

This is the multidimensional case:

Theorem 2.1.8. *Consider the perturbed primal 2.16, with $\mathcal{A}^{\mathrm{T}}(b)$ of rank $n-k$. That is $\lambda_{n-k+1} = \cdots = \lambda_{n-1} = 0$. Let $V \in \mathcal{S}^n_{\succeq 0}$ such that $\|V\|_F = 1$. Denote $\widetilde{\mathcal{A}}$ and $\widetilde{b}$ as the corresponding perturbed values appearing in the dual:*

$$\begin{aligned}&\inf_{y \in \mathbb{R}^m} \widetilde{b}^{\mathrm{T}} y \\ &\text{s. t. } C - \widetilde{\mathcal{A}}^{\mathrm{T}}(y) \preceq 0 \\ &\qquad Py \geq 0\end{aligned} \tag{2.24}$$

Then $\widetilde{A}^{\mathrm{T}}(\widetilde{b}) = \mathcal{A}^{\mathrm{T}}(b) + \varepsilon(\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(V) + V)$. Moreover if for each $v \in \mathbb{R}^n$ with $\mathcal{A}^{\mathrm{T}}(b)v = 0$, $v^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(V)v + v^{\mathrm{T}}Vv > 0$, then for any $\varepsilon < \delta$ for a sufficient small δ implies $\widetilde{A}^{\mathrm{T}}(\widetilde{b}) \succ 0$ (is positive definite).

Proof. For the first part this proof is identical to the theorem 2.1.7:

$$\widetilde{b} = \left[\left(b + \varepsilon \mathcal{A}(V)\right)^{\mathrm{T}} \mid \varepsilon \right]^{\mathrm{T}} \tag{2.25}$$

$$\widetilde{\mathcal{A}}^{\mathrm{T}}((y_1, \ldots, y_m, y_{m+1})^{\mathrm{T}}) = \mathcal{A}^{\mathrm{T}}((y_1, \ldots, y_m)^{\mathrm{T}}) + y_{m+1}V \tag{2.26}$$

$$P = e_{m+1} \tag{2.27}$$

And:

$$\widetilde{\mathcal{A}}^{\mathrm{T}}(\widetilde{b}) = \mathcal{A}^{\mathrm{T}}(b) + \varepsilon(\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(V) + V) \tag{2.28}$$

Also define:

$$\begin{aligned} f &: \mathbb{R} \to \mathcal{S}^n,\ z \mapsto \mathcal{A}^{\mathrm{T}}(b) + zE \\ E &= \mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(V) + V \end{aligned} \tag{2.29}$$

Again continuity follows from theorem 2.2.4 which implies that for small value of $z > 0$ the dimension of the null space of $f(z)$ cannot increase.

Since this time the eigenvalue corresponding to the null space is multiple it is invalid to use theorem 2.2.7. Instead apply theorem 2.2.15: Note that the supgradient of λ_n consists of the convex hull of vector outproducts that span the null space, that is:

$$\begin{aligned} \partial\lambda_{n-k+1}(f(0)) &= \{Z \in \mathcal{S}^n \text{ s.t. } \langle Z, \mathcal{A}^{\mathrm{T}}(b)\rangle = 0,\ Z \succeq 0\} \\ &= \mathrm{Co}\{qq^{\mathrm{T}} \text{ s.t. } \mathcal{A}^{\mathrm{T}}(b)q = 0\} \end{aligned} \tag{2.30}$$

(This follows from corollary 2.2.23.)

The condition that for each $v \in \mathbb{R}^n$ with $\mathcal{A}^{\mathrm{T}}(b)v = 0$, $v^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}} \circ \mathcal{A}(V)v + v^{\mathrm{T}}Vv$ implies that for any $Z \in \partial\lambda_{n-k+1}(f(0))$, $\langle E, Z\rangle > 0$, hence E is an ascent direction for λ_n. Because λ_n is the minimum of all eigenvalues it follows that all eigenvalues λ_{n-k+1} up to λ_n ascent in the direction of E (theorem 2.2.4.

Hence also $\lambda_n(f(z)) > 0$ for $z > 0$ small. Thus there exists a $\delta > 0$ such that for all $0 < \varepsilon < \delta$, $f(\varepsilon)$ is of full rank. □

What's the catch of this perturbation? For a start, in the perturbed case $Pb = \varepsilon$. Hence to solve it by using the eigenvalue optimization equivalence, the problem has to be relaxed a little and perturbed. But because it is expected that $\varepsilon\lambda_{\max}$ is small positive, this will not affect the optimal solution in a significant way, albeit it is a source of error.

However, another catch of the perturbation is that the system becomes more sensitive to numerical errors. The condition number of $\widetilde{\mathcal{A}}^{\mathrm{T}}(b)$ will be large because of the small eigenvalue, hence the Cholesky decomposition step is prone to large numerical errors.

Thus for some instances the perturbation can be made large enough for $\widetilde{\mathcal{A}}^{\mathrm{T}}(b)$ to be stable, and for some instances the numerical errors will be too large in magnitude to work on the system.

2.1.2 Some examples

As examples of deriving the eigenvalue optimization problem for some SDP the examples from section 1.4 will be cast into an eigenvalue problem.

Max-cut

As explained in section 1.4.1, the dual of the max-cut is given by:

$$\begin{aligned} &\min_{y \in \mathbb{R}^{|V|}} \sum_i y_i \\ &\text{s.t. } \frac{1}{4} L_G - \sum_i y_i E_{i,i} \preceq 0 \end{aligned} \tag{2.31}$$

(This is the same as equation 1.56.)
To start:

$$\mathcal{A}^{\mathrm{T}}(\hat{b}) = \sum_i E_{i,i} = (|V|)^{-\frac{1}{2}} I \tag{2.32}$$

Which has a trivial Cholesky decomposition: $\mathcal{A}^{\mathrm{T}}(\hat{b}) = LL^{\mathrm{T}}$ with $L = (|V|)^{-\frac{1}{4}} I$. Furthermore, let $m = |V|$, $q_1, \ldots, q_m \in \mathbb{R}^m$ with $q_m = \hat{b} \in \mathbb{R}^m$ be an orthogonal basis. Then $\mathcal{A}^{\mathrm{T}}(q_1), \ldots, \mathcal{A}^{\mathrm{T}}(q_m) \in \mathcal{S}^n$ are all diagonal matrices with on their diagonals $q_1, \ldots, q_m$. That all makes SDP 2.31 equivalent to:

$$\begin{aligned} &\inf_{z \in \mathbb{R}^{|V|-1}} (|V|)^{\frac{1}{2}} \lambda_{\max}(\frac{1}{4} L^{-1} L_G L^{-\mathrm{T}} - \sum_i z_i L^{-1} \operatorname{diag}(q_i) L^{-\mathrm{T}}) \\ &\inf_{z \in \mathbb{R}^{|V|-1}} (|V|)^{\frac{1}{2}} \lambda_{\max}((|V|)^{\frac{1}{2}} \frac{1}{4} L_G - \sum_i z_i (|V|)^{\frac{1}{2}} \operatorname{diag}(q_i)) \\ &\inf_{z \in \mathbb{R}^{|V|-1}} |V| \lambda_{\max}(\frac{1}{4} L_G - \sum_i z_i \operatorname{diag}(q_i)) \end{aligned} \tag{2.33}$$

It follows from the fact that $\operatorname{diag}(q_1), \ldots, \operatorname{diag}(q_{|V|-1}) \perp \operatorname{diag}(q_m) = (|V|)^{-\frac{1}{2}} I$ that there is no sum possible such that $\sum_i \gamma_i \operatorname{diag}(q_i) \succ 0$. (If there would be, then $q_m^{\mathrm{T}}(\sum_i \gamma_i q_i) > 0$, which is a contradiction because all $q_i \perp q_m$.) Therefore the eigenvalue problem is bounded.

The difficulty of this problem lies in generating a basis $q_1, \ldots, q_m$, which can be computational hard for large m. If memory requirements are such that $q_1, \ldots, q_m$ do not have to be optimized for as few non-zeros as possible, then a fast way to compute a basis is by using Lanczos iterations.

Note that if the SDP was sparse, then the eigenvalue problem is also sparse. That will be an important property when solving the eigenvalue problem in chapter 4.

Ideal GMRES polynome

The ideal GMRES polynome of order k for matrix A can be computed using:

$$\begin{aligned} &\min_{t\in\mathbb{R},\, y\in\mathbb{R}^k} t \\ &\text{s.t. } \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} + \sum_{1\leq i\leq k} y_i \begin{pmatrix} 0 & A^i \\ A^i & 0 \end{pmatrix} - t \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \preceq 0 \end{aligned} \tag{2.34}$$

As was explained in section 1.4.3.

Denote $y' = (y^{\mathrm{T}}, t)^{\mathrm{T}}$, then $b = e_{k+1}$, and $\mathcal{A}^{\mathrm{T}}(b) = I$. Hence an orthogonal basis $q_1, \ldots, q_{k+1} \in \mathbb{R}^{k+1}$ is $e_1, \ldots e_{k+1} \in \mathbb{R}^{k+1}$. That makes the eigenvalue problem transformation trivial easy:

$$\inf_{z\in\mathbb{R}^k} \lambda_{\max}(\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} + \sum_{1\leq i\leq k} z_i \begin{pmatrix} 0 & A^i \\ A^i & 0 \end{pmatrix}) \tag{2.35}$$

Phase retrieval

The dual of 1.76 is:

$$\begin{aligned} &- \min_{y\in\mathbb{R}^{|\Omega|}} b^{\mathrm{T}} y \\ &\text{s.t. } -I - \mathcal{A}^{\mathrm{T}}(y) \preceq 0 \end{aligned} \tag{2.36}$$

Or similar for the $\log\det$ heuristic:

$$\begin{aligned} &- \min_{y\in\mathbb{R}^{|\Omega|}} b^{\mathrm{T}} y \\ &\text{s.t. } -W_k - \mathcal{A}^{\mathrm{T}}(y) \preceq 0 \end{aligned} \tag{2.37}$$

Because $A_\omega = a_\omega a_\omega^*$ and $b_\omega > 0$, it is for certain than $\mathcal{A}^{\mathrm{T}}(b) \succeq 0$. If in addition $\mathcal{A}^{\mathrm{T}}(b) \succ 0$, then theorem 2.1.4 can be applied. Let LL^{T} be a Cholesky decomposition of $\mathcal{A}^{\mathrm{T}}(b)$, then problem 2.37 is equivalent to:

$$- \inf_{z\in\mathbb{R}^{|\Omega|-1}} \lambda_{\max}(L^{-1} W_k L^{-\mathrm{T}} - \sum_j z_j L^{-1}\, \mathcal{A}^{\mathrm{T}}(q_j) L^{-\mathrm{T}}) \tag{2.38}$$

As mentioned at the end of 1.4.4, if the optimal y_k^* for iteration k is known, then a good dual starting point for the next iteration is $z = Q^{\mathrm{T}} y_k^*$. Here are some arguments:

Let $D_k = W_{k+1} - W_k$, such that:

$$- \inf_{z\in\mathbb{R}^{|\Omega|-1}} \lambda_{\max}(L^{-1}(I + \sum_i^k D_i) L^{-\mathrm{T}} - \sum_j z_j L^{-1}\, \mathcal{A}^{\mathrm{T}}(q_j) L^{-\mathrm{T}}) \tag{2.39}$$

As the log det heuristic converges it is expected that $D_k \to 0$ as $k \to \infty$. Hence the dual programs will converge. Because the term $L^{-1}\,\mathcal{A}^{\mathrm{T}}(q_j)L^{-\mathrm{T}}$ is constant between the programs, and the term $L^{-1}(I + \sum_i^k D_i)L^{-\mathrm{T}}$ converges, it is expected that the optimal solution z_k^* also converges.

2.2 Analysis on eigenvalues

In order to keep notation clean, in the remainder of this chapter the subscripts and similarity transformations in equation 2.14 are omitted.

Definition 2.2.1 (Eigenvalue optimization)**.** *An eigenvalue optimization problem is of the following form:*

$$\begin{aligned} &\inf_{z\in\mathbb{R}^m} \lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z)) \\ &\text{s. t. } Pz \geq 0 \end{aligned} \tag{2.40}$$

With $C \in \mathcal{S}^n$, $\mathcal{A}^{\mathrm{T}} : \mathbb{R}^m \to \mathcal{S}^n$, $z \mapsto \sum_i z_i A_i$ linear operator and $P \in \mathbb{R}^{k\times m}$. Also $\geq$ is per component defined.

In this section some properties on eigenvalues as matrix perturbation will be derived which will be used to derive important properties of eigenvalue optimization problems. It will be shown that the problem is convex, although not everywhere differentiable.

Since $C \in \mathcal{S}^n$ as well as $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$, everything is about symmetric real matrices. Hence the further discussion in this section will be restricted to symmetric real matrices.

First a remark on notation, in this section some properties of eigenvalues as perturbation are examined, i.e. functions of the form $E \mapsto A + E$ with $A, E \in \mathbb{R}^{n\times n}$. In most cases are A and E restricted to $\mathcal{S}^n$.

A has n eigenvalues $\lambda_i(A)$ which are labeled as:

$$\lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_n(A) \tag{2.41}$$

If an eigenvalue has multiplicity $t > 1$, then it is counted t times, e.g. if λ_1 has multiplicity 2, then $\lambda_1 = \lambda_2$.

Moreover, if from the context it is clear for a given eigenvalue which matrix is meant, the argument might be omitted.

In the next subsection some theorems concerning continuity and differentiability of eigenvalues are discussed. For the first case it does not matter whether an eigenvalue is simple or non-simple, for the second case such a distinction is necessary. In the second subsection differentiability of non-simple eigenvalues is discussed.

2.2.1 Continuity and differentiability for the simple case

The following characterization due to Courant, Fischer and Weyl of λ_i forms the cornerstone of all the other derivations:

Theorem 2.2.2 (Courant–Fischer–Weyl min–max principle)**.** *Let A be a Hermitian matrix with eigenvalues labeled as described. Then*

$$\lambda_i = \max_{\substack{\mathcal{X}\subseteq\mathcal{L}^n \\ \dim(\mathcal{X})=i}} \min_{\substack{x\in\mathcal{X} \\ \|x\|_2=1}} x^{\mathrm{T}}Ax \tag{2.42}$$

And:

$$\lambda_i = \min_{\substack{\mathcal{X}\subseteq\mathcal{L}^n \\ \dim(\mathcal{X})=n-i+1}} \max_{\substack{x\in\mathcal{X} \\ \|x\|_2=1}} x^{\mathrm{T}}Ax \tag{2.43}$$

Proof. The first equality: Because A is Hermite it is diagonizable with eigenvectors u_k corresponding to λ_k. Since $\dim(\mathcal{X}) = i$, there is an intersection between $\mathcal{X}$ and $\operatorname{span}\{u_i, \dots u_n\}$. Therefore:

$$\lambda_i \leq \max_{\substack{\mathcal{X}\subseteq\mathcal{L}^n \\ \dim(\mathcal{X})=i}} \min_{\substack{x\in\mathcal{X} \\ \|x\|_2=1}} x^{\mathrm{T}}Ax \tag{2.44}$$

By choosing $\mathcal{X} = \operatorname{span}\{u_1, \dots, u_i\}$ the $\lambda_i \geq \max\min x^{\mathrm{T}}Ax$ inequality is observed. Hence $\lambda_i = \max\min x^{\mathrm{T}}Ax$.

The second equality follows by replacing A by $-A$. □

Corollary 2.2.3 (Rayleigh quotient)**.**

$$\lambda_1 = \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}}Ax = \sup_{q\in\mathbb{R}^n,\ \|q\|_2=1} \langle A, qq^{\mathrm{T}}\rangle = \sup_{\substack{Z\in\mathcal{S}^n_{\succeq 0} \\ \langle Z,I\rangle=1}} \langle A, Z\rangle \tag{2.45}$$

And:

$$\lambda_n = \min_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}}Ax = \inf_{q\in\mathbb{R}^n,\ \|q\|_2=1} \langle A, qq^{\mathrm{T}}\rangle = \inf_{\substack{Z\in\mathcal{S}^n_{\succeq 0} \\ \langle Z,I\rangle=1}} \langle A, Z\rangle \tag{2.46}$$

Proof. Proof for the first equation is shown, the second equation follows in a similar manner.

The first equality is a direct consequence of theorem 2.2.2. The second equality follows: $Z \in \mathcal{S}^n_{\succeq 0} : \langle Z, I\rangle = 1$ implies that Z has an eigendecomposition $Z = \sum_i \alpha_i q_i q_i^{\mathrm{T}}$ with $0 < \alpha_i \leq 1$, $\sum_i \alpha_i = 1$ and $q_i^{\mathrm{T}} q_j = \delta_{ij}$ (that is 0 if $i \neq j$ and 1 otherwise). Then:

$$\begin{aligned} \langle A, Z\rangle &= \operatorname{Tr}(A(\sum_i \alpha_i q_i q_i^{\mathrm{T}})) \\ &= \sum_i \alpha_i \operatorname{Tr}(A q_i q_i^{\mathrm{T}}) \\ &= \sum_i \alpha_i \operatorname{Tr}(q_i^{\mathrm{T}} A q_i) \end{aligned} \tag{2.47}$$

Which is maximized if all q_i are eigenvectors corresponding to $\lambda_{\max}$. Hence the supremum of $\langle A, Z\rangle = \sum_i \alpha_i \lambda_{\max} = \lambda_{\max}$. □

It is easy by using the min-max principle to show continuity of the eigenvalues of $A(E) = A + E$ and putting an upper bound on the variation of $\lambda_i(A(E))$ as function of E:

Corollary 2.2.4. *The perturbation of $\lambda_i(A+E)$ is bounded by:*

$$\lambda_{\min}(E) \leq \lambda_i(A+E) - \lambda_i(A) \leq \lambda_{\max}(E) \tag{2.48}$$

Which is equivalent to:

$$\lambda_i(A+E) \in [\lambda_i(A) + \lambda_{\min}(E), \lambda_i(A) + \lambda_{\max}(E)] \tag{2.49}$$

Which implies $f(E) = \lambda_i(A+E)$ is continuous in E.

Proof. First $\lambda_i(A+E) - \lambda_i(A) \leq \lambda_{\max}(E)$:

$$\begin{aligned}
\lambda_i(A+E) - \lambda_i(A) &= \max_{\dim(\mathcal{X})=i} \min_{\|x\|_2=1} x^\top (A+E)x - \max_{\dim(\mathcal{X})=i} \min_{\|x\|_2=1} x^{\mathrm{T}} A x \\
&\leq \max_{\dim(\mathcal{X})=i} \min_{\|x\|_2=1} x^{\mathrm{T}} A x + \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}} E x - \max_{\dim(\mathcal{X})=i} \min_{\|x\|_2=1} x^{\mathrm{T}} A x \\
&= \max_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}} E x \\
&= \lambda_{\max}(E)
\end{aligned} \tag{2.50}$$

Second $\lambda_i(A+E) - \lambda_i \geq \lambda_{\min}(E)$:

$$\begin{aligned}
\lambda_i(A+E) - \lambda_i(E) &= \min_{\dim(\mathcal{X})=n-i+1} \max_{\|x\|_2=1} x^{\mathrm{T}} (A+E)x - \min_{\dim(\mathcal{X})=n-i+1} \max_{\|x\|_2=1} x^{\mathrm{T}} A x \\
&\geq \min_{\dim(\mathcal{X})=n-i+1} \max_{\|x\|_2=1} x^{\mathrm{T}} A x + \min_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}} E x - \min_{\dim(\mathcal{X})=n-i+1} \min_{\|x\|_2=1} x^{\mathrm{T}} A x \\
&= \min_{\substack{x\in\mathbb{R}^n \\ \|x\|_2=1}} x^{\mathrm{T}} E x \\
&= \lambda_{\min}(E)
\end{aligned} \tag{2.51}$$

□

This gives direct a way to identify whether problem 2.40 is unbounded:

Corollary 2.2.5. *Eigenvalue optimization problem 2.40 is unbounded if and only if there exists $z \in \mathbb{R}^m$ such that $Pz = 0$ and $\mathcal{A}^{\mathrm{T}}(z) \succeq 0$ (positive definite).*

Note that the power of corollary 2.2.4 is that it does not require that λ_i is simple. In most cases it is not so easy to say something when λ_i is non-simple.

When $\lambda_i(A)$ is simple, then the derivative exists in closed form (based on [18]):

Lemma 2.2.6. *Let $\lambda_i(A) \in \mathbb{R}$ simple with corresponding eigenvector $u_i(A) \in \mathbb{R}^n$. Then $\lambda_i(A+E)$ and $u_i(A)$ are ∞ times differentiable at A.*

Proof. Define the function:

$$f : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n+1}, (u, \lambda; A) \mapsto \begin{pmatrix} (A - \lambda I)u \\ u^{\mathrm{T}}u - 1 \end{pmatrix} \tag{2.52}$$

Then $f(u_i, \lambda_i) = 0$. Note that f is ∞ times differentiable. The Jacobian of f to u and λ is given by:

$$D_{u,\lambda} f(u, \lambda; A) = \begin{pmatrix} A - \lambda I & u \\ u' & 0 \end{pmatrix} \tag{2.53}$$

Which is non-singular by lemma 1.2.11 (Schur complement). Hence by the implicit function theorem there exists an open interval in $N(A) \subset \mathbb{R}^{n\times n}$ containing A. Moreover, there exists functions $u_i : \mathbb{R}^{n\times n} \to \mathbb{R}^n$ and $\lambda_i : \mathbb{R}^{n\times n} \to \mathbb{R}$ which are continuous and ∞ times differentiable on $N(A)$. □

Theorem 2.2.7. *Let $\lambda_i(A)$ simple with corresponding normalized eigenvector $u_i(A)$. Then the derivative of $\lambda_i(A+E)$ to E is:*

$$\frac{\mathbf{d}\,\lambda_i(A+E)}{\mathbf{d}\,E} = u_i^{\mathrm{T}}(A) E u_i(A) \tag{2.54}$$

Proof. By using the identity $u_i^{\mathrm{T}}(A)(A - \lambda_i(A)I)u_i(A) = 0$ and the product rule:

$$\begin{aligned}
&\frac{\mathbf{d}}{\mathbf{d}\,E}\left(u_i^{\mathrm{T}}(A+E)(A+E-\lambda_i(A+E)I)u_i(A+E)\right) \\
&\quad = \frac{\mathbf{d}\,u_i^{\mathrm{T}}(A+E)}{\mathbf{d}\,E}(A-\lambda_i(A))u_i(A) + u_i^{\mathrm{T}}(A)(E - \frac{\mathbf{d}\,\lambda_i(A+E)}{\mathbf{d}\,E})Eu_i(A) \\
&\qquad + u_i^{\mathrm{T}}(A)(A-\lambda_i(A))\frac{\mathbf{d}\,u_i^{\mathrm{T}}(A+E)}{\mathbf{d}\,E} \\
&\quad = u_i^{\mathrm{T}}(A)(E - \frac{\mathbf{d}\,\lambda_i(A+E)}{\mathbf{d}\,E})u_i(A) = 0
\end{aligned} \tag{2.55}$$

The last equation stems from that $(A-\lambda_i(A))u_i(A) = u_i^{\mathrm{T}}(A)(A-\lambda_i(A)) = 0$. The last line can be simplified:

$$\frac{\mathbf{d}\,\lambda_i(A+E)}{\mathbf{d}\,E} = u_i^{\mathrm{T}}(A) E u_i(A) \tag{2.56}$$

Because $u_i^{\mathrm{T}}(A)u_i(A) = I$. □

Corollary 2.2.8. *If for a given $y \in \mathbb{R}^m$, $\lambda_i(C - \mathcal{A}^{\mathrm{T}}(y))$ is simple with eigenvector u_i, then the gradient of*

$$f : \mathbb{R}^m \to \mathbb{R},\ y \to \lambda_i(C - \mathcal{A}^{\mathrm{T}}(y)) \tag{2.57}$$

is given by:

$$\nabla f(y) = -(u_i^{\mathrm{T}} A_1 u_i, \ldots, u_i^{\mathrm{T}} A_m u_i)^{\mathrm{T}} \tag{2.58}$$

Also the second derivative is computable:

Lemma 2.2.9. *Let $\lambda_i(A)$ simple with corresponding eigenvector $u_i(A)$. Then the first derivative of $u_i(A + E)$ to E is:*

$$\frac{\mathbf{d}\, u_i(A+E)}{\mathbf{d}\, E} = -(A - \lambda_i I)^{\dagger} E u_i \tag{2.59}$$

(For ease of notation: $\lambda_j = \lambda_j(A)$ and $u_j = u_j(A)$ for all $j = 1, \ldots, n$. Furthermore $(A - \lambda_i I)^{\dagger}$ denotes the Moore-Penrose pseudoinverse.)

