CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

MASTER'S THESIS

Semidefinite Programming for Geometric Problems in Computer Vision

Pavel Trutman

pavel.trutman@fel.cvut.cz

April 19, 2017

Thesis Advisor: Ing. Tomáš Pajdla, PhD.

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Acknowledgements

Author’s declaration

I declare that I have work out the presented thesis independently and that I have listed all information sources used in accordance with the Methodical Guidelines about Maintaining Ethical Principles for Writing Academic Theses.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Abstract

Keywords:

Abstrakt

Keywords:

Contents

List of Figures

List of Algorithms

List of Listings

List of Symbols and Abbreviations

$\mathrm{cl}(S)$	Closure of the set S.
$\mathrm{dom}\ f$	Domain of the function f.
$\mathrm{Dom}\ f$	$\mathrm{cl}(\mathrm{dom}\ f)$.
$\mathrm{int}\ S$	Interior of the set S.
$\mathcal{P}^n$	Cone of positive semidefinite $n \times n$ matrices.
$\mathcal{S}^n$	Space of $n \times n$ real symmetric matrices.
$\mathrm{tr}(A)$	Trace of the matrix A.
$x^{(i)}$	i-th element of the vector x.

1. Introduction

[2]

2. Semidefinite programming

2.1. Preliminaries on semidefinite programs

We introduce here some notation and preliminaries about symmetric matrices and semidefinite programs. We will introduce further notation and preliminaries later on in the text when needed.

At the beginning, let us denote the inner product for two vectors x, $y \in \mathbb{R}^n$

$$\langle x, y\rangle = \sum_{i=1}^{n} x^{(i)} y^{(i)} \tag{2.1}$$

and the Frobenius inner product for two matrices X, $Y \in \mathbb{R}^{n\times m}$.

$$\langle X, Y\rangle = \sum_{i=1}^{n}\sum_{j=1}^{m} X^{(i,j)} Y^{(i,j)} \tag{2.2}$$

2.1.1. Symmetric matrices

Let $\mathcal{S}^n$ denotes the space of $n \times n$ real symmetric matrices.

For matrix $M \in \mathcal{S}^n$, the notation $M \succeq 0$ means that M is positive semidefinite. $M \succeq 0$ if and only if any of the following equivalent properties holds.

1. $x^\top M x \geq 0$ for all $x \in \mathbb{R}^n$.
2. All eigenvalues of M are nonnegative.

For matrix $M \in \mathcal{S}^n$, the notation $M \succ 0$ means that M is positive definite. $M \succ 0$ if and only if any of the following equivalent properties holds.

1. $M \succeq 0$ and $\operatorname{rank} M = n$.
2. $x^\top M x > 0$ for all $x \in \mathbb{R}^n$.
3. All eigenvalues of M are positive.

2.1.2. Semidefinite programs

The standard (primal) form of a semidefinite program in variable $X \in \mathcal{S}^n$ is defined as follows:

$$\begin{aligned} p^* = \sup_{X\in\mathcal{S}^n} \quad & \langle C, X\rangle \\ \text{s.t.} \quad & \langle A_i, X\rangle = b^{(i)} \quad (i = 1, \ldots, m) \\ & X \succeq 0 \end{aligned} \tag{2.3}$$

where C, A_1, $\ldots$, $A_m \in \mathcal{S}^n$ and $b \in \mathbb{R}^m$ are given.

The dual form of the primal form is the following program in variable $y \in \mathbb{R}^m$.

$$\begin{aligned} d^* = \inf_{y\in\mathbb{R}^m} \quad & b^\top y \\ \text{s.t.} \quad & \sum_{i=1}^{m} A_i y^{(i)} - C \succeq 0 \end{aligned} \tag{2.4}$$

2.2. State of the art review

2.3. Nesterov's approach

In this section, we will follow Chapter 4 of [5] by Y. Nesterov, which is devoted to the convex minimization problems. We will extract from it only the minimum, just to be able to introduce an algorithm for semidefinite programs solving. We will present some basic definitions and theorems, but we will not prove them. For the proofs look into [5].