Proof. Starting with differentiating the following equation:

$$\frac{\mathbf{d}}{\mathbf{d}\, E}((A + E - \lambda_i I)u) = (E - \frac{\mathbf{d}\, \lambda_i}{\mathbf{d}\, E} I)u + (A - \lambda_i I)\frac{\mathbf{d}\, u}{\mathbf{d}\, E} = 0 \tag{2.60}$$

Observe that $(A - \lambda_i I)^{\dagger} u_i = 0$ because the kernel of $A - \lambda_i I$ is exactly spanned by u_i (by assumption that λ_i is simple). Also $(A - \lambda_i I)^{\dagger} \frac{\mathbf{d}\, u_i}{\mathbf{d}\, E} = \frac{\mathbf{d}\, u_i}{\mathbf{d}\, E}$ because $\frac{\mathbf{d}\, u_i}{\mathbf{d}\, E} \perp u_i$ and has therefore no components in the null space of $A - \lambda_i I$. (To see that $\frac{\mathbf{d}\, u_i}{\mathbf{d}\, E} \perp u_i$, note that $u_i^{\mathrm{T}} u_i = 1$ thus $\frac{\mathbf{d}\, u_i^{\mathrm{T}}}{\mathbf{d}\, E} u_i = 0$.)

Hence multiply the equation left with $-(A - \lambda_i I)^{\dagger}$ to obtain equation 2.59. □

Theorem 2.2.10. *Let $\lambda_i(A)$ simple with corresponding eigenvector $u_i(A)$. Then the second derivative of $\lambda_i(A + E)$ to $E \in \mathcal{S}^n$ is:*

$$\begin{aligned} \frac{\mathbf{d}^2\, \lambda_i(A + E_1 + E_2)}{\mathbf{d}\, E_1\, \mathbf{d}\, E_2} &= -2 u_i^{\mathrm{T}} E_1 (A - \lambda_i I)^{\dagger} E_2 u_i \\ &= 2 \sum_{r \neq i} \frac{u_i^{\mathrm{T}} E_1 u_r u_r^{\mathrm{T}} E_2 u_i}{\lambda_i - \lambda_r} \end{aligned} \tag{2.61}$$

Proof. Start with equation 2.54 and differentiate it by using the chain-rule and

the lemma above:

$$\begin{aligned}\frac{\mathbf{d}}{\mathbf{d}\,E_2}\frac{\mathbf{d}\,\lambda_i(A+E_1+E_2)}{\mathbf{d}\,E_1} &= u_i^{\mathrm{T}}E_2u_i \\ &= \frac{\mathbf{d}\,u_i^{\mathrm{T}}}{\mathbf{d}\,E_1}E_2u_1 + u_i^{\mathrm{T}}E_2\frac{\mathbf{d}\,u_i}{\mathbf{d}\,E_1} \\ &= -u_i^{\mathrm{T}}E_1(A-\lambda_iI)^{\dagger}E_2u_i - u_i^{\mathrm{T}}E_2(A-\lambda_iI)^{\dagger}E_1u_i \\ &= -2u_i^{\mathrm{T}}E_1(A-\lambda_iI)^{\dagger}E_2u_i \\ &= 2\sum_{r\neq i}\frac{u_i^{\mathrm{T}}E_1u_ru_r^{\mathrm{T}}E_2u_i}{\lambda_i-\lambda_r}\end{aligned} \tag{2.62}$$

In the last equations it is used that $(A-\lambda_iI)^{\dagger}$, E_1 and E_2 are all symmetric. □

Corollary 2.2.11. *If for a given $y \in \mathbb{R}^m$, $\lambda_i(C-\mathcal{A}^{\mathrm{T}}(y))$ is simple with eigenvector u_i, then the Hessian of*

$$f : \mathbb{R}^m \to \mathbb{R},\ y \to \lambda_i(C-\mathcal{A}^{\mathrm{T}}(y)) \tag{2.63}$$

is given by:

$$(Hf(y))_{k,l} = -2\sum_{j\neq i}\frac{u_i^{\mathrm{T}}A_ku_ju_j^{\mathrm{T}}A_lu_i}{\lambda_i-\lambda_j} \tag{2.64}$$

If λ_i is non-simple, then there are results that all eigenvalues can be labeled (counted to multiplicity) λ_i such that all λ_i curves are smooth [1]. The limitation is that the eigenvectors are not necessarily smooth. Hence, λ_i as function of perturbation E is not always in closed form. This result is stated here for completeness without proof. The reason is that the proof is a deeper result based on smoothness of the roots of a polynome under perturbation. Hence to repeat it here would require another few pages describing functional analysis on polynomes.

Theorem 2.2.12. *Let $A \in \mathcal{S}^n$ and $E \in \mathcal{S}^n$, and let $\mathcal{A}(t) = A+tE$. It is possible to label all eigenvalues of $\mathcal{A}(t)$ as $\lambda_i(t)$ for $i = 1,\ldots n$, (counted to multiplicity), such that each $\lambda_i(t)$ is a smooth function.*

Proof. See [1]. □

2.2.2 The non-simple case

For the non simple case there is no reason to expect that the derivative of $\lambda_i(A+E)$ exists. However the subgradient of the maximum eigenvalue, and the supgradient of the minimal eigenvalue does exist. Those are sets which will be quite useful:

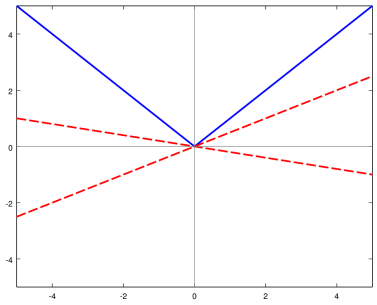

Figure 2.1: $f(x) = |x|$ is plotted with solid blue, two possible subgradients at $x = 0$ are plotted with dashed red

Definition 2.2.13 (Sub- and supgradient)**.** *The subgradient of $f : \mathbb{R}^n \to \mathbb{R}$ at $x_0 \in \mathbb{R}^n$ is defined as the largest closed convex set $\partial^- f(x_0)$ such that for all $v \in \partial^- f(x_0)$ holds:*

$$f(x) - f(x_0) \geq \langle v, x - x_0 \rangle \ (\forall x \in \mathbb{R}^n) \tag{2.65}$$

Moreover, if for all $x \in \mathbb{R}^n$, $\partial^- f(x)$ is non-empty, then it is said that f is subdifferential.

Analogous at $x_0 \in \mathbb{R}^n$ is the supgradient of $g : \mathbb{R}^n \to \mathbb{R}$ the largest closed convex set $\partial^+ g(x_0)$ such that for all $v \in \partial^+ g(x_0)$ holds:

$$g(x) - g(x_0) \leq \langle v, x - x_0 \rangle \ (\forall x \in \mathbb{R}^n) \tag{2.66}$$

Similar if for all $x \in \mathbb{R}^n$, $\partial^+ g(x)$ exists, then g is supdifferential.

If from the context it is clear whether a sub- or supgradient of f is meant, then the set will simply be denoted by ∂f, omitting the $+$ or $-$.

(These definitions are based on the generalized Clark gradient defined in [8] and [7].)

The power of sub/supgradients is that they can define a descent / ascent direction for non differentiable functions. The following two are worded for the subgradient, but is should be clear that there are equivalent statements for concave supdifferentiable functions.

To get a bit more familiar with subgradients, consider the following example:

Example 2.2.14. *Let $f : \mathbb{R} \to \mathbb{R}, x \mapsto |x|$, then for $x < 0$, $\partial f(x) = \{-1\}$, for $x > 0$, $\partial f(x) = \{1\}$ and for $x = 0$, $\partial f(x) = [-1, 1]$. Two possible subgradients are shown in figure 2.1.*

Lemma 2.2.15. *Let $f : R^m \to \mathbb{R}$ be convex subdifferential. If for some $x \in R^m$ there exists $d \in R^m$ such that for all $y \in \partial f(x)$, $\langle y, d \rangle < 0$, then, and only then is d a descent direction at x.*

Proof. The first implication: Assume the contradiction that such a $d \in R^m$ exists but is not a descent direction. Then because f is convex either x is the global minimum for f or $-d$ is not an ascent direction. In the first case, $0 \in \partial f(x)$ and $\langle 0, d\rangle = 0$, hence d does not fit the constraints. In the second case $d \in \partial f(x)$ and $\langle d, d\rangle > 0$, thus also in this case there's a contradiction.

The second implication: Assume d is a descent direction, but there exists $y \in \partial f(x)$ such that $\langle y, d\rangle \geq 0$. Since $y \in \partial f(x)$: $f(x + td) \geq f(x) + \langle y, d\rangle$ for all $t \in \mathbb{R}$, hence d is not a descent direction, which is a contradiction. □

Corollary 2.2.16. *Let $f : R^m \to \mathbb{R}$ be convex subdifferential. If for $x \in R^m$, $0 \in \partial f(x)$, then and only then is x the global minimum.*

Proof. This is the previous lemma but then in other words. □

Furthermore, subgradients are powerful tools to compute directional derivatives when the gradient does not exist (based on [11]):

Theorem 2.2.17. *Assuming that $\partial f(x)$ is bounded in a neighborhood around x: The directional derivative of f can be expressed in terms of the subgradient of f:*

$$\nabla_v f(x) = \lim_{h \downarrow 0} \frac{f(x + hv) - f(x)}{h} = \max_{g \in \partial f(x)} g^{\mathrm{T}} v \tag{2.67}$$

Proof. Define $(h)_k$ as a decreasing sequence in $\mathbb{R}$ such that $h \downarrow 0$, and define $x_k = x + h_k v$. Then:

$$\nabla_v f(x) = \lim_{k \to \infty} \frac{f(x_k) - f(x)}{h_k} \tag{2.68}$$

By the definition of the subgradient:

$$\begin{aligned} f(x_k) - f(x) &\geq h_k v^{\mathrm{T}} g \ \ (\forall g \in \partial f(x)) \\ f(x) - f(x_k) &\geq -h_k v^{\mathrm{T}} g_k \ \ (\forall g_k \in \partial f(x_k)) \end{aligned} \tag{2.69}$$

Therefore:

$$v^{\mathrm{T}} g_k \geq \nabla_v f(x) \geq \max_{g \in \partial f(x)} g^{\mathrm{T}} v \tag{2.70}$$

Because $\partial f(y)$ is closed and bounded for all y near x, and $\partial f(x_k)$ converges to a subset of $\partial f(x)$. It follows that there exists a sequence $g_k \in \partial f(x_k)$ that converges to $\arg\max_{g \in \partial f(x)} g^{\mathrm{T}} v$. □

Before computing the subgradient / supgradient of the maximum / minimum eigenvalue it should be marked that they are convex / concave functions. Indeed if they are not convex / concave, then subgradient / supgradient does not exist.

Lemma 2.2.18. *The function:*

$$\lambda_{\max}(A) : \mathcal{S}^n \to \mathbb{R},\ A \to \lambda_{\max}(A) \tag{2.71}$$

is convex. Similar the function:

$$\lambda_{\min}(t) : \mathbb{R}^m \to \mathbb{R},\ t \to \lambda_{\min}(A_0 + t_1 A_1 + \cdots + t_m A_m) \tag{2.72}$$

is concave.

Proof. For the first statement a proof is given. The second statement follows in a similar manner.

The inequality for convexity follows direct from application of the Rayleigh quotient: For all $t \in [0,1]$ and $A, B \in \mathcal{S}^n$:

$$\begin{aligned} t\lambda_{\max}(A) + (1-t)\lambda_{\max}(B) &= t \max_{\substack{x \in \mathbb{R}^n \\ \|x\|_2 = 1}} x^{\mathrm{T}} A x + (1-t) \max_{\substack{x \in \mathbb{R}^n \\ \|x\|_2 = 1}} x^{\mathrm{T}} B x \\ &\geq \max_{\substack{x \in \mathbb{R}^n \\ \|x\|_2 = 1}} x^{\mathrm{T}} (tA + (1-t)B) x \\ &= \lambda_{\max}(tA + (1-t)B) \end{aligned} \tag{2.73}$$

□

Theorem 2.2.19 (Dubovitskii–Milyutin)**.** *Let f be a convex function defined on $\mathbb{R}^n$ as:*

$$f(x) = \sup_{\alpha \in \mathcal{A}} f_\alpha(x) \tag{2.74}$$

Where $\mathcal{A}$ is some index-set and each f_α is convex and subdifferentiable. Denote the active set $I_x = \{\alpha \in \mathcal{A} \text{ s.t. } f_\alpha(x) = f(x)\}$.

If for all $d \in \mathbb{R}^n$:

$$\nabla_d f(x) = \sup\{\nabla_d f_\alpha(x) \text{ s.t. } \alpha \in I_x\} \tag{2.75}$$

holds, then:

$$\partial f(x) = \mathrm{Co}(\cup\{\partial f_\alpha(x) \text{ s.t. } f(x) = f_\alpha(x)\}) \tag{2.76}$$

That is $\partial f(x)$ is the convex hull of the subgradients of the active functions.

Proof. Observe that $\mathrm{Co}(\cup\{\partial f_\alpha(x) \text{ s.t. } f(x) = f_\alpha(x)\} \subset \partial f(x)$ follows: each $g \in \partial f_{i \in I}(x)$ is in $\partial f(x)$.

The other inclusion: let $g \in \partial f(x)$. If g is not in $\mathrm{Co}(\partial f_{i \in I_x}(x))$ then there exists a strict separating plane $d \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that:

$$g^{\mathrm{T}} d > \alpha > \max_{i \in I_x} \sup_{\epsilon \in \partial f_i(x)} \epsilon^{\mathrm{T}} d \tag{2.77}$$

Furthermore for all $i \in I_x$:

$$\sup_{\epsilon \in \partial f_i(x)} \epsilon^{\mathrm{T}} d = \lim_{t \downarrow 0} \frac{f_i(x+td) - f_i(x)}{t} \tag{2.78}$$

Hence: $\max_{i \in I_x} \lim_{t \downarrow 0} \frac{f_i(x+td)-f_i(x)}{t} < g^{\mathrm{T}} d$. Because J is non-empty and finite around x, the max and lim can be interchanged (define the limit in terms of decreasing sequence $t_k \geq 0$, then there is a sub-sequence such that the statement holds). That gives:

$$\max_{i \in I_x} \sup_{\epsilon \in \partial f_i(x)} = \lim_{t \downarrow 0} \max_{i \in I_x} \frac{f_i(x+td) - f_i(x)}{t} = \lim_{t \downarrow 0} \frac{f(x+td) - f(x)}{t} \tag{2.79}$$

However, note that $g \in \partial f(x)$ implies that $f(x+td) - f(x) \geq t g^{\mathrm{T}} d$ for each $t \geq 0$, hence there is a contradiction. Therefore by contradiction $g \in \partial f_{i \in I}(x)$. □

Corollary 2.2.20. *Let f be a concave function defined on $\mathbb{R}^n$ as:*

$$f(x) = \inf_{\alpha \in \mathcal{A}} f_\alpha(x) \tag{2.80}$$

Where each f_α is concave and supdifferentiable. Suppose the supremum is for each x attained. In that case:

$$\partial^+ f(x) = \mathrm{Co}(\cup\{\partial^+ f_\alpha(x) \text{ s.t. } f(x) = f_\alpha(x)\}) \tag{2.81}$$

That is $\partial^+ f(x)$ is the convex hull of the supgradients of the active functions.

As a simple example of theorem 2.2.19:

Example 2.2.21. *Let $f_1 : \mathbb{R} \to \mathbb{R}, x \mapsto x^2$ and $f_2 : \mathbb{R} \to \mathbb{R}, x \mapsto (x-2)^2$ and $f(x) = \max_i f_i(x)$. The for $x < 1$, $\partial f(x) = 2(x-2)$, for $x > 1$, $\partial f(x) = 2x$, and for $x = 1$, $\partial f(x) = [-2, 2]$. This example is shown in figure 2.2.*

Another example of theorem 2.2.19, which shows that there should be a neighborhood at which the supremum is attained by a finite set of functions:

Example 2.2.22. *Let for $\zeta \in \mathbb{R}$:*

$$\begin{aligned} f_\zeta(x) &= (2\zeta + \frac{\zeta}{|\zeta|})x - \zeta^2 \quad \text{for} \quad \zeta \neq 0 \\ f_0(x) &= 0 \quad \text{for} \quad \zeta = 0 \end{aligned} \tag{2.82}$$

Then it can be checked that:

$$\begin{aligned} f(x) &= \max_{\zeta \in \mathbb{R}} f_\zeta(x) \\ &= f_x(x) \\ &= x^2 + |x| \end{aligned} \tag{2.83}$$

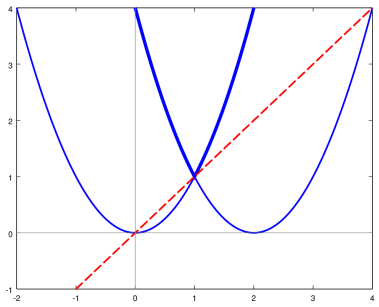

Figure 2.2: $f_1(x) = x^2$ and $f_2(x) = (x-2)^2$ are plotted with solid blue, $f(x) = \max(f_1(x), f_2(x))$ is plotted with thick blue, and a possible subgradients at $x = 1$ is plotted with dashed red

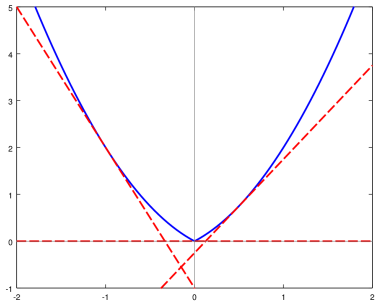

Figure 2.3: $f(x) = \max_{\zeta \in \mathbb{R}} f_\zeta(x))$ is plotted with thick blue, and in dashed red are plotted f_{-1}, f_0, and $f_{0.5}$

The function $f(x)$ is shown in figure 2.3.

Clearly $f(x)$ is convex, the maximum is attained at each point $x = \zeta$ by exactly one function in the set $\{f_\zeta\}$.

Moreover, the subgradient at $x = 0$ is: $\partial f(0) = [-1, 1]$, but the gradient of the active function at $x = 0$ is $\partial f_0(0) = 0$.

Note however that for each x there exists no neighborhood $\mathcal{N}(x)$ for which the collection $\{f_y | y \in \mathcal{N}(x)\}$ is finite. Hence theorem 2.2.19 was not applicable.

The subgradient of the maximum eigenvalue follows:

Corollary 2.2.23. *The subgradient of $\lambda_{\max}(A)$ is:*

$$\begin{aligned} \partial^- \lambda_{\max}(A) &= \{Z \in \mathcal{S}^n_{\succeq 0} \text{ s.t. } \langle A, Z\rangle = \lambda_{\max},\ \langle Z, I\rangle = 1\} \\ &= \mathrm{Co}\{qq^{\mathrm{T}} \text{ s.t. } q \in \mathbb{R}^n,\ Aq = \lambda_{\max} q,\ \|q\|_2 = 1\} \end{aligned} \tag{2.84}$$

Similar the supgradient of $\lambda_{\min}$ is:

$$\begin{aligned} \partial^+ \lambda_{\min}(A) &= \{Z \in \mathcal{S}^n_{\succeq 0} \text{ s.t. } \langle A, Z\rangle = \lambda_{\min},\ \langle Z, I\rangle = 1\} \\ &= \mathrm{Co}\{qq^{\mathrm{T}} \text{ s.t. } q \in \mathbb{R}^n,\ Aq = \lambda_{\min} q,\ \|q\|_2 = 1\} \end{aligned} \tag{2.85}$$

Proof. Notice that by the Rayleigh quotient $\lambda_{\max}(A) = \sup_{Z \in \mathcal{S}^n_{\succeq 0},\, \langle Z, I\rangle = 1} \langle A, Z\rangle$ and $\lambda_{\min}(A) = \inf_{Z \in \mathcal{S}^n_{\succeq 0},\, \langle Z, I\rangle = 1} \langle A, Z\rangle$ and also $\partial \langle \cdot\,, Z\rangle = Z$.

The corollary follows as a immediate consequence of theorem 2.2.19.

The second identity follows because the set $\{Z \in \mathcal{S}^n_{\succeq 0} \text{ s.t. } \langle A, Z\rangle = \lambda_{\max},\ \langle Z, I\rangle = 1\}$ can be described as the convex hull of $\{qq^{\mathrm{T}} \text{ s.t. } q \in \mathbb{R}^n,\ Aq = \lambda_{\min} q,\ \|q\|_2 = 1\}$, which finishes the proof. □

Corollary 2.2.24. *Define:*

$$f : \mathbb{R}^m \to \mathbb{R},\ y \mapsto \lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(y)) \tag{2.86}$$

Then the subgradient of $f(y)$ is given by:

$$\begin{aligned} \partial f(y) &= \{-\mathcal{A}(Z) \text{ s.t. } Z \in \mathcal{S}^n_{\succeq 0},\ \langle Z, C - \mathcal{A}^{\mathrm{T}}(y)\rangle = f(y),\ \langle Z, I\rangle = 1\} \\ &= \mathrm{Co}\{-\mathcal{A}(qq^{\mathrm{T}}) \text{ s.t. } q \in \mathbb{R}^n,\ (C - \mathcal{A}^{\mathrm{T}}(y))q = f(y)q,\ \|q\|_2 = 1\} \end{aligned} \tag{2.87}$$

Furthermore the directional derivative is given by:

$$\begin{aligned} \nabla_d f(y) &= \max_{v \in \partial f(y)} v^{\mathrm{T}} d \\ &= - \min_{\substack{q \in \mathbb{R}^n \\ (C - \mathcal{A}^{\mathrm{T}}(y))q = f(y)q \\ \|q\|_2 = 1}} \langle \mathcal{A}^{\mathrm{T}}(d), qq^{\mathrm{T}}\rangle \end{aligned} \tag{2.88}$$

Proof. The function f can be written as the composition $f = g \circ h(y)$ with $h : \mathbb{R}^m \to \mathcal{S}^n,\ y \mapsto C - \mathcal{A}^{\mathrm{T}}(y)$ and $g : \mathcal{S}^n \to \mathbb{R},\ X \mapsto \lambda_{\max}(X)$. Then by using the definition of the subgradient:

$$\begin{aligned} f(y) - f(y_0) = g(h(y)) - g(h(y_0)) &\geq \langle W, h(y) - h(y_0)\rangle \ \ (\forall W \in \partial g(h(y_0)) \\ &= \langle W, -\mathcal{A}^{\mathrm{T}}(y - y_0)\rangle \\ &= \langle -\mathcal{A}(W), y - y_0\rangle \end{aligned} \tag{2.89}$$

Hence the elements of $\partial f(y) = \{-\mathcal{A}(W) \text{ s.t. } W \in \partial g(h(y))\}$, which equals by corollary 2.2.23 equation 2.87.

The second equation follows easily:

$$\begin{aligned} \nabla_d f(y) &= \max_{v \in \partial f(y)} v^{\mathrm{T}} d \\ &= \max_{W \in \partial g(h(y))} \langle -\mathcal{A}(W), d\rangle \\ &= -\min_{W \in \partial g(h(y))} \langle W, \mathcal{A}^{\mathrm{T}}(d)\rangle \\ &= -\min_{\substack{q \in \mathbb{R}^n \\ (C - \mathcal{A}^{\mathrm{T}}(y))q = f(y)q \\ \|q\|_2 = 1}} \langle \mathcal{A}^{\mathrm{T}}(d), qq^{\mathrm{T}}\rangle \end{aligned} \tag{2.90}$$

□

Chapter 3

Sensitive Pole Algorithm

There are different approaches to finding the intersecting point of two eigencurves. One approach is through a discrete line search such as a bisection method, however that is computational expensive. Another approach is to calculate a full eigendecomposition of the initial matrix and use linear projections to estimate where the crossingpoint is. That is also computational expensive, but for small matrices less expensive than bisection. However, this approach is unable to take non-simple eigenvalues in consideration because the gradient is not well defined.

The Sensitive Pole Algorithm is another approach based on linearization of the eigencurves which takes in consideration eigenvalues with a large (sub)-gradient. It only computes those eigencurves which are likely to intersect the maximum eigencurve and thus is faster than a full eigendecomposition.

This algorithm will form an important part of a novel algorithm to solve eigenvalue optimization problems in chapter 4.

The Sensitive Pole Algorithm is closely related to the Dominant Pole Algorithm and the Rayleigh Quotient Iteration. The former was actually developed through modifying the Dominant Pole Algorithm, hence it is instructive to explain first how the Rayleigh Quotient Iteration and the Dominant Pole Algorithm works. In the first section Rayleigh Quotient Iteration is explained, second the Dominant Pole Algorithm will be explained. The third section will modify the Dominant Pole Algorithm to a subspace method for better control of convergence and faster convergence. In the fourth section the Sensitive Pole Algorithm will be explained. Finally, in the fifth section will be explained on how to use the Sensitive Pole Algorithm to compute the largest eigenvalue of a matrix as function of perturbation.

Most of this section is based on the work of Rommes and Sleijpen [28]. Rommes and Sleijpen consider the computation of simple dominant poles of a transfer function as application to model order reduction.

The Dominant Pole Algorithm as described in [28] works also on asymmetric matrices with generalized eigenvalues of matrix pair (A, E). Instead for application in chapter 4, it is sufficient to consider symmetric matrices and restrict

to the classical eigenvalues of A.

3.1 Rayleigh Quotient Iteration

Rayleigh Quotient Iteration (RQI) allows fast computation of eigenvalues with corresponding eigenvectors if an eigenvalue and vector are roughly known. For instance near the end of other iterative eigenvalue solvers which are slower, a few RQI steps can drastically increase the precision. That can be useful for example when high precision is necessary for deflating a subspace.

Let $A \in \mathcal{S}^n$, the objective of RQI is to compute an eigenvalue with corresponding eigenvector roughly in direction $x_0 \in \mathbb{R}^n$. RQI works through repeatedly calculating the Rayleigh quotient (theorem 2.2.3) and using the result as the new shift:

Algorithm 3.1.1 Rayleigh Quotient Iteration

Require:
$A \in \mathcal{S}^n$: matrix
$x_0 \in \mathbb{R}^n$: initial normalized vector
$\epsilon \in \mathbb{R}_{>0}$: tolerance
Ensure:
$\lambda \in \mathbb{R}$: eigenvalue of A
$x \in \mathbb{R}^n$: corresponding eigenvector

1: $k \leftarrow 0$, $s_0 \leftarrow x_0^{\mathrm{T}} A x_0$
2: **while** $\|Ax_k - s_k x_k\| \geq \epsilon$ **do**
3: Solve $(A - s_k I)x_{k+1} = x_k$ for $x_{k+1} \in \mathbb{R}^n$
4: Normalize x_{k+1}: $x_{k+1} \leftarrow \frac{x_{k+1}}{\|x_{k+1}\|}$
5: Take the Rayleigh Quotient: $s_{k+1} \leftarrow x_{k+1}^{\mathrm{T}} A x_{k+1}$
6: $k \leftarrow k + 1$
7: **end while**
8: $\lambda \leftarrow s_k$, $x \leftarrow x_k$

It can be proven that if the Rayleigh Quotient Iteration converges, then it converges cubicly. This proof will not be repeated here but can be found in [20] and [23].

3.2 Dominant Pole Algorithm

The Dominant Pole Algorithm can be understood as Rayleigh Quotient Iteration with fixed right hand to steer convergence to eigenvalues with corresponding eigenvector components in the initial vector:

Algorithm 3.2.1 Dominant Pole Algorithm

Require:
$A \in \mathcal{S}^n$: matrix

$b \in \mathbb{R}^n$: residue vector
$s_0 \in \mathbb{R}$: initial pole estimate
$\epsilon \in \mathbb{R}_{>0}$: tolerance
Ensure:
$\lambda \in \mathbb{R}$: dominant pole (eigenvalue of A)
$x \in \mathbb{R}^n$: corresponding eigenvector

1: $k \leftarrow 0$
2: **while** $\|Ax_k - s_k x_k\| \geq \epsilon$ **do**
3: Solve $(A - s_k I)x_{k+1} = b$ for $x_{k+1} \in \mathbb{R}^n$
4: Normalize x_{k+1}: $x_{k+1} \leftarrow \frac{x_{k+1}}{\|x_{k+1}\|}$
5: Take the Rayleigh Quotient: $s_{k+1} \leftarrow x_{k+1}^{\mathrm{T}} A x_{k+1}$
6: $k \leftarrow k + 1$
7: **end while**
8: **return** $\lambda \leftarrow s_k$, $x \leftarrow x_k$

Observe the similarity with algorithm 3.1.1, only the third line is different. The effect is that Dominant Pole Algorithm can be steered towards converging to eigenvalues with large eigenvector components in $b \in \mathbb{R}^n$.

The Dominant Pole Algorithm does not converge cubicly, as the Rayleigh Quotient Iteration, but quadraticly:

Lemma 3.2.2. *Consider algorithm 3.2.1. Let $\lambda \in \mathbb{R}$ be an eigenvalue of A to which the algorithm converges, then $s_k \to \lambda$ quadraticly.*

Proof. It can be shown that s_k follows a Newton scheme:

Define $H : \mathbb{R} \to \mathbb{R}, s \mapsto b^{\mathrm{T}}(A - sI)^{\dagger} b$. Note that $\lim_{s \to \lambda} |H(S)| = \infty$. Also, let $G(s) = H(s)^{-1}$, then the roots of $G(s)$ are exactly the eigenvalues of A for which there exists an vector in the eigenspace $q \in \mathbb{R}^n$ such that $b^{\mathrm{T}} q \neq 0$. Next it is shown that the Dominant Pole Algorithm is a Newton Method used to compute the roots of $G(s)$, which implies that $s \to \lambda$ quadraticly:

The derivative of $G(s)$ to s is given by:

$$G'(s) = -\frac{H'(s)}{H^2(s)} \tag{3.1}$$

And the derivative of $H(s)$ to s is given by:

$$H'(s) = b^{\mathrm{T}}(A - sI)^{\dagger}(A - sI)^{\dagger} b \tag{3.2}$$

(This can easily be checked by using the identity $(A - sI)^{\dagger} = \sum_i \frac{u_i u_i^{\mathrm{T}}}{\lambda_i - s}$ in which λ_i is an eigenvalue of A with corresponding eigenvector u_i.)

Both those equations lead to the following Newton scheme:

$$\begin{aligned} s_{k+1} &= s_k + \frac{G(s_k)}{G'(s_k)} \\ &= s_k + \frac{1}{H(s_k)} \frac{H^2(s_k)}{H'(s_k)} \\ &= s_k + \frac{b^{\mathrm{T}}(A - s_k I)^{\dagger} b}{b^{\mathrm{T}}(A - s_k I)^{\dagger}(A - s_k I)^{\dagger} b} \end{aligned} \tag{3.3}$$

Let $x_k = (A - s_k I)^{\dagger} b$, then equation 3.3 can be simplified further:

$$\begin{aligned} s_{k+1} &= s_k + \frac{b^{\mathrm{T}}(A - s_k I)^{\dagger} b}{b^{\mathrm{T}}(A - s_k I)^{\dagger}(A - s_k I)^{\dagger} b} \\ &= s_k + \frac{b^{\mathrm{T}} x_k}{x_k^{\mathrm{T}} x_k} \\ &= \frac{s_k x_k^{\mathrm{T}} x_k + b^{\mathrm{T}} x_k}{x_k^{\mathrm{T}} x_k} \\ &= \frac{(s_k x_k^{\mathrm{T}} + x_k^{\mathrm{T}}(A - s_k I)) x_k}{x_k^{\mathrm{T}} x_k} \\ &= \frac{x_k^{\mathrm{T}} A x_k}{x_k^{\mathrm{T}} x_k} \end{aligned} \tag{3.4}$$

The implementation of this Newton scheme is exactly algorithm 3.2.1. Hence in the Dominant Pole Algorithm, s_k converges quadraticly. □

Theorem 3.2.3. *Consider algorithm 3.2.1. Let $\lambda \in \mathbb{R}$ be a simple eigenvalue of A to which the algorithm converges, and let $x \in \mathbb{R}^n$ be its corresponding normalized eigenvector. If $b^{\mathrm{T}} x \neq 0$, then $x_k \to x$ up to a scalar, quadraticly.*

Proof. Let $\tau_k \in \mathbb{R}$ be a scalar such that:

$$\tau_k x_k = x + e_k \tag{3.5}$$

With $e_k \perp x$, the 'error' vector. That is, x_k is scaled such that $\tau_k x_k^{\mathrm{T}} x = 1$. It can be shown that for a sufficient large k, $\frac{1}{\tau_k}\|e_k\| \to 0$ quadraticly:

$$\begin{aligned} \|e_k\| &= \frac{1}{\tau_k} \|(I - x x^{\mathrm{T}})(A - s_k I)^{\dagger} b\| \\ &= \frac{1}{\tau_k} \| \sum_r \frac{u_r (u_r^{\mathrm{T}} b)}{\lambda_r - s} \| \\ &\leq \frac{C}{\tau_k} \end{aligned} \tag{3.6}$$

Hence $x_k = \tau_k^{-1}x + \tau_k^{-1}e_k \to \tau_k^{-1}x$ as $k \to \infty$ quadraticly iff $\tau_k^{-1} \to 0$ as $k \to \infty$ quadraticly. The latter statement is shown to be true:

Note that algorithm 3.2.1 computes:

$$x_k = (A - s_k I)^{\dagger} b = \frac{u_i u_i^{\mathrm{T}} b}{\lambda_i - s_k} \tag{3.7}$$

Hence, let $s_k \to \lambda\ (= \lambda_j)$ and $x\ (= u_j)$ be the corresponding eigenvector, then τ_k is of the form:

$$\tau_k = \frac{x^{\mathrm{T}} b}{\lambda - s_k} \tag{3.8}$$

Moreover, as $s_k \to \lambda$ quadraticly as was shown in lemma 3.2.2, it follows that $\tau_k^{-1} \to 0$ quadraticly. □

3.3 Subspace accelerating with deflation

Most of this section is based on [26]. Novel is the idea of considering different selection strategies and the inclusion of a Rayleigh Quotient Iteration step.

It is possible to accelerate algorithm 3.2.1 by modifying it into a subspace method. The main idea is that the vectors $x_1, \ldots, x_k$ are used to construct subspace X_k which spans the eigenspace associated with the eigenvalue to converge to. The matrix $S_k := X_k^{\mathrm{T}} A X_k$ is the projection of A onto X_k. The eigenvalue of S_k converges to the dominant eigenvalue of A as the span of X_k comes closer to the eigenspace associated with that eigenvalue.

S_k has different eigenvalues which are the projected eigenvalues of A. Based on some selection rule we select one of those and try to expand X_{k+1} such that in the next iteration X_k has a better cover of the corresponding eigenspace.

In standard Dominant Pole Algorithm that would be through taking the inproduct between the eigenvectors of S_k and b, then the most dominant is selected. In standard SASPA, introduced in the next section, all eigenvalues would be tested on sensitivity and the most sensitive would be selected.

The Subspace accelerated Dominant Pole Algorithm is:

Algorithm 3.3.1 Subspace Accelerated Dominant Pole Algorithm

Require:

$A \in \mathcal{S}^n$: matrix
$b \in \mathbb{R}^n$: residue vector
$s_0 \in \mathbb{R}$: initial pole estimate
$\delta \in \mathbb{R}_{>0}$: threshold from where to start Rayleigh Quotient Iteration
$\epsilon \in \mathbb{R}_{>0}$: tolerance
p, number of poles to compute
$k_{\min}, k_{\max}$ minimum and maximum size of search space for restart

Ensure:

$\lambda_1, \ldots, \lambda_p \in \mathbb{R}$: eigenvalues of A
$q_1, \ldots, q_p \in \mathbb{R}^n$: corresponding eigenvectors

1: $k \leftarrow 0$, $i \leftarrow 1$, $X \leftarrow []$
2: **while** $i \leq p$ **do**
▷ Extend search space
3: Solve $(A - s_k I)x_k = b$ for $x_k \in \mathbb{R}^n$
4: $x_k \leftarrow \mathrm{GS}(X, x_k)$, $X \leftarrow [X, x_k]$ ▷ GS: Stabilized Gram-Schmidt
▷ Select most promising
5: $S \leftarrow X^{\mathrm{T}} A X$
6: $(\tilde{\lambda}, \tilde{x}) \leftarrow \mathrm{Select}(S, X^{\mathrm{T}} b)$
7: $q_i \leftarrow X^{\mathrm{T}} \tilde{x}$
▷ Elective Rayleigh Quotient Iteration
8: **if** $\|Aq_i - \tilde{\lambda} q_i\| \leq \delta$ **then**
9: **while** $\|Aq_i - \tilde{\lambda} q_i\| \geq \epsilon$ **do**
10: Solve $(A - \tilde{\lambda} I)\dot{q}_i = q_i$ for $\dot{q}_i \in \mathbb{R}^n$
11: $q_i \leftarrow \frac{\dot{q}_i}{\|\dot{q}_i\|}$
12: $\tilde{\lambda} \leftarrow q_i^{\mathrm{T}} A q_i$
13: **end while**
14: **end if**
▷ Check convergence and deflate
15: **while** $\|Aq_i - \tilde{\lambda} q_i\| \leq \epsilon$ **do**
16: $\lambda_i \leftarrow \tilde{\lambda}$
17: Deflate b, and X with q_i
18: $i \leftarrow i + 1$
19: $\tilde{\lambda} \leftarrow \mathrm{Select}(S, X^{\mathrm{T}} b)$
20: $q_i \leftarrow X^{\mathrm{T}} \tilde{x}$
21: **end while**
▷ Check size search space and restart
22: **if** #columns of X is more than $k_{\max}$ **then**
23: $S \leftarrow X^{\mathrm{T}} A X$
24: $((\mu_1, u_1), \ldots, (\mu_{k_{\min}}, u_{k_{\min}})) \leftarrow \mathrm{Select}_{k_{\min}}(S, X^{\mathrm{T}} b)$
25: $\dot{X} \leftarrow X^{\mathrm{T}}[u_1, \ldots, u_{k_{\min}}]$
26: Orthogonalize $\dot{X}$
27: $X \leftarrow \dot{X}$
28: **end if**

29: $s_{k+1} \leftarrow \tilde{\lambda}$
30: $k \leftarrow k + 1$
31: **end while**

A quick explanation of the inner loop of algorithm 3.3.1:

- Line 3: determine the vector x_k which will be used to expand the search spaces.
- Line 4: use a stable variant of Gram-Schmidt to orthogonalize the new vectors to the existing vectors in the search space, and add them.

- Line 5-7: select poles to converge to from the search space, for an example algorithm see 3.3.2.
- Line 8-14: elective Rayleigh Quotient Iteration for faster convergence, see algorithm 3.1.1.
- Line 15-21: check if the algorithm has converged, if so deflate and continue (deflating will be explained later on in this section).
- Line 22-28: restart of the search space.

As a side-note: The algorithm written here starts with an empty search space, however it can also be chosen to be filled already with vectors that are expected to be near optimum. For example vectors known from previous computations. Moreover, if such vectors are unknown, then it can also be chosen to fill the search space by Lanczos/Arnoldi vectors (that is a basis for the Krylov subspace $\mathcal{K}_k(A, b)$).

In simulations the fastest convergence was obtained by filling the initial search space with a few Lanczos/Arnoldi vectors.

3.3.1 Selecting poles

The matrix S is of low dimension and stems from the projected eigenvalue problem:

$$(X^{\mathrm{T}}AX)\tilde{x} = \tilde{\lambda}\tilde{x} \tag{3.9}$$

Hence, for any eigenvector $\tilde{x}$ of S, the vector $x = X^{\mathrm{T}}\tilde{x}$ is an eigenvector of A projected on X^{T}. As the number of columns of X increase, the dimension of the projected spaces grow, which in turn implies that the approximations get closer. The idea is that S contains eigenvectors and eigenvalues which will under the right transformation increasingly precision approximate eigenvectors of A with corresponding eigenvalues.

Selecting poles for the low dimension system S is done by computing an eigendecomposition and with a dense eigenvalue method such as the QR-algorithm.

For computation of the dominant pole as in algorithm 3.2.1, that would be:

Algorithm 3.3.2 Selection strategy

Require:
$S \in \mathbb{R}^{m\times m}$: matrix
$b \in \mathbb{R}^n$: residue vector
Ensure:
$\lambda \in \mathbb{R}$: dominant pole (eigenvalue of S)
$x \in \mathbb{R}^n$: corresponding eigenvector

1: Compute eigenpairs of S: (λ_i, x_i)
2: Compute $R_i \leftarrow (x_i^{\mathrm{T}}b)^2$ for all i
3: $j \leftarrow \arg\max_i |R_i|$

4: $\lambda \leftarrow \lambda_j$, $x \leftarrow x_j$

Algorithm 3.3.2 is a sample selection-rule. Other selection rules include $\arg\max_i \frac{|R_i|}{|\lambda_i|}$ and $\arg\max_i \frac{|R_i|}{|C-\lambda_i|}$.

However, note that by incorporating a selection strategy possible quadraticly convergence is lost. That is because on line 15: $s_{k+1} \leftarrow \lambda$, hence s_k does not follow the Newton scheme. For the proposed selection rule (algorithm 3.3.2) this poses no problem because it can be proved that this s_{k+1} is closer to the pole than in algorithm 3.2.1.

If using a selection strategy which questionable converges, for instance which forces the algorithm to cycle between different poles without reaching one with enough precision, then it is recommended to use the Rayleigh Quotient Iteration step (line 10-16). This stops the algorithm from cycling because the poles that the algorithm cycles between are in an early stadium deflated from the search space. That allows the algorithm to converge to the other, more favorable, poles without hindrance of the other poles.

3.3.2 Deflating

Deflating means removing a vector component out of the system. For instance, deflating A with x_q means:

$$A \leftarrow (I - x_q x_q^{\mathrm{T}}) A (I - x_q x_q^{\mathrm{T}}) \tag{3.10}$$

That way the eigenvalue corresponding to x_q s reduced to 0. However, if A was sparse, then after applying equation 3.10, it surely is not and it is computational more difficult to solve equations of the form $(A - sI)x = b$ for $x \in \mathbb{R}^n$.

Therefore a better choice is to not deflate A but instead deflate b:

$$b \leftarrow (I - x_q x_q^{\mathrm{T}}) b \tag{3.11}$$

This stops the algorithm from converging again to the already found pole because $(b^{\mathrm{T}}(I - x_q x_q^{\mathrm{T}})^{\mathrm{T}} x_q)(x_q^{\mathrm{T}}(I - x_q x_q^{\mathrm{T}})b) = 0$.

Note that this poses no stability problem, although x_q might contain a large nummerical error, because this step only happens once after convergence. After that the eigenvector component in b of the dominant pole is diminished such that the algorithm will not converge again towards that pole.

Also to make sure that the pole does not pop up again during the selection stage in the next iteration, deflate X:

$$X \leftarrow (I - x_q x_q^{\mathrm{T}}) X \tag{3.12}$$

By deflating X, orthogonality of the columns will be lost. Hence they have to be reorthogonalized after deflation. That can be done through a stabilized version of Gram-Schmidt.

3.3.3 Restart search space

If the search space grows too large, then the selecting of poles on line 7-9 becomes computational more expensive. Hence by incorporating an implicit strategy for the search space it is made sure that it is always of low dimensional.

The selection criteria for the $k_{\min}$ vectors to be included in the search space should be similar to the selection strategy on line 7-9. A suitable choice is to order them all during line 8 to some favorable ranking, return the most favorable and then on line 24 select the top $k_{\min}$ most favorable vectors.

Note that if the system has converged and some vector is being deflated from the system, then the dimension of the search space is most likely also decreased by one. Thus the results of line 7-8 are almost surely applicable during line 23-24, which might save some computation time. However, if the Rayleigh Quotient Iteration step is incorporated, then it follows not necessarily that the search space will be decreased. Therefore if the Rayleigh Quotient Iteration is being elected, then either line 23-24 have to computed from scratch, or the implicit restart has to be canceled for one iteration, i.e. change the 'if' on line 22 to an 'else if'.

After selecting the vectors to be included in the new search space, they should be orthogonalized. Again, that can be done through applying Givens rotations or any other stable Gram-Schmidt variant on their columns. For stability it is recommended to not skip this step although it might be expected that u_1 to $u_{k_{\min}}$ are already orthogonal because they are eigenvectors.

3.4 Sensitive Pole Algorithm

This section is largely inspired by [27]. As with the previous section, novel is the inclusion of the Rayleigh Quotient Iteration step.

The Sensitive Pole Algorithm tries to compute the poles of $A \in \mathcal{S}^n$ with a large derivative under perturbation. That is, let $\dot{A}$ denote the perturbation of A. Then we are interested in the eigenvalues of the function:

$$f(t) := A + t\dot{A} \tag{3.13}$$

The derivative of a simple eigenvalue $\lambda_i \in \mathbb{R}$ with corresponding normalized eigenvector $u_i \in \mathbb{R}^n$ of A is given by:

$$\frac{\mathbf{d}\,\lambda_i(f(t))}{\mathbf{d}\,t} = t u_i^{\mathrm{T}} \dot{A} u_i \tag{3.14}$$

(Theorem 2.2.7.)

The Sensitive Pole Algorithm computes the eigenvalues $\lambda_i \in \mathbb{R}$ for which $|\frac{\mathbf{d}\,\lambda_i(f(t))}{\mathbf{d}\,t}|$ is considerable large in comparison with the other eigenvalues.

It does that by replacing $b \in \mathbb{R}^n$ in algorithm 3.2.1:

$$b \leftarrow \dot{A} x_{k-1} \tag{3.15}$$

The motivation for doing so is that ideally one wants to compute the most sensitive pole which is the eigenvalue λ_i with a large residual ($|u_i^{\mathrm{T}} \dot{A} u_i|$) for some $u_i \in \mathbb{R}^n$ in the eigenspace of λ_i. If $b = \dot{A}u_i$, then the residual at step k is $(b^{\mathrm{T}}x_k)(x_k^{\mathrm{T}})b = (u_i^{\mathrm{T}}\dot{A}x_k)(x_k^{\mathrm{T}}\dot{A}u_i)$, which converges to $(u_i^{\mathrm{T}}\dot{A}v_i)^2$, with both u_i and v_i in the eigenspace corresponding to λ_i. However to compute the residual the eigenvector u_i is necessary. The best approximation of u_i at step k is x_{k-1}, hence the choice $b \leftarrow \dot{A}x_{k-1}$.

The full algorithm for Sensitive Pole Algorithm is:

Algorithm 3.4.1 Sensitive Pole Algorithm

Require:
$A \in \mathcal{S}^n$: matrix
$\dot{A} \in \mathcal{S}^n$: perturbation
$s_0 \in \mathbb{R}$: initial pole estimate
$\epsilon \in \mathbb{R}_{>0}$: tolerance
Ensure:
$\lambda \in \mathbb{R}$: sensitive pole (eigenvalue of A)
$x \in \mathbb{R}^n$: corresponding eigenvector

1: $k \leftarrow 0$, $x_{-1} = (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$
2: **while** not converged **do**
3: $\quad b \leftarrow \dot{A}x_{k-1}$
4: $\quad$ Solve $(A - s_k I)x_k = b$ for $x_k \in \mathbb{R}^n$
5: $\quad s_{k+1} \leftarrow s_k + \frac{c^{\mathrm{T}}x_k}{x_k^{\mathrm{T}}x_k}$ ▷ $(\frac{x_k^{\mathrm{T}}Ax_k}{x_k^{\mathrm{T}}x_k})$
6: $\quad$ **if** $\|Ax_k - s_{k+1}x_k\| \leq \epsilon$ **then**
7: $\quad\quad$ **return** $\lambda \leftarrow s_{k+1}$, $x \leftarrow x_k$
8: $\quad$ **end if**
9: $\quad k \leftarrow k + 1$
10: **end while**

And also based on algorithm 3.3.1, a subspace accelerated variant:

Algorithm 3.4.2 Subspace Accelerated Sensitive Pole Algorithm

Require:
$A \in \mathcal{S}^n$: matrix
$\dot{A} \in \mathcal{S}^n$: perturbation
$s_0 \in \mathbb{R}$: initial pole estimate
$\delta \in \mathbb{R}_{>0}$: threshold from where to start Rayleigh Quotient Iteration
$\epsilon \in \mathbb{R}_{>0}$: tolerance
p, number of poles to compute
$k_{\min}, k_{\max}$ minimum and maximum size of search space for restart
Ensure:
$\lambda_1, \ldots, \lambda_p \in \mathbb{R}$: dominant sensitive poles (eigenvalues of A)
$q_1, \ldots, q_p \in \mathbb{R}^n$: corresponding eigenvectors

1: $k \leftarrow 0$, $i \leftarrow 1$, $x_{-1} = (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$

2: **while** $i \leq p$ **do**

▷ Extend search space

3: $b \leftarrow \dot{A}x_{k-1}$

4: **if** $i \geq 2$ **then** ▷ Deflate b with $q_1, \ldots, q_{i-1}$

5: $b \leftarrow \mathrm{GS}([q_1, \ldots, q_{i-1}], b)$

6: **end if**

7: Solve $(A - s_k I)x_k = b$ for $x_k \in \mathbb{R}^n$

8: $x_k \leftarrow \mathrm{GS}(X, x_k)$, $X \leftarrow [X, x_k]$

▷ Select most promising

9: $S \leftarrow X^{\mathrm{T}} A X$

10: $(\tilde{\lambda}, \tilde{x}) \leftarrow \mathrm{Select}(S, X^{\mathrm{T}} b)$

11: $q_i \leftarrow X^{\mathrm{T}} \tilde{x}$, $q_i \leftarrow \frac{q_i}{\|q_i\|}$

▷ Elective Rayleigh Quotient Iteration

12: **if** $\|Aq_i - \tilde{\lambda} q_i\| \leq \delta$ **then**

13: **while** $\|Aq_i - \tilde{\lambda} q_i\| \geq \epsilon$ **do**

14: Solve $(A - \tilde{\lambda} I)\dot{q_i} = q_i$ for $\dot{q_i} \in \mathbb{R}^n$

15: $q_i \leftarrow \frac{\dot{q_i}}{\|\dot{q_i}\|}$

16: $\tilde{\lambda} \leftarrow q_i^{\mathrm{T}} A q_i$

17: **end while**

18: **end if**

▷ Check convergence and deflate

19: **if** $\|Aq_i - \tilde{\lambda} q_i\| \leq \epsilon$ **then**

20: $\lambda_i \leftarrow \tilde{\lambda}$

21: Deflate X with q_i

22: $i \leftarrow i + 1$

23: **end if**

▷ Check size search space and implicit restart

24: **if** #columns of X is more than $k_{\max}$ **then**

25: $S \leftarrow X^{\mathrm{T}} A X$

26: $((\mu_1, u_1), \ldots, (\mu_{k_{\min}}, u_{k_{\min}})) \leftarrow \mathrm{Select}_{k_{\min}}(S, X^{\mathrm{T}} b)$

27: $\dot{X} \leftarrow X^{\mathrm{T}}[u_1, \ldots, u_{k_{\min}}]$

28: Orthogonalize $\dot{X}$

29: $X \leftarrow \dot{X}$

30: **end if**

31: $s_{k+1} \leftarrow \lambda$

32: $k \leftarrow k + 1$

33: **end while**

3.5 Computing cross points of eigencurves

One of the problems that has to be dealt with during eigenvalue optimization is to estimate when the largest eigenvalue gets overtaken by another eigenvalue during a perturbation, i.e. when two eigenvalue curves cross.

For clarity, in this section the following problem is considered: let $A \in \mathcal{S}^n$

be the unperturbed matrix, $\dot{A} \in \mathcal{S}^n$ the perturbation and $A(t) = A + t\dot{A}$ the perturbation function. By theorem 2.2.12 it is possible to label all eigenvalues of $A(t)$ as $\lambda_i(t)$ with at $t = 0$:

$$\lambda_1(0) \geq \lambda_2(0) \geq \cdots \geq \lambda_n(0) \tag{3.16}$$

(If at $t = 0$, an eigenvalue has multiplicity τ, then there is a sequence $\lambda_i(0) = \cdots = \lambda_{i+\tau-1}(0)$.)

The functions $\lambda_i(t)$ for $t \geq 0$ are called eigencurves.

Let further on in this section τ denote the multiplicity of λ_1 at $t = 0$. The problem is stated as estimating what the minimal $t^* > 0$ is for which there exists an index $i^* > \tau$ such that $\lambda_1(t^*) = \lambda_{i^*}(t^*)$:

If A is small with only simple eigenvalues, then the problem can be solved by performing a full eigendecomposition of A such that for each $\lambda_i(t)$ a unique eigenvector $u_i \in \mathbb{R}^n$ at $t = 0$ is known. Then all directional derivatives $v_i = u_i^{\mathrm{T}} \dot{A} u_i$ (theorem 2.2.7) are calculated and let $T = \{(\frac{\lambda_1(0)-\lambda_i(0)}{v_i-v_1}, i) \text{ s.t. } i > 1\}$:

$$(t^*, i^*) = \underset{(t,i)\in T}{\arg\min}\ t \text{ s.t. } t > 0 \tag{3.17}$$

However, A is for many instances large and sparse with possible non-simple eigenvalues. In such cases not all derivatives of the eigencurves are known, nor is a full eigendecomposition feasible to compute.

Despite that a full eigendecomposition is infeasible, it is still possible to calculate equation 3.17: with the Sensitive Pole Algorithm it is possible to compute a subset of T which contains the most promising elements.

By using algorithm 3.4.2 with some kind of deflation and the standard selection rule it is possible to calculate the eigencurves with a large absolute derivative. By employing a selection rule based on the linear approximation of the intersections convergence is steered to more promising eigencurves.

The modified selection rule is:

Algorithm 3.5.1 Selection strategy for intersecting maximum eigencurve

Require:
$S \in \mathbb{R}^{m\times m}$: matrix
$\dot{S} \in \mathbb{R}^{m\times m}$: perturbation
$\lambda_{\max} \in \mathbb{R}$: maximum eigenvalue at $t = 0$
$v_{\max} \in \mathbb{R}$: derivative $\frac{\mathbf{d}\,\lambda_{\max}}{\mathbf{d}\,t}$
Ensure:
$\lambda \in \mathbb{R}$: sensitive pole (eigenvalue of S)
$x \in \mathbb{R}^n$: corresponding eigenvector

1: Compute eigenpairs of S: (λ_i, x_i)
2: Compute $R_i \leftarrow \frac{\lambda_{\max}-\lambda_i}{x_i^{\mathrm{T}}\dot{S}x_i - v_{\max}}$ for all i
3: $j \leftarrow \arg\max_i R_i$ s.t. $R_i > 0$
4: $\lambda \leftarrow \lambda_j$, $x \leftarrow x_j$

Moreover, as it is unpractical to run 3.4.2 for a long time it is necessary to think of a stopping criteria. It is undesirable to stop the method too soon since than it is unlikely to have found the crossing eigencurve, on the other hand it is also desired to stop as soon as possible and not waste computing cycles to non-nearest eigencurves.

In simulations it was chosen to stop the algorithm as soon as the converged to eigencurve has the nearest intersection point ever seen by the algorithm.

The full algorithm:

Algorithm 3.5.2 SASPA to compute nearest intersection

Require:

$A \in \mathcal{S}^n$: matrix
$\dot{A} \in \mathcal{S}^n$: perturbation
$\lambda \in \mathbb{R}$: largest eigenvalue at $t = 0$
$q_0, \ldots, q_k \in \mathbb{R}^n$: corresponding eigenvectors
$f \in \mathbb{R}^n$: forward derivative largest eigencurve
$\epsilon_{RQI} \in \mathbb{R}_{>0}$: threshold from where to start Rayleigh Quotient Iteration
$\epsilon_{tol} \in \mathbb{R}_{>0}$: tolerance
$k_{\min}, k_{\max}$ minimum and maximum size of search space for restart
$iter_{\max}$, maximum number of iterations

Ensure:

$\hat{t} \in \mathbb{R}_+$: intersection point
$\hat{\lambda} \in \mathbb{R}$: nearest crossing eigenvalues at $t = 0$
$\hat{q} \in \mathbb{R}^n$: corresponding eigenvector

$iter \leftarrow 1$, $num \leftarrow k$, $s \leftarrow \lambda$, $x \leftarrow (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$, $\hat{t} \leftarrow \infty$, $\tau \leftarrow \infty$
while $iter \leq iter_{\max}$ **do**
    ▷ Extend search space
    $b \leftarrow \dot{A}x$
    $b \leftarrow \mathrm{GS}([q_0, \ldots, q_{num}], b)$ ▷ Deflate b with $q_0, \ldots, q_{num}$
    Solve $(A - sI)x = b$ for $x \in \mathbb{R}^n$
    $x \leftarrow \mathrm{GS}(X, x)$, $X \leftarrow [X, x]$
    ▷ Select most promising
    $S \leftarrow X^{\mathrm{T}}AX$
    Compute all eigenpairs (s_i, x_i) of S
    Compute for all pairs corresponding cross points: $t_i \leftarrow \frac{\lambda - s_i}{x_i^{\mathrm{T}} X^{\mathrm{T}} \dot{A} X x_i - q_0^{\mathrm{T}} \dot{A} q_0}$
    Reorder all triplets (t_i, s_i, x_i) such that $t_i \geq t_{num}$ for all $i \leq j$
    $(t, s, x) \leftarrow (t_1, s_1, Xx_1)$
    $x \leftarrow \frac{x}{\|x\|}$
    ▷ Elective Rayleigh Quotient Iteration
    if $\|Ax - sx\| \leq \epsilon_{RQI}$ **then**
        while $\|Ax - sx\| \geq \epsilon_{tol}$ **do**
            Solve $(A - sI)\dot{x} = x$ for $\dot{x} \in \mathbb{R}^n$
            $x \leftarrow \frac{\dot{x}}{\|\dot{x}\|}$
            $s \leftarrow x^{\mathrm{T}}Ax$

$t \leftarrow \frac{\lambda - s}{x^{\mathrm{T}} \dot{A} x_i - q_0^{\mathrm{T}} \dot{A} q_0}$
end while
end if
▷ Update stop criteria
$\tau \leftarrow \min(\tau, t)$
▷ Check convergence and deflate
if $\|Ax - sx\| \leq \epsilon_{tol}$ **then**
$num \leftarrow num + 1$
$q_{num} \leftarrow x$
Deflate X with q_{num}
if $t \leq \hat{t}$ **then**
$\hat{\lambda} \leftarrow s$, $\hat{q} \leftarrow q_{num}$, $\hat{t} \leftarrow t$
end if
if $t \leq \tau$ **then**
return
end if
$(t, s, x) \leftarrow (t_2, s_2, Xx_2)$
$x \leftarrow \frac{x}{\|x\|}$
end if
▷ Check size search space and implicit restart
if #columns of X is more than $k_{\max}$ **then**
$X \leftarrow [Xx_1, Xx_2, \ldots, Xx_{k_{\min}}]$
Orthogonalize and normalize X
end if

$iter \leftarrow iter + 1$
end while

The `Matlab`implementation being used is included in appendix A.2.

3.5.1 Numerical results

As a demonstration of algorithm 3.5.2, some numerical results are presented.

The described experiment is with A taken from the 10th DIMACS challenge [3]. The matrix A is chosen to be the Laplacian matrix from a graph in the Random Geometric Graphs set. That are random geometric graphs labeled `rgg_n_2_X_s0.graph`, each with 2^X vertices with X ranging from 15 up to 24. Each vertex is a random point in the unit square and edges connect vertices whose Euclidean distance is below $0.55\frac{\log(n)}{n}$.

The non-zero elements of the Laplacian matrix of `rgg_n_2_15_s0.graph` are shown in figure 3.1.

As perturbation $\dot{A}$, a random sparse diagonal matrix is chosen with elements uniform distributed between -1 and 1. If the largest eigenvalue has a positive gradient, then $\dot{A}$ is flipped, that way as t increases the largest eigenvalue will always decrease.

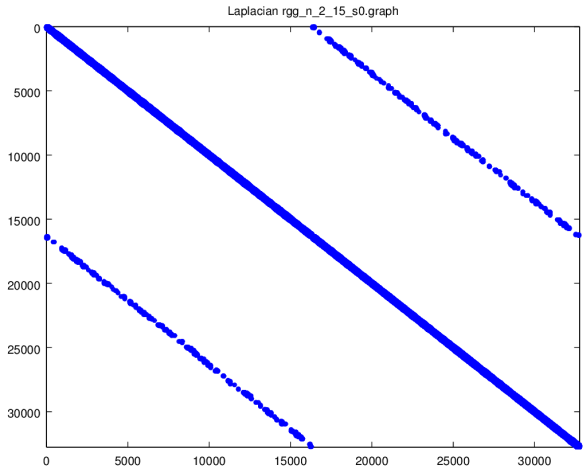

Figure 3.1: Non-zero elements of the Laplacian matrix for `rgg_n_2_15_s0.graph`

t^* through bisection	t^* through SASPA	iterations SASPA
2.0887	2.1283	21
0.5380	0.5662	13
1.1191	1.3674	33
0.6787	0.7236	19

Table 3.1: Numerical results for `rgg_n_2_15_s0.graph` (size of Laplacian is 32768×32768, spectrum is $[0, 25.896]$)

An example of such a perturbation is shown in 3.2, the matrix A is the Laplacian matrix of `rgg_n_2_15_s0.graph`. It can be seen that the nearest intersection of the largest eigencurve is around 0.6. The algorithm computes the intersection to be at 0.56.

As parameters for the algorithm, in all cases is $\epsilon = 10^{-12}$, $k_{\min} = 4$ and $k_{\max} = 20$. The initial $\lambda_{\max}$ and its corresponding eigenvector are computed at the initial stage with the built in `Matlab`method `eigs`.

The results for random diagonal matrices are shown in tables 3.1, 3.2 and 3.3. Each row represents a different test case. The first column is the intersection calculated by using a golden section search method, which is the most accurate value for t^* known, the second column through SASPA with the described rule and the third column the number of iterations necessary for SASPA.

The exact cross-point of the eigencurves was computed by using a golden section search, which is a variant of bisection. Per computation of the intersection point up to four decimals accurate, 19 computations of the largest eigenvalue were necessary. Thus even if computing the largest eigenvalue can be done within a few iterations of Jacobi–Davidson, Rayleigh Quotient Iteration

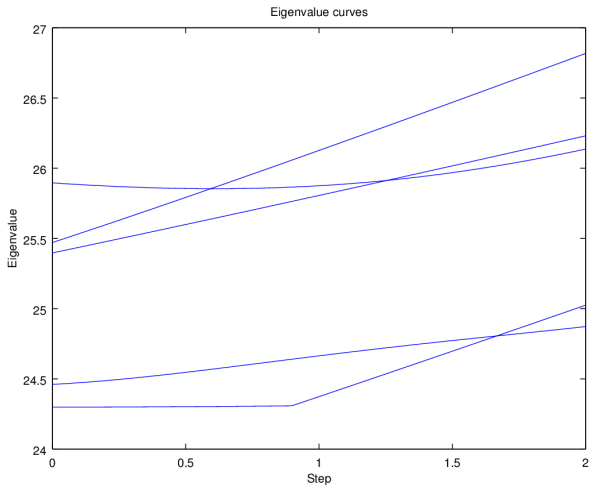

Figure 3.2: The largest five eigenvalue of $A + t\dot{A}$ with A the Laplacian matrix for `rgg_n_2_15_s0.graph` and $\dot{A}$ as described

t^* through bisection	t^* through SASPA	iterations SASPA
0.7537	0.9641	11
1.6514	1.7891	21
1.0576	1.3721	17
2.2711	2.4224	9

Table 3.2: Numerical results for `rgg_n_2_16_s0.graph` (size of Laplacian is 65536×65536, spectrum is $[0, 28.580]$)

t^* through bisection	t^* through SASPA	iterations SASPA
2.1570	2.0325	21
0.6842	1.7685	20
0.1985	0.2199	5
2.2538	3.0589	31

Table 3.3: Numerical results for `rgg_n_2_17_s0.graph` (size of Laplacian is 131072×131072, spectrum is $[0, 29.655]$)

or the power method, it still takes a lot more time to find the intersection point through the bisection method than with SASPA.

Although the reported t^* is less accurate than the t^* determined by the bisection method, SASPA with the proposed selection rule is able to compute the intersection point for different matrix perturbations within a few iterations.

For larger matrices, the error increases. That can be attributed to the fact that second order effects increase in magnitude for the larger instances. That is because the spectrum is more dense. Hence by theorem 2.2.11 it can be expected that the second order derivatives of the eigenvalues are larger for dense spectrum's than for less dense spectrum's.

To get a sense of the computation time difference: By using the `eigs` method from `Matlab`, the instances from `rgg_n_2_15_s0.graph` take 30–35 seconds to compute; by using SASPA with selection rule it takes 24–30 seconds for 50 iterations (that includes initial computation of $\lambda_{\max}$ and the corresponding eigenvector).

For `rgg_n_2_17_s0.graph` the computation time by bisection rose to 125–175 seconds; by SASPA to 140–150 seconds for 50 iterations. The instances in which the bisection method was faster were those with the intersection point close to 0. That is because in those instances the eigencurves diverge fast, hence the computation for the largest eigenvalue takes less time. However, in those instances SASPA converges within a few iterations, thus it is wasteful to run actually 50 iterations.

If for `rgg_n_2_17_s0.graph` the maximum number of iterations is reduced to 25, then computation by SASPA takes 100–115 seconds to converge. And in most cases, the method converges to the nearest intersection. However, it should be noted that the method does not converge always to the nearest within 25 iterations.

Hence SASPA is in most cases much faster than the bisection method. If the required accuracy is not too large then SASPA is a suitable choice for these kind of computations. If accuracy is very important, then the bisection method is better suited.

Note that these computations time have a sense of arbitrariness. They depend on the current load of the computer on which they are run, the underlying method used by `eigs`, the availability of multicore processors, etc. Most of those parameters cannot be controlled on the machine that these methods are tested on. Hence the computation times that are reported should not be taken as absolute values, but only used to form a general sense of expected computation times of both methods relative to each other.

Following those computational results, the proposed method is faster than a bisection method based on `eigs` from `Matlab`, if the maximum number of iterations is bounded correct. Therefore implementations should focus to bound the number of iterations based on the best estimate so far of the nearest intersection.

Chapter 4

Algorithms for eigenvalue optimization

In this chapter it will be described how to solve eigenvalue optimization problems of the following form:

$$\begin{aligned} &\inf_{z \in \mathbb{R}^m} \lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z)) \\ &\text{s. t. } Pz \geq 0 \end{aligned} \tag{4.1}$$

With $C \in \mathcal{S}^n$, $\mathcal{A}^{\mathrm{T}} : \mathbb{R}^m \to \mathcal{S}^n$, $z \mapsto \sum_i z_i A_i$, $P \in \mathbb{R}^{k \times m}$ with normalized rows and $\geq$ component wise.

These problems arise in the study of semidefinite optimization problems as was explained in chapter 2. The emphasis in this chapter will be on sparse instances of eigenvalue problems. Also it will be marked when the distinction between sparse and non-sparse instances becomes important.

The basic idea of solving problem 4.1 in an iterative manner is as follows:

1. Start with a feasible solution $z \in \mathbb{R}^m$, i.e. $Pz \geq 0$, e.g. $z = 0$
2. (a) Calculate a descent direction
   (b) Calculate global optimum in that direction
   (c) Update z
   (d) Repeat until steps are too small for steady improvement

Except for the first step, which is self-evident, each of these steps will be explained with more rigor in the next sections.

On notation: in the next section the formula $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ might be denoted as $\lambda_{\max}(z)$. The reason being that in chapter 2 already many results were obtained for functions such as $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ which are better resembled by writing it as a function $\lambda_{\max}(z)$. The optimum (i.e. the global minimum)

is denoted as $z^* \in \mathbb{R}^m$. Also, let z_i be the optimum point being calculated in the ith iteration, and d_i and t_i the search direction and the step size in the ith iteration. If from the context is clear which iteration is meant, then the subscript is omitted.

As a quick recap on some results derived in chapter 2: The function $\lambda_{\max}(z)$ is convex, and if for $z \in \mathbb{R}^m$, $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is simple, then and only then does the gradient $\nabla\lambda_{\max}(z)$ exists. For all $z \in \mathbb{R}^m$ with $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ multiple is the subgradient defined. Only the subgradient at $z^* \in \mathbb{R}^m$ contains the all zero vector.

4.1 Calculating a descent direction

This section is concerned with finding a feasible descent direction $d \in \mathbb{R}^m$ such that there exists a step size $t > 0$ such that $\lambda_{\max}(z + td) < \lambda_{\max}(z)$ and $P(z + td) \geq 0$.

The first section gives some examples and explains the steepest descent for when $\lambda_{\max}(z)$ is simple. The second section generalizes the first section to include the non-simple case. The third section is concerned with cutting infeasible directions from d.

Although the first section might seem easy it is noteworthy that the examples presented there provide for a better insight for the non-simple case, which is considerable harder. That is because if $\lambda_{\max}(z)$ is no longer simple, than the problem is not smooth and the only tools available are the subgradients introduced in section 2.2.

4.1.1 Descent direction in the simple case

In the case that for a given $z \in \mathbb{R}^m$, $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is simple, then it is easy to compute the steepest descent direction. For the steepest direction only the gradient is necessary which only depends on the largest eigenvalue with corresponding eigenvector. Hence the steepest descent direction is preferable if the instance of problem 4.1 is large and sparse.

If for a given $z \in \mathbb{R}^m$, $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is simple with normalized eigenvector q, then the gradient $\nabla\lambda_{\max}(z)$ is well defined (corollary 2.2.8). Thus an obvious choice for descent direction would be $d := -\nabla\lambda_{\max}(z)^{\mathrm{T}}$:

Definition 4.1.1. *If $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is simple, then the steepest descent direction is:*

$$d := -\nabla\lambda_{\max}(z) = (q^{\mathrm{T}}A_1q, \ldots, q^{\mathrm{T}}A_mq)^{\mathrm{T}} \tag{4.2}$$

The next three examples show how the steepest direction works in practice for unconstrained problems:

Example 4.1.2. *Consider problem 4.1 with $m = 2$ and no inequalities and:*

$$C = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{4.3}$$

Analytically the eigenvalues of $C-\mathcal{A}^{\mathrm{T}}(z)$ *are* $\lambda_1 = 1+\sqrt{(z_1-1)^2+(z_2-1)^2}$ *and* $\lambda_2 = 1-\sqrt{(z_1-1)^2+(z_2-1)^2}$.

Thus the eigenvalues at $z=0$ *are* $\lambda_1 = 1+\sqrt{2}$ *and* $\lambda_2 = 1-\sqrt{2}$*; hence* λ_1 *is simple. The eigenvector* q *corresponding to* λ_1 *is* $q = (1+\sqrt{2},1)^{\mathrm{T}}$*. Then:*

$$\nabla f(0) = -(q^{\mathrm{T}}A_1 q, q^{\mathrm{T}}A_2 q) \approx -(4.83, 4.83) \tag{4.4}$$

Hence $d = -\nabla f(0)^{\mathrm{T}}$ *is a descent direction. And indeed, the optimum is at* $z^* = (1,1)^{\mathrm{T}}$ *at which* $C-\mathcal{A}^{\mathrm{T}}(z)$ *is the identity and* $\lambda_1 = \lambda_2 = 1$.

The next example shows that for simple instances with $m=2$ the steepest direction is not necessarily the best direction:

Example 4.1.3. *Consider problem 4.1 with* $m=2$*, and no inequalities, and:*

$$C = \begin{pmatrix} -2 & 1 & -2 \\ 1 & 2 & 0 \\ -2 & 0 & 2 \end{pmatrix},\ A_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.5}$$

Analytically the eigenvalues are:

$$\begin{aligned} \lambda_1 &= \frac{1}{2}(\sqrt{4(z_2-1)^2+\frac{1}{5}((5z_1+12)^2+16)} - z_1) \\ \lambda_2 &= 2 \\ \lambda_3 &= \frac{1}{2}(-\sqrt{4(z_2-1)^2+\frac{1}{5}((5z_1+12)^2+16)} - z_1) \end{aligned} \tag{4.6}$$

λ_1 *is minimized for* $z^* = (-2,1)^{\mathrm{T}}$*, at which* $\lambda_1 = 2$.

The eigenvalues of C *(that is at* $z=0$*) are* $\lambda_1 = 3$*,* $\lambda_2 = 2$ *and* $\lambda_3 = -3$*. Hence* λ_1 *is simple. The eigenvector* q *corresponding to* λ_1 *is* $q = (-1,-1,2)^{\mathrm{T}}$*. Then:*

$$\nabla f(0) = -(q^{\mathrm{T}}A_1 q, q^{\mathrm{T}}A_2 q) = (3,-2) \tag{4.7}$$

And $d = \frac{-\nabla f(0)^{\mathrm{T}}}{\|\nabla f(0)^{\mathrm{T}}\|_2} \approx (-0.83, 0.55)^{\mathrm{T}}$*. Hence the global minimizer* z^* *is not necessarily of the form* $z_0 + td$ *for* $m \geq 2$.

The next example shows that steepest directions turn a simple largest eigenvalue fast in a non-simple one:

Example 4.1.4. *Consider problem 4.1 with* $m=2$*, no inequalities and:*

$$C = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix},\ A_1 = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 0 \\ 0 & -2 \end{pmatrix} \tag{4.8}$$

Analytically the eigenvalues are:

$$\begin{aligned} \lambda_1 &= 2+z_1 \\ \lambda_2 &= 3-z_1+2z_2 \end{aligned} \tag{4.9}$$

Which is clearly unbounded for $(z_1, z_2) \to (-\infty, -\infty)$.

Hence at $z = 0$ *the largest eigenvalue is* $\lambda_{\max} = 3$ *and corresponding eigenvector is* $q = (0, 1)^{\mathrm{T}}$. *The corresponding gradient is*

$$\nabla f(0) = -(q^{\mathrm{T}} A_1 q, q^{\mathrm{T}} A_2 q) = (-1, 2) \tag{4.10}$$

After one step (assuming an exact line search in direction $d = -\nabla\lambda_{\max}(z)$*)* $z = (\frac{1}{4}, -\frac{1}{2})^{\mathrm{T}}$ *and* $\lambda_1 = \lambda_2$.

Hence the steepest direction is not necessarily the best direction, but in practice for unconstrained problems it works quite decent. It seems that for problems in which the largest eigenvalues stay simple that a good strategy is to use some algorithm which minimizes a quadratic function, such as conjugated gradients, e.g. the Fletcher-Reeves method. However in practice, most instances of eigenvalue optimization in which the starting eigenvalue is simple, have within a few iterations a non-simple largest eigenvalue.

Note that the steepest descent direction d is not necessarily a feasible descent direction because of the constraint $Pz \geq 0$ in equation 4.1. This will be returned on this in subsection 4.1.3.

4.1.2 Descent direction in the non-simple case

Often $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is non-simple. Hence the gradient $\nabla\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ does not exist. However corollary 2.2.24 gives a generalization of the gradient and directional derivative which is applicable for this situation which allows to compute a steepest direction.

Before continuing, The following terms and definitions will be encountered in the rest of this section: Denote the multiplicity of $\lambda_{\max}(z)$ with t and the normalized eigenvectors that span the whole eigenspace as $q_1, \ldots, q_t$. Moreover, let Q be the matrix which spans the eigenspace, that is with $q_1, \ldots, q_t$ as columns.

If $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is non-simple, then the steepest descent direction is:

$$\begin{aligned}
d &= \arg\min_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \nabla_d \lambda_{\max}(z) \\
&= \arg\min_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} -\min_{q\in Q} \langle \mathcal{A}^{\mathrm{T}}(d), qq^{\mathrm{T}} \rangle \\
&= \arg\max_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \min_{q\in Q} \langle \mathcal{A}^{\mathrm{T}}(d), qq^{\mathrm{T}} \rangle \\
&= \arg\max_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \lambda_{\min}(Q^{\mathrm{T}}\, \mathcal{A}^{\mathrm{T}}(d) Q) \\
&= \arg\max_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \min_{\substack{m\in\mathbb{R}^t \\ \|m\|=1}} m^{\mathrm{T}} Q^{\mathrm{T}}\, \mathcal{A}^{\mathrm{T}}(d) Q m \\
&= \arg\max_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \min_{\substack{m\in\mathbb{R}^t \\ \|m\|=1}} \langle Q m m^{\mathrm{T}} Q^{\mathrm{T}}, \mathcal{A}^{\mathrm{T}}(d) \rangle \\
&= \arg\max_{\substack{d\in\mathbb{R}^m \\ \|d\|=1}} \min_{\substack{m\in\mathbb{R}^t \\ \|m\|=1}} \langle \mathcal{A}(Q m m^{\mathrm{T}} Q^{\mathrm{T}}), d \rangle
\end{aligned} \tag{4.11}$$

A linear program will be devised to solve equation 4.11.

This LP is based on the observation that in the last identity $Qmm^{\mathrm{T}}Q$ can be written as $\sum_{i,j} M_{i,j} q_i q_j^{\mathrm{T}}$ with $M = mm^{\mathrm{T}}$. Also because $\|m\| = 1$, each diagonal element of M is non-negative and $\sum_i M_{i,i} = 1$. Hence the last identity in 4.11 can be written as a convex sum over $q_i q_i^{\mathrm{T}}$ plus some cross terms.

In experiments it was confirmed that a good strategy is to minimize with d the elements in the convex sum and constraint d to be such that the cross terms are zero. The LP used:

$$\begin{aligned}
&\min_{\substack{\delta\in\mathbb{R} \\ d\in\mathbb{R}^m}} \delta \\
&\text{s.t. } \delta + d^{\mathrm{T}}\, \mathcal{A}(q_i q_i^{\mathrm{T}}) \geq 0 \qquad (\forall i = 1, \ldots, t) \\
&\quad d^{\mathrm{T}}\, \mathcal{A}(q_i q_j^{\mathrm{T}}) = 0 \qquad (\forall i, j = 1, \ldots, t \quad i \neq j) \\
&\quad -1 \leq d \leq 1
\end{aligned} \tag{4.12}$$

(This LP is similar to the quadratic program in the successive quadratic programming approach explained in [21] and [22] and shortly described in section 4.3. However in this LP there are less constraints on d with respect to other eigenvalues than $\lambda_{\max}$. Also the QP in [22] uses a quadratic objective to stop d in the optimum from being too large, but that is not a requirement in this case because only the direction of d is important.)

If there are active constraints, i.e. $P_a \neq 0$, then linear program 4.12 can be

augmented with the active constraints:

$$\begin{aligned}
&\min_{\substack{\delta\in\mathbb{R}\\ d\in\mathbb{R}^m}} \delta \\
&\text{s. t. } \delta + d^{\mathrm{T}}\,\mathcal{A}(q_i q_i^{\mathrm{T}}) \geq 0 \qquad (\forall i = 1,\ldots,t) \\
&\qquad d^{\mathrm{T}}\,\mathcal{A}(q_i q_j^{\mathrm{T}}) = 0 \qquad (\forall i,j = 1,\ldots,t \quad i \neq j) \\
&\qquad P_a d \geq 0 \\
&\qquad -1 \leq d \leq 1
\end{aligned} \tag{4.13}$$

This linear program has $\frac{t(t+1)}{2}$-constraints in m-variables, hence is actually quite small and suitable to be solved through a simplex solver.

Example 4.1.5. *Consider problem 4.1 with $m = 2$, no inequalities and:*

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \tag{4.14}$$

Analytically the eigenvalues are:

$$\begin{aligned}
\lambda_1 &= 1 - z_1 \\
\lambda_2 &= 1 - 2z_2
\end{aligned} \tag{4.15}$$

Clearly the problem is unbounded in the direction $d = (2,1)^{\mathrm{T}}$.
Solving linear program 4.12 with $q_1 = e_1$ and $q_2 = e_2$ yields:

$$\begin{aligned}
\mathcal{A}(q_1 q_1^{\mathrm{T}}) &= (1,0)^{\mathrm{T}} \\
\mathcal{A}(q_2 q_2^{\mathrm{T}}) &= (0,2)^{\mathrm{T}} \\
d &= (1,\frac{1}{2})^{\mathrm{T}} \\
\delta &= \nabla_d \lambda_{\max}(z) = -1
\end{aligned} \tag{4.16}$$

4.1.3 Cut infeasible descent directions

If the problem is constrained, i.e. $Pz \geq 0$ with P not trivial, then the region in which $\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z))$ is minimized is within a convex polytope. That is because $Pz \geq 0$ implies that each row of P describes a half space, and all those together in $Pz \geq 0$ is the intersection of all those half spaces. To continue, the following definitions give some more grasp of the problem.

Definition 4.1.6. *The set $Pz \geq 0$ is called the feasible set and is denote by $\mathcal{F}$, the interior of $\mathcal{F}$, i.e. $Pz > 0$, is called the strict feasible set and is denoted as* $\mathrm{int}(\mathcal{F})$ *and the boundary of $\mathcal{F}$ is denoted as $\partial\,\mathcal{F}$.*

Definition 4.1.7. *The active constraints on $Pz \geq 0$ is the index set $\mathcal{I}$, such that $p_i^{\mathrm{T}} z = 0$, with p_i the ith row of P as column vector, for all $i \in \mathcal{I}$. The active constraints matrix P_a has as rows exactly the active rows of P. That is $P_a^{\mathrm{T}} = (p_{I_1} \mid \ldots \mid p_{I_l})$.*

If there are no active constraints at $z_i \in \mathcal{F}$, then $z_i \in \operatorname{int}(\mathcal{F})$.

Constraint p_i (or short i) is said to be violated if $p_i^{\mathrm{T}} z < 0$. Hence if there are violated constraints at z, then $z \notin \mathcal{F}$.

Clearly if $z \in \operatorname{int}(\mathcal{F})$, then there is no constraint on search direction d, however there is a constraint on the maximum step size. This point is something that will be discussed further in section 4.2.1.

If for a given $z \in \partial\mathcal{F}$ with active constraints, then those constraints also put a constraint on descent direction d. Specially, if P_a is the active constraints matrix at z, then $P_a d \geq 0$ for any feasible descent direction d.

Hence, for a given descent direction d, vector $\tilde{d} \in \mathbb{R}^m$ is sought, which is a feasible descent direction, which minimizes $\|d - \tilde{d}\|_2$. If all rows of P_a are orthogonal, then that is easy by using some kind of Gram-Schmidt decomposition and discard the violating directions in P. However, the difficulty of this problem is that the rows of P_a are not necessarily orthogonal.

A strategy is to calculate d as described in the previous subsections and then try to find a vector $r \in \mathbb{R}^m$ of small norm such that $\tilde{d} = d + r$ is feasible. This can be solved by the following quadratic program:

$$\begin{aligned} &\min_{r \in \mathbb{R}^m} r^{\mathrm{T}} r \\ &\text{s. t. } P_a(d + r) \geq 0 \end{aligned} \tag{4.17}$$

Quadratic program 4.17 has l constraints in m variables, which is not too large and can be solved fast by a simplex solver. (See for instance [33]; the idea is akin to linear simplex by rotating through vertexes of the polytope described by $P_(d + r) = 0$ while also checking the edges for optimality.)

Algorithm 4.1.8 Algorithm to cut infeasible directions from d

Require:
$z \in \mathbb{R}^m$: current location
$P \in \mathbb{R}^{l \times m}$: constraint matrix, i.e. $Pz \geq 0$
$d \in \mathbb{R}^m$: descent direction
Ensure:
$\tilde{d} \in \mathbb{R}^m$: feasible descent direction

1: **function** CUT INFEASIBLE(z, P, d)
2: $P_a \leftarrow \{p_i \text{ s. t. } (Pz)_i = 0\}$, i.e. the set of active constraints
3: Solve QP 4.17 for $r \in \mathbb{R}^m$
4: $\tilde{d} \leftarrow d + r$
5: **return** $\tilde{d}$
6: **end function**

4.2 Determine max step size

After determining a descent direction $d \in \mathbb{R}^m$ in section 4.1, the next concern is computing a optimal step size $t^* \in \mathbb{R}^+$ that minimizes:

$$f(t) : \mathbb{R}^+ \to \mathbb{R}, t \mapsto \lambda_{max}(z + td) = \lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(z + td)) \tag{4.18}$$

Such that $P(z+t^*d) \geq 0$. That is, the step size should not be too large such that $z + td$ becomes infeasible. Obvious if there are no inequalities, then the problem is unbounded.

This is a non-smooth convex problem, where non-smoothness arises if eigenvalue curves cross. For large sparse problems, f is hard to compute directly because the eigenvalues tend to cluster. That is, the gap between two distinct eigenvalues becomes very small. Hence the power method will converge very slow and neither the Lanczos method works efficiently because of the large multiplicity of the eigenvalues. Moreover, the matrix $C - \mathcal{A}^{\mathrm{T}}(z + td)$ is near the optimum far from diagonal dominant, hence the Davidson method does not work as well.

The first subsection gives a short analysis of the problem and states some notation which will be used further in this section. In the second subsection an algorithm is developed for large problems which cannot be tackled by classic line (secant or Newton method) searches.

4.2.1 Analysis of the problem

Some words on notation: by theorem 2.2.12 it is possible to label all eigencurves of $C - \mathcal{A}^{\mathrm{T}}(z + td)$ at $t = 0$ as:

$$\lambda_1(z) \geq \cdots \geq \lambda_n(z) \tag{4.19}$$

With $t \to \lambda_i(z + td)$ smooth. If $\lambda_{\max}(z)$ has multiplicity τ, then $\lambda_1(z) = \cdots = \lambda_\tau(z) > \lambda_{\tau+1}(z)$. Note that this labeling of the eigencurves does not imply that $\lambda_2(z + td) \leq \lambda_1(z + td)$ for all $t \in \mathbb{R}^+$. In fact minimizing equation 4.18 involves the computation of t^* such that $\lambda_1(z + t^*d) \leq \lambda_{\max}(z + t^*d)$.

Moreover, in order to handle the problem some kind of derivatives are necessary:

Definition 4.2.1. *Observe that the forward derivative of $f(t)$ to t exists and is defined as:*

$$D_t f(t) = \lim_{h \downarrow 0} \frac{\lambda_{\max}(z + td + hd) - \lambda_{\max}(z + td)}{h} \tag{4.20}$$

Similar the backward derivative is defined as:

$$D'_t f(t) = \lim_{h \uparrow 0} \frac{\lambda_{\max}(z + td + hd) - \lambda_{\max}(z + td)}{h} \tag{4.21}$$

From these observations the following observation can be deduced:

Theorem 4.2.2. *Consider the function f from equation 4.18. If for $t' \in \mathbb{R}^+$, function f is non-smooth at t', then $D_t f(t')$ is left discontinuous and at least two eigencurves $\lambda_i(t)$, $\lambda_j(t)$ with $i \neq j$ cross each other at t'.*

Proof. This follows direct from theorem 2.2.12: All eigencurves can be labeled such that they are smooth. Hence if f is non-smooth at t', then $\lambda_{\max}(z + t'd)$ is non-simple and thus at least two distinct eigencurves cross each other. Moreover, there is a neighborhood N of t' such that for all $t \in N$, $t \geq t'$, $D_t f(t)$ follows one eigencurve other than the distinct curve followed $D_t f(t)$ for $t < t'$. Thus for $t \in N \cap (-\infty, t')$, $D_t f(t)$ is continuous and for $t \in N \cap [t', \infty)$, $D_t f(t)$ is continuous, □

Also:

Theorem 4.2.3. *Consider the function f from equation 4.18. Denote the minimizer with t^*, i.e. $f(t^*) \leq f(t)$ for all $t \in \mathbb{R}^+$. Then the following optimality condition holds at t^*, given that $z + t^* d \in \operatorname{int}(\mathcal{F})$:*

$$\begin{aligned} D_t f(t) &\leq 0 \quad (\forall t \in [0, t^*)) \\ D_t f(t) &\geq 0 \quad (\forall t \in [t^*, \infty)) \end{aligned} \tag{4.22}$$

Or equivalent:

$$\begin{aligned} D'_t f(t) &\leq 0 \quad (\forall t \in [0, t^*]) \\ D'_t f(t) &\geq 0 \quad (\forall t \in (t^*, \infty)) \end{aligned} \tag{4.23}$$

Proof. All these conditions follow from noting that f is convex and the identity:

$$f(t) = \int_0^t D_t f(\tau) \, \mathbf{d}\, \tau + f(0) \tag{4.24}$$

□

By theorem 2.2.17 it follows that:

$$D_t f(t) = \max_{\substack{u \in \mathbb{R}^m \\ (C - \mathcal{A}^{\mathrm{T}}(z+td))u = \lambda_{\max}(z+td)u \\ \|u\|_2 = 1}} -u^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) u \tag{4.25}$$

And similar:

$$D'_t f(t) = - \max_{\substack{u \in \mathbb{R}^m \\ (C - \mathcal{A}^{\mathrm{T}}(z+td))u = \lambda_{\max}(z+td)u \\ \|u\|_2 = 1}} u^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) u \tag{4.26}$$

Hence the computation of $D_t f(t)$ can be seen as an eigenvalue problem: Let $q_1(t), \ldots, q_{l(t)}(t)$ be a basis for the eigenspace to the largest eigenvalue, then any $u \in \mathbb{R}^m$ can be expressed as:

$$u = Q(t)z \quad \text{with} \quad Q(t) = (q_1(t), \ldots, q_{l(t)}(t)) \quad \text{and} \quad \|z\| = 1 \tag{4.27}$$

Equation 4.25 is then equivalent to:

$$D_t f(t) = -\lambda_{\min}(Q(t)^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) Q(t)) \tag{4.28}$$

Similar the backward derivative:

$$D'_t f(t) = -\lambda_{\max}(Q(t)^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) Q(t)) \tag{4.29}$$

And checking the optimality conditions 4.2.3:

$$\begin{aligned} D_t f(t^*) &= -\lambda_{\min}(Q(t^*)^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) Q(t^*)) \geq 0 \\ D'_t f(t^*) &= -\lambda_{\max}(Q(t^*)^{\mathrm{T}} \mathcal{A}^{\mathrm{T}}(d) Q(t^*)) \geq 0 \end{aligned} \tag{4.30}$$

The bounded case

If there exists a $\hat{t} > 0$ such that $z + td \notin \mathcal{F}$ ($P(z + td) \ngeq 0$) for all $t > \hat{t}$, then the problem is bounded. Hence optimality conditions 4.30 do not hold necessary in the optimum.

It can be that $D_t f(\hat{t}) < 0$, thus that in the unbounded case $t^* > \hat{t}$, but that would imply $z + t^* d \notin \mathcal{F}$. Therefore t^* should be bounded by $\hat{t}$.

To compute the minimal $\hat{t}$ (further denoted as $t_{\max}$: Let $\mathcal{I}$ be the index set such that $P_{\mathcal{I}} d < 0$, that is the row index set of all constraints which will be violated if $t \to \infty$. Then $t_{\max}$ is:

$$t_{\max} = \min_{i \in \mathcal{I}} -\frac{P_i z}{P_i d} \tag{4.31}$$

4.2.2 Line search methods based on intersections

If all eigencurves are assumed to be linear then the minimum for f is obtained for certain at an intersection between two eigencurves. Hence a good start for minimizing f is by using the algorithm developed in section 3.4 to find the nearest intersection point for the largest eigencurve.

After a nearby intersection of two eigencurves is found at $\hat{t}$, the following step is to check optimality conditions from theorem 4.2.3. That can be done through checking the intersecting curve, if it has a positive slope then because of smoothness (theorem 2.2.12) it follows that $D_t f(t) \geq 0$ for $t \geq \hat{t}$.

Therefore the main idea is to iterative compute the nearest intersection $\hat{t}$, if the slope of the crossing eigencurve is positive, then stop, else assume that the crossing eigencurve is the new maximum and search for the next intersection.

The algorithm based on section 3.4 to compute the nearest intersection:

Algorithm 4.2.4 SASPA to compute nearest intersection

Require:
$A \in \mathcal{S}^n$: matrix
$\dot{A} \in \mathcal{S}^n$: perturbation
$\lambda \in \mathbb{R}$: largest eigenvalue at $t = 0$
$q_0 \in \mathbb{R}^n$: corresponding eigenvector
$\epsilon_{RQI} \in \mathbb{R}_{>0}$: threshold from where to start Rayleigh Quotient Iteration
$\epsilon_{tol} \in \mathbb{R}_{>0}$: tolerance
$\epsilon_{stop} \in \mathbb{R}$: additional stop criteria
$k_{\min}, k_{\max}$ minimum and maximum size of search space for restart
$iter_{\max}$, maximum number of iterations

Ensure:
$\hat{t} \in \mathbb{R}_+$: intersection point
$\hat{\lambda} \in \mathbb{R}$: nearest crossing eigenvalues at $t = 0$
$\hat{q} \in \mathbb{R}^n$: corresponding eigenvector

$iter \leftarrow 1$, $num \leftarrow 0$, $s \leftarrow \lambda$, $x = (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$, $\hat{t} \leftarrow \infty$
while $iter \leq iter_{\max}$ **do**
▷ Extend search space
  $b \leftarrow \dot{A}x$
  $b \leftarrow \mathrm{GS}([q_0, \ldots, q_{num}], b)$ ▷ Deflate b with $q_0, \ldots, q_{num}$
  Solve $(A - sI)x = b$ for $x \in \mathbb{R}^n$
  $x \leftarrow \mathrm{GS}(X, x)$, $X \leftarrow [X, x]$
▷ Select most promising
  $S \leftarrow X^{\mathrm{T}}AX$
  Compute all eigenpairs (s_i, x_i) of S
  Compute for all pairs corresponding cross points: $t_i \leftarrow \frac{\lambda - s_i}{x_i^{\mathrm{T}} X^{\mathrm{T}} \dot{A} X x_i - q_0^{\mathrm{T}} \dot{A} q_0}$
  Reorder all triplets (t_i, s_i, x_i) such that $t_i \geq t_{num}$ for all $i \leq j$
  $(t, s, x) \leftarrow (t_1, s_1, Xx_1)$
  $x \leftarrow \frac{x}{\|x\|}$
▷ Elective Rayleigh Quotient Iteration
  if $\|Ax - sx\| \leq \epsilon_{RQI}$ **then**
    while $\|Ax - sx\| \geq \epsilon_{tol}$ **do**
      Solve $(A - sI)\dot{x} = x$ for $\dot{x} \in \mathbb{R}^n$
      $x \leftarrow \frac{\dot{x}}{\|\dot{x}\|}$
      $s \leftarrow x^{\mathrm{T}}Ax$
    end while
  end if
▷ Check convergence and deflate
  if $\|Ax - sx\| \leq \epsilon_{tol}$ **then**
    $num \leftarrow num + 1$
    $q_{num} \leftarrow x$
    Deflate X with q_{num}
    if $t \leq \hat{t}$ **then**

25: $\hat{\lambda} \leftarrow s$, $\hat{q} \leftarrow q_{num}$, $\hat{t} \leftarrow t$
26: **end if**
27: **if** $t \leq \epsilon_{stop}$ **then**
28: **return** $(\hat{t}, \hat{\lambda}, \hat{q})$
29: **end if**
30: $(t, s, x) \leftarrow (t_2, s_2, Xx_2)$
31: $x \leftarrow \frac{x}{\|x\|}$
32: **end if**
▷ Check size search space and implicit restart
33: **if** #columns of X is more than $k_{\max}$ **then**
34: $X \leftarrow [Xx_1, Xx_2, \ldots, Xx_{k_{\min}}]$
35: Orthogonalize and normalize X
36: **end if**
37: $iter \leftarrow iter + 1$
38: **end while**

Note that it is very similar to algorithm 3.5.2; the only difference is the additional stopping criteria based on the discussion of the numerical results in section 3.4.

Moreover, if the maximum eigencurve is a parabola, which can happen near the optimum, then there is not a nearby intersection. Hence at the nearest intersection the optimality conditions 4.2.3 are not full filed.

Note that the parabola which interpolates $\lambda_{\max}(t)$ is convex because the Hessian (2.2.11) of the largest eigenvalue is almost everywhere equal or greater than 0. Therefore a viable tactic is to compute the optimum through a parabola fit.

Denote the nearest intersection point by $\hat{t} \in \mathbb{R}$. The parabola interpolation for $\lambda_{\max}(0)$, $\lambda_{\max}(\hat{t})$ and forward derivative $f(0)$ at $t = 0$ is given by:

$$\lambda_{\max}(t) \approx t^2\left(\frac{\lambda_{\max}(\hat{t}) - \lambda_{\max}(0)}{\hat{t}^2} - \frac{f(0)}{\hat{t}}\right) + tf(0) + \lambda_{\max}(0) \tag{4.32}$$

If $\frac{\lambda_{\max}(\hat{t}) - \lambda_{\max}(0)}{\hat{t}^2} - \frac{f(0)}{\hat{t}} > 0$, then the minimum of that parabola is attained at:

$$t^* = \frac{-f(0)}{2\left(\frac{\lambda_{\max}(\hat{t}) - \lambda_{\max}(0)}{\hat{t}} - f(0)\right)} \tag{4.33}$$

If $\frac{\lambda_{\max}(\hat{t}) - \lambda_{\max}(0)}{\hat{t}^2} - \frac{f(0)}{\hat{t}} < 0$, then the parabola has a maximum, however this is not something bound to happen because the parabola tends to be convex and not concave. If this happens then the interpolating interval $[0, \hat{t}]$ is very small and the interpolation parabola is a bad fit. In that case a better method would be a line search through bisection.

The full algorithm to determine the step size for unbounded problems:

Algorithm 4.2.5 Method to compute optimal step size

Require:
$A = C - \mathcal{A}^{\mathrm{T}}(z)$: current location, matrix form
$\mathcal{A}^{\mathrm{T}}(d) \in \mathcal{S}^n$: search direction, matrix form
$f(0) \in \mathbb{R}$: forward derivative at $t = 0$
$t_{\max} \in \mathbb{R}$: see equation 4.31
Ensure:
$t^* \in \mathbb{R}$: minimizer of $\lambda_{\max}(A - t\,\mathcal{A}^{\mathrm{T}}(d))$
$Q(t^*) \in \mathbb{R}^{n \times l}$: orthogonal normalized basis of the eigenspace to $\lambda_{\max}$

1: Compute nearest intersection (t, s, q) with algorithm 4.2.4
2: $t \leftarrow \min(t, t_{\max})$
3: Compute orthonormal eigenbasis $Q(t)$ for $A - t\,\mathcal{A}^{\mathrm{T}}(d)$ corresponding to $\lambda_{\max}(t)$
4: Compute optimality conditions 4.30:
$f \leftarrow -\lambda_{\min}(Q(t)^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}}(d)Q(t))$
$b \leftarrow \lambda_{\max}(Q(t)^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}}(d)Q(t))$
5: **while** $f > 0$ and $b < 0$ **do** ▷ Proceed by interpolation
6: $t' \leftarrow \frac{-f(0)}{2(\frac{\lambda_{\max}(t) - \lambda_{\max}(0)}{t} - f(0))}$
7: Compute orthonormal eigenbasis $Q(t')$ for $A - t'\,\mathcal{A}^{\mathrm{T}}(d)$ corresponding to $\lambda_{\max}(t')$
8: Compute optimality conditions 4.30:
$f \leftarrow -\lambda_{\min}(Q(t')^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}}(d)Q(t'))$
$b \leftarrow \lambda_{\max}(Q(t')^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}}(d)Q(t'))$
9: $t \leftarrow t'$
10: **end while**
11: **if** $t < t_{\max}$ and $f < 0$ **then** ▷ Further improvement is possible
12: $A' \leftarrow A - t\,\mathcal{A}^{\mathrm{T}}(d)$
13: Repeat algorithm 4.2.5 with: $A \leftarrow A'$ and $f(0) \leftarrow f$; returns t^* and $Q(t^*)$
14: $t \leftarrow t + t^*$
15: $Q(t) \leftarrow Q(t^*)$
16: **end if**
17: $t^* \leftarrow t$
18: $Q(t^*) \leftarrow Q(t)$

A practical note on implementation: Often a basis for the eigenspace corresponding to the largest eigenvalue has to be computed. Luckily the matrix is permuted in such a way that it is known which eigencurves meet after the permutation. Hence it is expected that the eigenvectors which compromise the basis, are somewhat near the eigenvectors of of the eigencurves at the beginning. Therefore the basis can be constructed by updating the old eigenvectors to the new matrix.

There are a number of iterative methods available to compute a basis includ-

ing: QR Method, Lanczos, Subspace Accelerated Rayleigh Quotient Iteration (SARQI) and Jacobi–Davidson [30].

The first method works well for small matrices and the second method for which no prior information on the eigenvectors are known. However, as described we have already quite a good idea on how the basis should look like. Therefore the choice is limited to the last three methods.

During simulation were the best results obtained by using Jacobi–Davidson in combination with `BiCGstab`.

4.3 Combining everything

The full method to solve eigenvalue equation 4.1 based on the previous sections is iterative: Compute steepest descent direction $d \in \mathbb{R}^m$, compute step size $t^* \in \mathbb{R}$, update $z \leftarrow z + t^* d$ and repeat.

The full algorithm written out:

Algorithm 4.3.1 Algorithm to solve eigenvalue problem 4.1

Require:
$C \in \mathcal{S}^n$: initial matrix
$A_1, \ldots, A_m \in \mathcal{S}^n$: matrix component of $\mathcal{A}^{\mathrm{T}}$
$P \in \mathbb{R}^{l \times m}$: constraints
Ensure:
$z \in \mathbb{R}^m$: argument that minimizes equation 4.1
$A \in \mathcal{S}^n$: the matrix $C - \mathcal{A}^{\mathrm{T}}(z)$

1: $z \leftarrow 0 \in \mathbb{R}^m$, $A \leftarrow C$
2: Compute a basis Q for the eigenspace corresponding to $\lambda_{\max}(C)$
3: **while** not converged **do**
4: **if** $\operatorname{rank}(Q) = 1$ **then** ▷ $\lambda_{\max}$ is simple
5: Compute $d \in \mathbb{R}^m$ as $-\nabla\lambda_{\max}(z) = \mathcal{A}(QQ^{\mathrm{T}})$
6: **else**
7: Compute $\delta \in \mathbb{R}$ and $d \in \mathbb{R}^m$ from LP 4.12
8: **end if**
9: $d \leftarrow$ CUT INFEASIBLE(z, P, d) ▷ See algorithm 4.1.8
10: $f \leftarrow -\lambda_{\min}(Q^{\mathrm{T}}\,\mathcal{A}^{\mathrm{T}}(d)Q)$
11: **if** $f \geq 0$ **then** ▷ Check convergence
12: **return** (z, A)
13: **end if**
14: Compute $t_{\max}$ by equation 4.31
15: Compute t^* and Q by algorithm 4.2.5
16: $z \leftarrow z + t^* d$
17: $A \leftarrow A - t^*\,\mathcal{A}^{\mathrm{T}}(d)$
18: **end while**

An implementation is included in appendix A.3.

On implementation: When the first basis Q that has to be computed, no prior information is yet known. Thus a suitable iterative method is Lanczos Method. During simulations the best results were obtained by combining Lanczos Method with Jacobi–Davidson.

When the algorithm is in the main loop the eigenspace is updated each time during algorithm 4.2.5.

4.3.1 Comparison to other methods

Algorithm 4.3.1 is not the only known method to solve eigenvalue problems. Not all known algorithms will be discussed in full detail; but will give a brief overview of other known algorithms and highlight the main differences.

Successive Quadratic Programming Algorithm

Overton's method, also known as Successive Quadratic Programming Algorithm, developed around 1988, is an iterative method to solve eigenvalue problems of the form 4.1 [21] [22].

It works by solving each iteration a QP similar to 4.12, to compute a step direction d and step t in one go. Denote y_i as the argument in the ith iteration. Then to compute y_{i+1}: let λ_1 the largest eigenvalue of $C - \mathcal{A}^{\mathrm{T}}(y_i)$, $q_1, \ldots, q_k$ be eigenvectors corresponding to the largest k eigenvalues, and $q_{k+1}, \ldots q_m$ the other eigenvectors, $W \in \mathcal{S}^n_{\succeq 0}$:

$$\begin{aligned}
&\min_{\substack{\delta\in\mathbb{R}\\ d\in\mathbb{R}^m}} \delta + d^{\mathrm{T}} W d \\
&\text{s.t. } \delta + d^{\mathrm{T}}\,\mathcal{A}(q_i q_i^{\mathrm{T}}) = \lambda_i - \lambda_1 \qquad (\forall i = 1, \ldots, k) \\
&\qquad \delta + d^{\mathrm{T}}\,\mathcal{A}(q_i q_j^{\mathrm{T}}) = 0 \qquad (\forall i, j = 1, \ldots, k,\ \ i \neq j) \\
&\qquad \delta + d^{\mathrm{T}}\,\mathcal{A}(q_i q_i^{\mathrm{T}}) \geq \lambda_i - \lambda_1 \qquad (\forall i = k+1, \ldots, m) \\
&\qquad -\rho \leq d \leq \rho
\end{aligned} \tag{4.34}$$

Then $y_{i+1} = y_i + d$.

Note the similarity with the earlier devised LP: If taken k to be the multiplicity of λ_1, then the first two equality's are similar to the LP 4.12.

The argument for taking k larger is that near the optimum those eigencurves most likely coalesce. Hence it seems a good idea to let the linearized eigencurves join when taking step d.

The third inequality is to limit the size of d to be such that the linearized largest eigencurve will not cross any of the other eigencurves. Also the fourth inequality is to limit d from growing too large.

Moreover, in the objective the positive semidefinite matrix W also prevents d from growing too large. If W is chosen to be the Hessian matrix, then the method can be shown to be a Newton method.

Overton's method has been used for many different applications and is seen as a robust method. However one of its drawbacks is the necessity of a full

eigendecomposition of $C - \mathcal{A}^{\mathrm{T}}(y_1)$ which limits the method to small problems. Furthermore, solving a QP instead of a LP is considerable harder, and when y_i is far from optimum, then it can be argued that those extra computation cycles are unnecessary.

Spectral Bundle Method

The Spectral Bundle Method has been developed by Helmberg and Randl in 1997 [14] and is also a dual solver algorithm for semidefinite optimization problems. That is, during each iteration a full feasible dual solution is used as starting point fom which a descent direction is computed, similar as algorithm 4.3.1.

Starting from a semidefinite optimization problem, first they assume that the rank of X is constant such that the additional constraint:

$$\langle I, X \rangle = a \tag{4.35}$$

can be added. This implies that the dual is extended:

$$\begin{aligned} \min_{y \in \mathbb{R}^m, \lambda \in \mathbb{R}} & a\lambda + b^{\mathrm{T}} y \\ \text{s.t. } & C - \mathcal{A}^{\mathrm{T}}(y) - \lambda I \preceq 0 \end{aligned} \tag{4.36}$$

Which has for optimum (λ^*, z^*), $\lambda^* = \lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(y))$. Hence the dual problem can be written as:

$$\min_{y \in \mathbb{R}^m} a\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(y)) + b^{\mathrm{T}} y \tag{4.37}$$

Note that this dual is a mixed eigenvalue optimization and linear optimization problem.

Also note that the subgradient of:

$$f(y) := a\lambda_{\max}(C - \mathcal{A}^{\mathrm{T}}(y)) + b^{\mathrm{T}} y \tag{4.38}$$

can be shown to be:

$$\partial f(y) = \{b - \mathcal{A}(W) \text{ s.t. } \langle W, C - \mathcal{A}^{\mathrm{T}}(y) \rangle = \lambda_{\max}(C - A^{\mathrm{T}}(y)),\ \mathrm{Tr}(W) = 1, W \succeq 0\} \tag{4.39}$$

Next they proceed solving this optimization problem by projecting the eigenvalue optimization problem on some subspace, which consists of vectors computed in earlier iterations such as eigenvectors to the then largest eigenvalue. Then on that projected space they try to minimize $f(y + d) + \frac{u}{2}\langle d, d \rangle$, which is similar to minimizing $f(y + d)$ with the constraint that $\|d\|_2$ becomes not too large. The parameter u is a check on how large $\|d\|$ may grow. The reason that u should not be too small and let $\|d\|$ become too large is because then the difference between f and the projection of f becomes too large.

For a full discussion, see [14] or [4].

The main ingredients of the Spectral Bundle method are the proximity idea, the projection of f onto a subspace which is local closely related to the origional, and the use of subgradients to minimize f local.

The similarities with 4.3.1 are that it works by using subgradients and trying to follow the largest eigenvalues. It also works by remembering information from earlier iterations on the eigenbasis, but instead of algorithm 4.3.1 it works by keeping the change local. Algorithm 4.3.1 has no such limit and uses multiple Jacobi-Davidson steps to correct its eigenbasis.

Chapter 5

Numerical results

In this chapter some numerical results of algorithm 4.3.1 are presented.

The test cases are those described in section 1.4 and section 2.1.2.

5.1 Max-cut

This section follows section 1.4.1.

In section 2.1.2 was the max-cut eigenvalue version presented as:

$$\inf_{z\in\mathbb{R}^{|V|-1}} |V|\lambda_{\max}(\frac{1}{4}L_G - \sum_i z_i \operatorname{diag}(q_i)) \tag{5.1}$$

The size of the instance only depends on the number of edges, hence this problem is well suited for benchmarking. In this section some convergence results for the graphs `toruspm3--8--50` and `toruspm3--15--50` are presented.

The graphs `toruspm3--8--50` and `toruspm3--15--50` were taken from the seventh DIMACS Implementation Challenge [24]. They arise from the study of spin glass model through the Ising model.

In the Ising model there are n particles with each spin ± 1. Particles will align their spin in such a way to minimize the total energy. The total energy depends on, with $s_i = \pm 1$ the spin of particle i and $w_{i,j}$ the interaction between particle i and j:

$$H = \sum_{\substack{i,j \\ i\neq j}}^{n} w_{i,j} s_i s_j \tag{5.2}$$

In typical simulations all particles are placed on a grid, and for any two disconnected particles $w_{i,j} = 0$, and for the connected particles $w_{i,j}$ is taken to be random according to a given distribution. For instance, $w_{i,j} = \pm 1$ with 50% probability.

Table 5.1: Results for solving max-cut instances

Name:	Max-cut:	Iterations to 10%:	To 1%:	To 0.1%:
`toruspm3--8--50`	527.8	19	37	184
`toruspm3--15--50`	3474.4	62	–	–

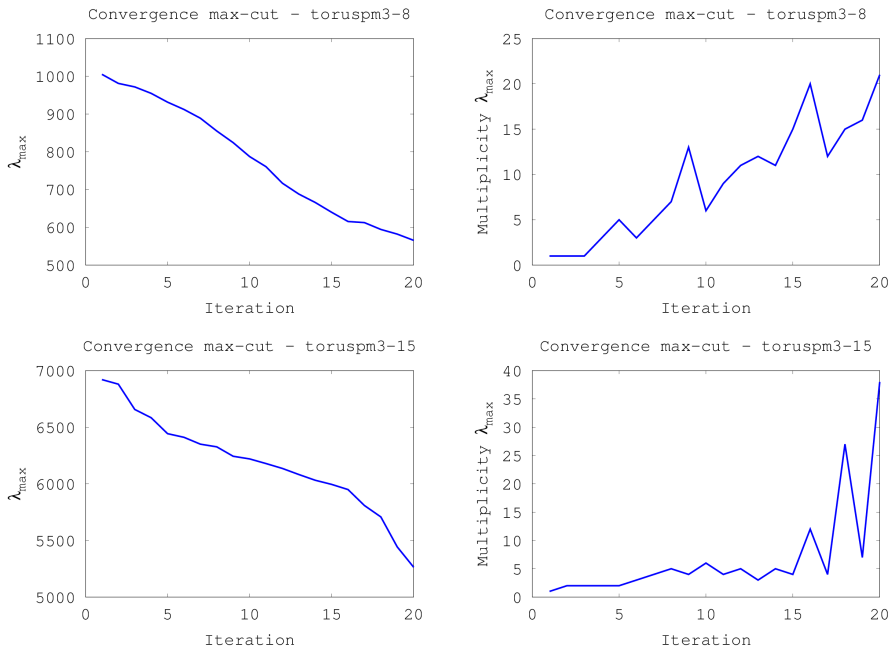

Figure 5.1: Convergence results on maxcut instances

The maximum energy of the particles on such a grid is a maximal cut, hence these kind of models can be studied computational by solving the max-cut problem. For more information, see [9].

The graphs `toruspm3--8--50` and `toruspm3--15--50` are from such models. They represent a grid on a three dimensional torus of size 8 and 15 (that is, `toruspm3--8--50` has $|V| = 8^3 = 512$ vertices and `toruspm3--15--50` has $|V| = 15^3 = 3375$ vertices) with edge weight $w_{i,j} = \pm 1$ with 50% probability.

The results of solving these max-cut instances with algorithm 4.3.1 are presented in table 5.1:

In figure 5.1 convergence of the algorithm is shown. It can be seen that the initial convergence is fast and that the rank of the largest eigenvalue is increasing as the algorithm gets closer to the optimum.

These simulations show that algorithm 4.3.1 has initial fast convergence and is able to solve large eigenvalue optimization problems.

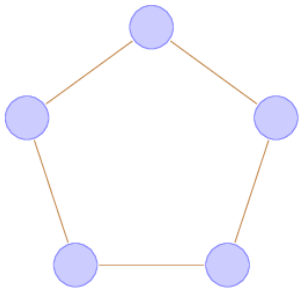

Figure 5.2: A cycle graph of order 5

5.2 Lovász theta number

This section follows section 1.4.2.

The theta number for a graph $G = (V, E)$ can be calculated through the following eigenvalue optimization problem:

$$\min_{y \in \mathbb{R}^{|E|}} \lambda_{\max}(J - \mathcal{E}^{\mathrm{T}}(y)) \tag{5.3}$$

In which $\mathcal{E}^{\mathrm{T}}(y) = \sum_{(i,j)\in E} y_{i,j} E_{i,j}$ with $E_{i,j} = e_i e_j^{\mathrm{T}} + e_j e_i^{\mathrm{T}}$. Note that this is equation is the same as 1.63.

The theta number for a few different types of graphs is known, hence those graphs are perfect suited to test algorithm 4.3.1 on. In this section some numerical results on cycle graphs and Kneser $(2, k)$ graphs, for which theta is known.

Those two set of graphs are interesting not because the theta number is hard to compute for those, it is not, but because they highlight two features of the method: Graphs with n nodes are represented by large matrices of order n, hence it is expected that for large graphs the computation of the eigenbasis forms a bottleneck. Graphs with m edges result in a semidefinite problem with m variables, which implies that the linear program in the method for computing a descent direction has m variables. Hence it is expected that for graphs with many edges solving the linear program forms a bottleneck. The set of cycle graphs contains graphs which have as many vertices as edges, it is expected that computing the eigenbasis will form the most time consuming step for large values of n. On the other hand the set of Kneser $(2, k)$ graphs contains graphs for which the number of edges is exponential larger than the number of vertices, therefore it is expected that for those graphs solving the linear program will be the most time consuming.

Table 5.2: Results for computation of $\theta(C_n)$

Order:	Vertices:	Edges:	θ:	Iterarions:	Multiplicity of $\lambda_{\max}$:	Time (s):
10	10	10	5.0000	2	2	0.04
100	100	100	50.000	2	2	0.06
101	101	101	50.488	2	3	0.04
500	500	500	250.00	2	2	0.86
501	501	501	250.50	2	3	0.62
1000	1000	1000	500.00	2	2	6.52
1001	1001	1001	500.50	2	3	5.17
5000	5000	5000	2500.0	2	5	1122
5001	5001	5001	2500.5	2	4	1035

Cycle graphs

A cycle graph of order n has n vertices and $n+1$ edges such that each vertex is connected with two neighbours and the graph forms a cycle. For an example, see figure 5.2 which depicts a cycle graph of order 5.

The theta number for cycle graphs of order n can be shown to be [17]:

$$\theta(C_n) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{n\cos(\pi/n)}{1+\cos(\pi/n)} & \text{else} \end{cases} \tag{5.4}$$

Results of computation of the theta number for cycle graphs of varying order are presented in table 5.2:

Note that the computation time presented in table 5.2 are not a good representation on the quality of the algorithm because the computer on which these instances were run was also idleing with other tasks. The column is only shown to give a qualitative idea on how well the algorithm scales and typical run times on an ordinary laptop.

The computation time for the large instances surge drastic. That is because during the algorithm an eigenbasis to the largest eigenvalue has to be computed. For the smaller instances that is done by using Lanczos method. For the larger instances that is no longer feasible hence the basis is computed by using an implementation of Jacobi–Davidson based on [30].

Kneser graphs

A Kneser graph (n, k) is such that its vertices correspond to k-element subsets of a set of n elements. Two vertices are connected if the their subsets are disjoint. For an example, see figure 5.3.

The theta number for Kneser graph (n, k) is:

$$\theta(K_{n,k}) = \binom{n-1}{k-1} \tag{5.5}$$

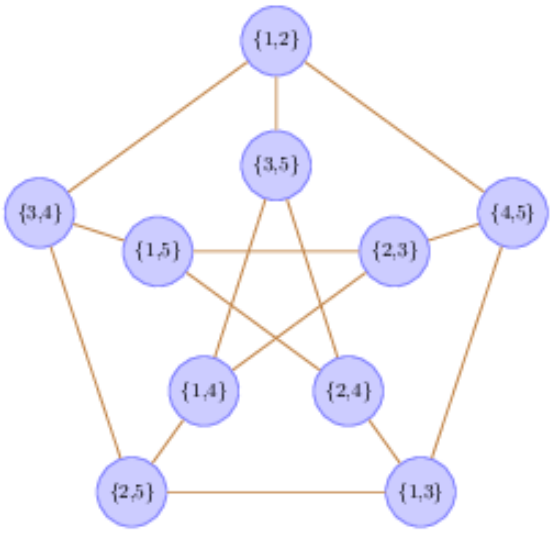

Figure 5.3: Kneser graph $(5, 2)$, also known as the Petersen graph

Table 5.3: Results for computation of $\theta(K_{2,k})$

k:	Vertices:	Edges:	θ:	Iterarions:	Multiplicity of $\lambda_{\max}$:	Time (s):
5	10	15	4	2	5	0.46
10	45	630	9	3	10	0.29
15	105	4095	14	2	15	20.59
20	190	14535	19	2	20	567.15

Some results of computation of the theta number for different Kneser graphs $(2, k)$ are presented in table 5.3:

Note that as was the case with table 5.2, the column showing the computation time is not very accurate. It only serves to indicate that the computation time increases exponential in k.

The computation time rises exponential in k because the linear program for computing a descent direction in the algorithm grows in size linear to the number of edges, which increase exponential. Hence most of the computation time is spend on solving the linear program.

Moreover, note that for the Kneser graph $(2, 10)$ one additional iteration was necessary. That is due to small numerical errors made in the first iteration which are being corrected in the second iteration.

5.3 Ideal GMRES and Arnoldi polynomes

This section follows section 1.4.3.

As described in section 2.1.2, the ideal GMRES polynome for matrix A can be computed by solving the eigenvalue optimization problem:

$$\inf_{z \in \mathbb{R}^k} \lambda_{\max}(\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} + \sum_{1 \leq i \leq k} z_i \begin{pmatrix} 0 & A^i \\ (A^i)^{\mathrm{T}} & 0 \end{pmatrix}) \tag{5.6}$$

In which k is the degree of the polynome.

Similar the ideal Arnoldi polynome for matrix A can be computed through:

$$\inf_{z \in \mathbb{R}^k} \lambda_{\max}(\begin{pmatrix} 0 & A^k \\ (A^k)^{\mathrm{T}} & 0 \end{pmatrix} + \sum_{1 \leq i \leq k} z_i \begin{pmatrix} 0 & A^{i-1} \\ (A^{i-1})^{\mathrm{T}} & 0 \end{pmatrix}) \tag{5.7}$$

For eigenvalue problems solvers form instances for which the ideal Arnoldi polynome has to be computed good benchmarks. That is because it is quite easy to vary in size of the problem.

In this section a few results on calculating these polynomes are presented. The first set of matrices is taken from [32] and [4]: Grcar, Ellipse and Lemniscate 1, all are 48×48. In [32] and [4] the ideal Arnoldi polynome of degree 8 is computed (note, in those articles the ideal Arnoldi polynome is called the Chebyshev polynome of the matrix).

Grcar:

$$\begin{pmatrix} -1 & 1 & 1 & 1 & 1 & 0 & & & \\ 0 & -1 & 1 & 1 & 1 & 1 & \ddots & 0 & \\ & 0 & -1 & 1 & 1 & 1 & \ddots & 1 & 0 \\ & & 0 & -1 & 1 & 1 & \ddots & 1 & 1 \\ & & & 0 & -1 & 1 & \ddots & 1 & 1 \\ & & & & 0 & -1 & \ddots & 1 & 1 \\ & & & & & 0 & \ddots & -1 & 1 \\ & & & & & & \ddots & 0 & -1 \end{pmatrix} \tag{5.8}$$

Ellipse:

$$\begin{pmatrix} 0 & 3 & 0 & & & & \\ 2 & 0 & 3 & 0 & & & \\ 0 & 2 & 0 & 3 & \ddots & & \\ & 0 & 2 & 0 & \ddots & 0 & \\ & & 0 & 2 & \ddots & 3 & 0 \\ & & & 0 & \ddots & 0 & 3 \\ & & & & \ddots & 2 & 0 \end{pmatrix} \tag{5.9}$$

Lemniscate 1:

$$\begin{pmatrix} 1 & 1 & 0 & & & & \\ 0 & -1 & 1 & 0 & & & \\ & 0 & 1 & 1 & \ddots & 0 & \\ & & 0 & -1 & \ddots & 1 & 0 \\ & & & 0 & \ddots & 1 & 1 \\ & & & & \ddots & 0 & -1 \end{pmatrix} \tag{5.10}$$

Numerical results on convergence on computation of the ideal Arnoldi polynome are presented in table 5.4 and figure 5.4.

As can be observed for all the three matrices is the initial convergence exponential and the algorithm reaches the ten procent mark within a few iterations. However, the computation of the last few digits takes numerous iterations. It was during simulations observed that for these instances, with the exception to Lemniscate1, near the optimum, the chosen descent directions were zig-zagging hence convergence was slow. Hence the method could be improved by taking some kind of conjugated gradients near the optimum.

Also note that for these computations the multiplicity of $\lambda_{\max}$ was highly volatile, typical being larger than 2 and smaller than 20. Almost nowhere, except for the initial starting point, was $\lambda_{\max}$ simple.

Moreover, although the number of iterations in table 5.4 may seem large, the computations were actually quite fast. It took little less than two hours to compute 1500 iterations for Grcar on a single core. To compute 250 iterations for Ellipse took roughly 16 minutes, and to compute 15 iterations for Lemniscate took little less than 2 minutes. Hence the computations were completed in a small time interval.

Although this method was not timed against efficient implementations of Overton's Succesive Quadratic Programming method or Helmberg and Randl's Spectral Bundle method, the time measured is expected to be near the other methods. In some cases, for example with high multiplicity of the largest eigenvalue, it is expected to be faster than the Spectral Bundle method and in cases

Table 5.4: Results for computation of ideal Arnoldi polynome of degree 8

Name:	$\|q(A)\|_2$	Iterations to 10%:	To 1%:	To 0.1%:	To 0.01%:	To 0.001%:
Grcar	1766.31	16	52	305	809	1409
Ellipse	7710.27	22	40	100	177	243
Lemniscate 1	1	7	9	10	11	14

when only small steps are taken (such as near the optimum), it is expected that the Spectral Bundle method has faster convergence. Moreover, for large and sparse problems both algorithm 4.3.1 and the Spectral Bundle method are expected to be faster than the Successive Quadratic Programming method.

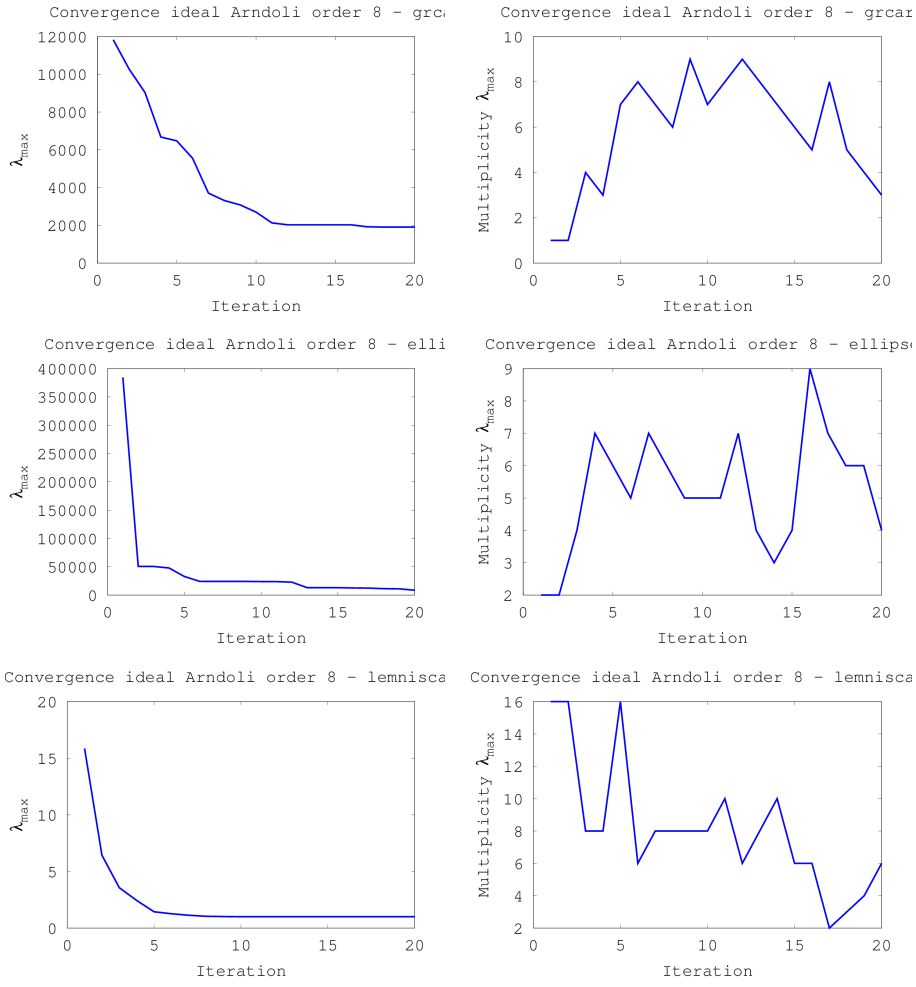

Figure 5.4: Convergence results for computation of ideal Arnoldi polynome of degree 8

Chapter 6

Conclusion

In this thesis I have presented my research on eigenvalue optimization problems and how those problems are linked to semidefinite optimization problems, how SASPA can be used to compute intersecting eigencurves and developed a novel algorithm to solve eigenvalue optimization problems based on those insights.

That semidefinite optimization problems and eigenvalue optimization problems are closely linked is well-known. However the relation developed in chapter 2, section 2.1 based on rotating and perturbing is original. This relation is useful in simplifying large sparse semidefinite optimization problems such as max-cut. However, a sever drawback is that for some instances more memory is required to store the eigenvalue optimization problem than the semidefinite problem.

Also was shown how SASPA can be used to compute intersecting eigencurves. SASPA was recently developed by Rommes and Martins in 2008 [27]. The original application was computing sensitive poles in the analysis of large scale system matrices. The idea that SASPA can also be used to compute the intersection of eigencurves presented in chapter 3 is novel. The adapted method works efficient for large matrices for which eigenvalues have a large derivative. This method has been used successfully to develop a novel algorithm for solving eigenvalue optimization problems.

The new algorithm developed in chapter 4 for solving eigenvalue optimization problems is largely inspired by [21], [14] and [4]. Each presented an algorithm to tackle these kind of problems based on gradients, Newton steps and subgradients. Novel is that algorithm 4.3.1 is based on only partial information each step (that is that no full eigendecomposition is necessary) and computes for each step a new descent direction. This algorithm has been shown to be efficient for computing large and sparse instances of eigenvalue optimization problems in chapter 5.

Future research possibilities include modifying SASPA and algorithm 4.3.1 for parallelization, adapt algorithm 4.3.1 for use in matrix rank minimization problems and incorporating an adapted Conjugated Gradients method such as the Fletcher-Reeves method.

Paralysing SASPA will benefit the method immensely. There are many

matrix-vector products which can be calculated much faster on a multi-core machine than on a single-core. It is exciting to see how much SASPA can benefit by going parallel.

As a beneficiary side-effect, if SASPA can be paralysed in combination with a paralysed version of Jacobi-Davidson (such as in [19]), then algorithm 4.3.1 can also be paralysed. It is expected that a paralysed version of this algorithm is able to solve considerable larger eigenvalue optimization problems in shorter time.

Many current applications which are closely related to semidefinite optimization problems are matrix rank minimization problems. That are problems in which in addition to regular semidefinite constraints there is also a constraint on the rank of X. Two important examples were shown in section 1.4: max-cut and phase retrieval. Phase retrieval is very important because it arises in many applications including MRI, astronomical imaging and microscopy.

Algorithm 4.3.1 can be used to solve the dual of relaxations of such problems to semidefinite optimization problems. It is expected that this algorithm in combination with other methods, such as the $\log\det$ heuristic [10], is able to solve these kind of problems. Solving these matrix-rank optimization problems through semidefinite optimization is not new [25], but the considering of a dual solving algorithm through eigenvalue optimization is novel. Due to the limitation of primal solvers on the instance size (primal solvers use much more memory), rank minimization problems can currently only be solved for small instances. It is interesting to see if with a (paralysed) dual solving algorithm larger instances can be solved.

Also, algorithm 4.3.1 has a drawback near the optimum in which computation of a descent direction takes considerable more time. It is suspected that many descent directions that are computed at this stage are not orthogonal and form a zig-zag path around the optimum. A Conjugated Gradient version such Fletcher-Reeves method will improve the convergence rate.

In conclusion, through research into eigenvalue optimization problems and SASPA a novel algorithm was developed which is shown to be able to tackle large eigenvalue optimization instances. However, its current implementation is severely limited by the fact that it runs single-core. With more research this method can be turned into a very fast method for solving eigenvalue optimization problems with a possible application to phase retrieval.

Appendix A

Implementation of some algorithms

A.1 Rotating SDP to Spectral Problems

This is the code based on theorem 2.1.4:

scripts/rotateSdpToSpectral.m

```
function [rotC, rotA, scale] = rotateSdpToSpectral (b, C, A)

    % First, determine rotation matrix that transforms b into e1
    [Q, R] = qr(b);

    if R(1) < 0
        Q = -Q;
        R = -R;
    end
    scale = R(1);

    % Second, determine matrices A^Q_{1..n}
    Aq = cell(size(A, 1), 1);

    for i = 1:size(A, 1)

        Aq{i} = sparse(size(A{1}, 1), size(A{1}, 1));

        for j = 1:size(A, 1)
            Aq{i} = Aq{i} + Q(j,i) .* A{j};
        end

    end

    % Third, make a decomposition of Aq{1}
    if Aq{1} == speye(size(A{1}, 1))

        rotA = Aq{2:size(A, 1)};
        rotC = C;
```

```
30
31      else
32
33          smallestEig = jd(Aq{1}, 'sa');
34
35  %        if smallestEig < 0
36  %            perturb = smallestEig + 0.01;
37  %            disp(sprintf('Warning: instance is permuted by %e',
        perturb))
38  %            b = [b; perturb];
39  %            A{end} = speye(size(A{1}));
40  %            [rotC, rotA, scale] = rotateSdpToSpectral(b, C, A);
41  %        else
42              Rchol = chol(Aq{1});
43              Rchol_inv = inv(Rchol);
44              rotC = Rchol_inv' * C * Rchol_inv;
45
46              rotA = cell(size(A, 1) - 1, 1);
47              for i = 1:size(A, 1) - 1
48                  rotA{i} = Rchol_inv' * Aq{i+1} * Rchol_inv;
49              end
50  %        end
51      end
52  end
```

A.2 SASPA to compute intersections

This is the code based on algorithm 3.5.2:

scripts/intersectMaxEigencurve.m

```
1   function [varargout] = intersectMaxEigencurve(varargin)
2
3           [A, dA, lambda_max, U, v_max, tol, itmax, kmin, kmax] = ...
4                   readOptions(varargin{:});
5
6           x = ones(size(A, 1), 1) + 0.05.*(2.*rand(size(A, 1), 1) -
                1);
7           X = lanczos(A, x, kmin);
8
9           deflate = orth(U);
10
11          X = X - U*(U'*X);
12          X = orth(X);
13
14          S = X' * A * X;
15          [V, D] = eig(S);
16          [s, i] = max(diag(D));
17          x = X*V(:, i);
18
19          t = Inf; tm = Inf;
20
21          u = [];
22          lambda = 0;
23
```

```
24          printf('Iter: \t|\t Eigenvalue: \t|\t Residue: \t|\t
                Crosspoint:\n')
25          printf('
                ------------------------------------------------------------------------------\
                n')
26
27          converged = false; iteration = 0;
28          stagcheck = false;
29          while ~converged && iteration < itmax
30
31                  if stagcheck
32                          x = x + 0.05.*(2.*rand(size(x))-1);
33                          stagcheck = false;
34                  end
35
36                  b = dA * x;
37                  if size(deflate, 2) > 0
38                          b = b - deflate * (deflate'*b);
39                  end
40
41                  x = bicgstab(A - s * speye(size(A)), b, 1e-8, 100);
42                  %x = (A - s * speye(size(A))) \ b;
43
44                  x(isinf(x)) = 0;
45                  if size(deflate, 2) > 0
46                          x = x - deflate * (deflate'*x);
47                  end
48
49                  if sum(isnan(x)) > 0
50                          stagcheck = true;
51                  else
52                          stag = size(X, 2);
53                          X = orth([X x]);
54                          if stag == size(X, 2)
55                                  stagcheck = true;
56                          end
57                  end
58
59                  S = X' * A * X; B = X' * dA * X;
60
61                  [V, D] = eig(S);
62                  T = zeros(size(V, 2), 1);
63                  for i = 1:size(V, 2)
64                          r = (lambda_max - D(i, i)) / ...
65                                  (V(:, i)' * B * V(:, i) - v_max);
66                          if ~isnan(r) && isreal(r) && r > tol
67                                  T(i) = r;
68                          else
69                                  T(i) = Inf;
70                          end
71                  end
72                  [T, I] = sort(T);
73                  V = V(:, I); D(I, I);
74
75                  x = X * V(:, 1);
76                  s = D(1, 1);
77
```

```
78                  res = norm(A*x - s.*x);
79
80                  printf('%d \t|\t %f \t|\t %f \t|\t %e\n', ...
81                                  iteration, s, res, T(1))
82
83                  tm = min(tm, T(1));
84
85                  if res > tol && res < 1
86                          [x, s] = RQI(A, x, s, tol);
87                  end
88
89                  if norm(A*x - s.*x) < tol
90
91                          printf('-----------------------------------
                                converged
                                -----------------------------------\n')
92                          printf('\tt: %e \tlambda: %f\tres: %e\n',
                                ...
93                                          T(1), s, norm(A*x-s.*x))
94
95                          if T(1) < t
96                                  u = x;
97                                  lambda = s;
98                                  t = T(1);
99                                  s = lambda_max;
100
101                                 if ~isinf(t) && t <= tm
102 %                                       converged = true;
103                                 end
104                         elseif ~isinf(t)
105 %                                 converged = true;
106                         end
107
108                         printf('\tt*: %e\tlambda*: %f\n', t, lambda
                                )
109                         printf('
                                -----------------------------------------------------------------
                                n')
110
111                         deflate = orth([deflate x]);
112
113                         X = X - deflate * (deflate'*X);
114                         X(:, 1) = X(:, 1) ./ norm(X(:, 1));
115                         X = orth(X);
116                         i = 2; while i < size(X, 2)
117                                 x = X(:, i) - X(:, 1:i-1) * (X(:,
                                        1:i-1)'* X(:, i));
118                                 if norm(x) > 1e-6
119                                         X(:, i) = x ./ norm(x);
120                                         i = i + 1;
121                                 else
122                                         X = [X(:, 1:i-1), X(:, i+1:
                                                end)];
123                                 end
124                         end
125                         X = orth(X);
126                         x = X * (2 .* rand(size(X, 2), 1) - 1);
```

```
127
128                 elseif size(X, 2) > kmax
129
130                         X = X * V(:, 1:kmin);
131                         X = orth(X);
132
133                 end
134
135                 if T(1) > t
136                         s = lambda_max;
137                 end
138
139                 iteration = iteration + 1;
140         end
141
142         if nargout == 1
143                 varargout{1} = u;
144         elseif nargout == 2
145                 varargout{1} = u;
146                 varargout{2} = lambda;
147         elseif nargout == 3
148                 varargout{1} = u;
149                 varargout{2} = lambda;
150                 varargout{3} = full(t);
151         elseif nargout >= 4
152                 varargout{1} = u;
153                 varargout{2} = lambda;
154                 varargout{3} = full(t);
155                 varargout{4} = full(tm);
156         end
157
158 end
159
160 function [A, dA, lambda_max, U, v_max, tol, itmax, kmin, kmax] =
    ...
161         readOptions(varargin)
162
163     A = varargin{1};
164     dA = varargin{2};
165
166     i = 3;
167
168     if length(varargin) >= i+1
169         lambda_max = varargin{i};i=i+1;
170         U = varargin{i};i=i+1;
171     else
172         [V, D] = eigs(A, 1, 'la');
173         lambda_max = D(1,1);
174         U = V(:, 1);
175     end
176
177     if length(varargin) >= i
178         v_max = varargin{i}; i=i+1;
179     else
180         v_max = max(U' * dA * U);
181     end
182
```

```
183     if length(varargin) < i, tol = 1e-8;   else tol = varargin{i};
            end, i=i+1;
184     if length(varargin) < i, itmax = 200;  else itmax = varargin{i
            }; end, i=i+1;
185     if length(varargin) < i, kmin = 4;     else kmin = varargin{i};
            end, i=i+1;
186     if length(varargin) < i, kmax = 8;     else kmax = varargin{i};
            end, i=i+1;
187
188 end
189
190 function V = lanczos(A, v0, n)
191
192         V = zeros(size(A, 1), n);
193         V(:, 1) = v0;
194
195         for i = 1:n-1
196
197                 v = A*V(:,i);
198                 a = v'*V(:,i);
199                 if i > 1, v = v - a.*V(:,i) - b.*V(:,i-1);
200                 else,     v = v - a.*V(:,i);
201                 end
202                 b = norm(v);
203                 V(:,i+1) = v ./ b;
204
205         end
206
207 end
```

A.3 Spectral Descent

:

This is the code based on algorithm 4.3.1.

scripts/spectralDescent.m

```
1  function [lambda_max, z, A, conv_his, rank_his, f_his] =
       spectralDescent (C, opA, P, z0, outz, output)
2
3          tau = 0.1; dtol = 1e-10; tstart = tic;
4
5          n = size(C, 1); m = size(opA, 1);
6          if size(z0, 1) == m && size(z0, 2) == 1
7                  z = z0; A = C;
8                  for i = 1:m, A = A - z(i).*opA{i}; end
9          else
10                 z = zeros(m, 1); A = C;
11         end
12
13         [Q, lambda_max] = eigenspace(A, 'la', [], tau);
14         lambda_max = max(diag(lambda_max));
15         conv_his = [lambda_max];
16         rank_his = [];
17         f_his = [];
```

```
18
19          not_converged = true;
20          while not_converged && tau > 1e-8
21
22                  if size(output, 2) > 1
23                          fid = fopen(output, 'a');
24                          fprintf(fid, ...
25                                  '%f : Calculating steepest
                                        direction, rank Q: %d\tlambda:
                                        %d\n', ...
26                                  toc(tstart), size(Q, 2), lambda_max
                                        );
27                          fclose(fid);
28                  end
29
30                  d = zeros(m, 1);
31                  if size(Q, 2) == 1
32                          for i = 1:m
33                                  d(i) = Q' * opA{i} * Q;
34                          end
35                          d = d ./ norm(d);
36                  else
37                          t = size(Q, 2);
38
39                          f = zeros(m+1, 1); f(1) = 1;
40                          Al = zeros(t*(t+1)/2, m+1);
41  %                        Al(:, 1) = ones(t*(t+1)/2, 1);
42                          temp = eye(t);
43                          Al(:, 1) = temp(tril(true(t)));
44                          for i = 1:m
45                                  temp = Q' * opA{i} * Q;
46                                  Al(:, i+1) = temp(tril(true(t)));
47                          end
48                          b = zeros(t*(t+1)/2, 1);
49                          lb = -ones(m+1, 1); lb(1) = -Inf;
50                          ub =  ones(m+1, 1); ub(1) =  Inf;
51
52  %                        x = linprog(f, Al, b, [], [], lb, ub);
53                          T = repmat('L', (t*(t+1)/2), 1);
54                          T(temp(tril(true(t))) == 0) = 'S';
55                          x = glpk(f, Al, b, lb, ub, T);
56                          d = x(2:m+1);
57
58                          if x(1) >= -dtol || isnan(x(1))
59                                  [Q, lambdas] = eigenspace(A, 'la',
                                        [], tau);
60                                  lambdas = diag(lambdas); lambda_max
                                        = max(lambdas);
61                                  tau = lambda_max - min(lambdas) - 1
                                        e-8;
62                                  if size(output, 2) > 1
63                                          fid = fopen(output, 'a');
64                                          fprintf(fid, ...
65                                                  '%f : No descent
                                                      direction found
                                                      , decreasing
                                                      tau to: %f\n',
```

```
                                                    ...
66                                                      toc(tstart), tau);
67                                              fclose(fid);
68                                      end
69                                      if tau < 1e-8
70                                              if size(output, 2) > 1
71                                                      fid = fopen(output,
                                                            'a');
72                                                      fprintf(fid, ...
73                                                              '%f :
                                                                    Converged
                                                                    , Df: %
                                                                    e\
                                                                    tLambda
                                                                    : %e\
                                                                    tTau: %
                                                                    f\n',
                                                                    ...
74                                                              toc(tstart)
                                                                    , x(1),

                                                                    lambda_max
                                                                    , tau);
75                                                      fclose(fid);
76                                              end
77                                              not_converged = false;
78                                              rank_his = [rank_his; size(
                                                    Q, 2)];
79                                              break
80                                      end
81                                      cut = abs(lambdas - lambda_max) <
                                            tau;
82                                      Q = Q(:, cut);
83                                      continue;
84                              end
85                      end
86
87                      rank_his = [rank_his; size(Q, 2)];
88
89                      if size(P, 1) > 0
90                              b = P*z; I = find(b <= 0);
91                              r = quadprog(speye(m), [], -P(I, :), P(I,
                                    :)*d);
92                              d = d + r;
93                      end
94
95                      dA = sparse(n, n);
96                      for i = 1:m
97                              dA = dA + d(i) .* opA{i};
98                      end
99
100                     f = -jd(Q'*dA*Q, 'sa');
101                     b = -jd(Q'*dA*Q, 'la');
102
103                     if (f >= -dtol || b >= dtol) && size(Q, 2) > 1
104                             lambdas = diag(Q'*A*Q);
105                             tau = lambda_max - min(lambdas) - 1e-8;
```

```
106                         if size(output, 2) > 1
107                                 fid = fopen(output, 'a');
108                                 fprintf(fid, ...
109                                         '%f : No descent direction
                                                found, decreasing tau
                                                to: %f\n', ...
110                                         toc(tstart), tau);
111                                 fclose(fid);
112                         end
113                         if tau < 1e-8
114                                 if size(output, 2) > 1
115                                         fid = fopen(output, 'a');
116                                         fprintf(fid, ...
117                                                 '%f : Converged, Df
                                                        : %e\tLambda: %
                                                        e\tTau: %f\n',
                                                        ...
118                                                 toc(tstart), f,
                                                        lambda_max);
119                                         fclose(fid);
120                                 end
121                                 not_converged = false;
122                                 break
123                         end
124                         cut = abs(lambdas - lambda_max) < tau;
125                         Q = Q(:, cut);
126                         continue;
127                 end
128
129                 if f >= -dtol
130
131                         if size(output, 2) > 1
132                                 fid = fopen(output, 'a');
133                                 fprintf(fid, ...
134                                         '%f : Converged, Df: %e\tBf
                                                : %e\tLambda: %e\n',
                                                ...
135                                         toc(tstart), f, b,
                                                lambda_max);
136                                 fclose(fid);
137                         end
138                         not_converged = false;
139                         break
140
141                 end
142
143                 f_his = [f_his; f];
144                 if size(output, 2) > 1
145                         fid = fopen(output, 'a');
146                         fprintf(fid, ...
147                                 '%f : Starting step calculation, Df
                                        : %e\n', ...
148                                 toc(tstart), f);
149                         fclose(fid);
150                 end
151
152                 while f < -dtol
```

```
153
154                     [q, lambda, t, tm] = intersectMaxEigencurve
                            (A, -dA, lambda_max, Q, f);
155
156                     if t == Inf && tm == Inf, t = 10/max(max(dA
                            )); end
157                     if t == Inf,                t = tm; end
158
159                     if size(output, 2) > 1
160                             fid = fopen(output, 'a');
161                             fprintf(fid, ...
162                                     '%f : Found intersection at
                                            , t: %e\n', ...
163                                     toc(tstart), t);
164                             fclose(fid);
165                     end
166
167                     if size(P, 1) > 0
168                             tmax = - min((P*z) ./ (P*d));
169                             t = min(tmax, t);
170                     end
171
172                     [Qt, lambda_t, dist] = eigenspace(A - t.*dA
                            , 'la', [Q q], tau);
173                     lambda_t = max(diag(lambda_t));
174
175                     if size(output, 2) > 1
176                             fid = fopen(output, 'a');
177                             fprintf(fid, ...
178                                     '%f : After step, rank Q: %
                                            d\tLambda: %e\n', ...
179                                     toc(tstart), size(Qt, 2),
                                            lambda_t);
180                             fclose(fid);
181                     end
182
183                     ft = -jd(Qt'*dA*Qt, 'sa');
184                     bt = -jd(Qt'*dA*Qt, 'la');
185
186                     if size(output, 2) > 1
187                             fid = fopen(output, 'a');
188                             fprintf(fid, ...
189                                     '%f : New Df: %e\t Bf: %e\n
                                            ', ...
190                                     toc(tstart), ft, bt);
191                             fclose(fid);
192                     end
193
194                     while ft > dtol && bt > dtol
195
196                             par = -f*t/(2*((lambda_t-lambda_max
                                    )/t-f));
197                             int = (lambda_max-lambda_t+t*bt)/(
                                    bt-f);
198
199                             if int > 1e-12 && int < t
200                                     if par > int && par < t, nt
```

```
                                = par;
201                         else,                                   nt
                                = int;
202                         end
203                 elseif par > 1e-12 && par < t
204                         nt = par;
205                 else, break, end
206
207                 [nQt, nlambda_t, ndist] =
                        eigenspace(A - nt.*dA, 'la', Qt
                        , tau);
208                 nlambda_t = max(diag(nlambda_t));
209
210                 if size(nQt, 2) == 0, break, end
211
212                 t = nt; Qt = nQt; lambda_t =
                        nlambda_t; dist = ndist;
213
214                 ft = -jd(Qt'*dA*Qt, 'sa');
215                 bt = -jd(Qt'*dA*Qt, 'la');
216
217                 if size(output, 2) > 1
218                         fid = fopen(output, 'a');
219                         fprintf(fid, ...
220                                 '%f : Through
                                    parabola fit, t
                                    :%e\trank Q: %d
                                    \tLambda: %e\
                                    tDf: %e\tBf: %e
                                    \n', ...
221                                 toc(tstart), t,
                                    size(Qt, 2),
                                    lambda_t, ft,
                                    bt);
222                         fclose(fid);
223                 end
224         end
225
226         if ft > -dtol && bt < dtol && lambda_t >
                lambda_max
227
228                 lambdas = diag(Q'*A*Q);
229                 if max(lambdas) - min(lambdas) < 2e
                        -8, break, end
230                 tau = max(lambdas) - min(lambdas) -
                         1e-8;
231
232                 if size(output, 2) > 1
233                         fid = fopen(output, 'a');
234                         fprintf(fid, ...
235                                 '%f : Decreased tau
                                     to: %f\n', ...
236                                 toc(tstart), tau);
237                         fclose(fid);
238                 end
239
240                 cut = abs(lambdas - lambda_max) <
```

```
                    tau;
241                 Q = Q(:, cut);
242
243             elseif (ft > -dtol || abs(lambda_t -
                    lambda_max) < 1e-8) && size(Qt, 2) ==
                    size(Q, 2)
244
245                 tau = tau + dist + 1e-8;
246                 [Qt, lambda_t] = eigenspace(A-t.*dA
                        , 'la', Qt, tau);
247                 lambda_t = max(diag(lambda_t));
248
249                 if size(output, 2) > 1
250                     fid = fopen(output, 'a');
251                     fprintf(fid, ...
252                             '%f : Increased tau
                                    to: %f\n', ...
253                             toc(tstart), tau);
254                     fclose(fid);
255                 end
256
257                 if lambda_t >= lambda_max
258                     [Q, lambda_max] =
                            eigenspace(A, 'la', [],
                             tau);
259                     lambda_max = max(diag(
                            lambda_max));
260                 end
261
262             end
263
264
265             if lambda_t < lambda_max
266
267                 A = A - t.*dA;
268                 z = z + t.*d;
269                 Q = Qt;
270                 lambda_max = lambda_t;
271                 f = ft;
272                 b = bt;
273
274                 if size(outz, 2) > 1
275                     save(outz, 'z', 'A', 'Q', '
                            lambda_max', ...
276                                  '
                                         not_converged
                                         ', '
                                         conv_his
                                         ', '
                                         rank_his
                                         ', '
                                         f_his')
                                         ;
277                 end
278
279             else, break, end
280         end
```

```
281
282                     conv_his = [conv_his; lambda_max];
283                     if size(output, 2) > 1
284                             fid = fopen(output, 'a');
285                             fprintf(fid, ...
286                                     '\n%f : New iteration, lambda: %e\n
                                            ', ...
287                                             toc(tstart), lambda_max);
288                             fclose(fid);
289                     end
290                     if size(outz, 2) > 1
291                             save(outz, 'z', 'A', 'Q', 'lambda_max', '
                                    not_converged', ...
292                                             'conv_his', 'rank_his', '
                                                f_his');
293                     end
294             end
295             if size(outz, 2) > 1
296                     tend = toc(tstart);
297                     save(outz, 'z', 'A', 'Q', 'lambda_max', '
                            not_converged', 'tend', ...
298                                     'conv_his', 'rank_his', 'f_his');
299             end
300     end
```

Bibliography

[1] Dmitri Alekseevsky, Andreas Kriegl, Peter W Michor, and Mark Losik. Choosing roots of polynomials smoothly. *Israel Journal of Mathematics*, 105(1):203–233, 1998.

[2] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951.

[3] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.

[4] Mischja Alexander van Bossum. Semidefinite optimization, a spectral approach. 2002.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[6] Emmanuel J Candes, Yonina C Eldar, Thomas Strohmer, and Vladislav Voroninski. Phase retrieval via matrix completion. *SIAM Review*, 57(2):225–251, 2015.

[7] Francis H Clarke, Yuri S Ledyaev, Ronald J Stern, and Peter R Wolenski. *Nonsmooth analysis and control theory*. Springer Science & Business Media, 1998.

[8] Frank H Clarke. *Optimization and nonsmooth analysis*. John Wiley & Sons, 1983.

[9] Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of ising spin glasses: New experimental results with a branch-and-cut algorithm. *Journal of Statistical Physics*, 80(1-2):487–496, 1995.

[10] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. Log-det heuristic for matrix rank minimization with applications to hankel and euclidean distance matrices. In *American Control Conference, 2003. Proceedings of the 2003*, volume 3, pages 2156–2162. IEEE, 2003.

[11] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 1990.

[12] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995.

[13] Anne Greenbaum and Lloyd N Trefethen. Gmres/cr and arnoldi/lanczos as matrix approximation problems. *SIAM Journal on Scientific Computing*, 15(2):359–368, 1994.

[14] Christoph Helmberg and Franz Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[15] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[16] Donald Ervin Knuth. *The sandwich theorem*. Stanford University, Department of Computer Science, 1993.

[17] László Lovász. On the shannon capacity of a graph. *Information Theory, IEEE Transactions on*, 25(1):1–7, 1979.

[18] Jan R Magnus. On differentiating eigenvalues and eigenvectors. *Econometric Theory*, 1(02):179–191, 1985.

[19] Margreet Nool and Auke van der Ploeg. A parallel jacobi–davidson-type method for solving large generalized eigenvalue problems in magnetohydrodynamics. *SIAM Journal on Scientific Computing*, 22(1):95–112, 2000.

[20] Alexander M Ostrowski. On the convergence of the rayleigh quotient iteration for the computation of the characteristic roots and vectors. i. *Archive for Rational Mechanics and Analysis*, 1(1):233–241, 1957.

[21] Michael L Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 9(2):256–268, 1988.

[22] Michael L Overton. Large-scale optimization of eigenvalues. *SIAM Journal on Optimization*, 2(1):88–120, 1992.

[23] Beresford N Parlett. The rayleigh quotient iteration and some generalizations for nonnormal matrices. *Mathematics of Computation*, 28(127):679–693, 1974.

[24] G Pataki and SH Schmieta. The dimacs library of mixed semidefinite-quadratic-linear programs. 2002.

[25] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[26] Joost Rommes and Nelson Martins. Efficient computation of transfer function dominant poles using subspace acceleration.

[27] Joost Rommes and Nelson Martins. Computing large-scale system eigenvalues most sensitive to parameter changes, with applications to power system small-signal stability. *Power Systems, IEEE Transactions on*, 23(2):434–442, 2008.

[28] Joost Rommes and Gerard LG Sleijpen. Convergence of the dominant pole algorithm and rayleigh quotient iteration. *SIAM Journal on Matrix Analysis and Applications*, 30(1):346–363, 2008.

[29] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[30] Gerard LG Sleijpen and Henk A Van der Vorst. A jacobi–davidson iteration method for linear eigenvalue problems. *SIAM Review*, 42(2):267–293, 2000.

[31] David Titley-Peloquin, Jennifer Pestana, and Andrew J Wathen. Gmres convergence bounds that depend on the right-hand-side vector. *IMA Journal of Numerical Analysis*, 34(2):462–479, 2014.

[32] Kim-Chuan Toh and Lloyd N Trefethen. The chebyshev polynomials of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 20(2):400–419, 1998.

[33] Philip Wolfe. The simplex method for quadratic programming. *Econometrica: Journal of the Econometric Society*, pages 382–398, 1959.