2.3.1. Self-concordant functions

Definition 2.1 (Self-concordant function in $\mathbb{R}$). A closed convex function $f : \mathbb{R} \mapsto \mathbb{R}$ is self-concordant if there exist a constant $M_f \geq 0$ such that the inequality

$$|f'''(x)| \leq M_f f''(x)^{3/2} \tag{2.5}$$

holds for all $x \in \operatorname{dom} f$.

For better understanding of self-concordant functions, we provide several examples.

Example 2.1.

1. Linear and convex quadratic functions.

$$f'''(x) = 0 \text{ for all } x \tag{2.6}$$

   Linear and convex quadratic functions are self-concordant with constant $M_f = 0$.

2. Negative logarithms.

$$\begin{aligned} f(x) &= -\ln(x) \text{ for } x > 0 && (2.7)\\ f'(x) &= -\frac{1}{x} && (2.8)\\ f''(x) &= \frac{1}{x^2} && (2.9)\\ f'''(x) &= -\frac{2}{x^3} && (2.10)\\ \frac{|f'''(x)|}{f''(x)^{3/2}} &= 2 && (2.11) \end{aligned}$$

   Negative logarithms are self-concordant functions with constant $M_f = 2$.

3. Exponential functions.

$$\begin{aligned} f(x) &= e^x && (2.12)\\ f''(x) = f'''(x) &= e^x && (2.13)\\ \frac{|f'''(x)|}{f''(x)^{3/2}} &= e^{-x/2} \to \infty \text{ as } x \to -\infty && (2.14) \end{aligned}$$

   Exponential functions are not self-concordant functions.

Definition 2.2 (Self-concordant function in $\mathbb{R}^n$). A closed convex function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is self-concordant if function

$$g(t) = f(x + tv) \tag{2.15}$$

is self-concordant for all $x \in \operatorname{dom} f$ and all $v \in \mathbb{R}^n$.

Now, let us focus on the main properties of self-concordant functions.

Theorem 2.1. Let functions f_i be self-concordant with constants M_i and let $\alpha_i > 0$, $i = 1, 2$. Then the function

$$f(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) \tag{2.16}$$

is self-concordant with constant

$$M_f = \max\left\{\frac{1}{\sqrt{\alpha_1}} M_1, \frac{1}{\sqrt{\alpha_2}} M_2\right\} \tag{2.17}$$

and

$$\operatorname{dom} f = \operatorname{dom} f_1 \cap \operatorname{dom} f_2. \tag{2.18}$$

Corollary 2.1. Let function f be self-concordant with some constant M_f and $\alpha > 0$. Then the function $\phi(x) = \alpha f(x)$ is also self-concordant with the constant $M_\phi = \frac{1}{\sqrt{\alpha}} M_f$.

We call function $f(x)$ as the standard self-concordant function if $f(x)$ is some self-concordant function with the constant $M_f = 2$. Using Corollary 2.1, we can see that any self-concordant function can be transformed into the standard self-concordant function by scaling.

Theorem 2.2. Let function f be self-concordant. If $\operatorname{dom} f$ contains no straight line, then the Hessian $f''(x)$ is nondegenerate at any x from $\operatorname{dom} f$.

For some self-concordant function $f(x)$, for which we assume, that $\operatorname{dom} f$ contains no straight line (which implies that all $f''(x)$ are nondegenerate, see Theorem 2.2), we denote two local norms as

$$\|u\|_x = \sqrt{u^\top f''(x) u} \tag{2.19}$$

$$\|u\|_x^* = \sqrt{u^\top f''(x)^{-1} u}. \tag{2.20}$$

Consider following minimization problem

$$x^* = \arg \min_{x \in \operatorname{dom} f} f(x) \tag{2.21}$$

as the minimization of the self-concordant function $f(x)$. Algorithm 2.1 is describing the iterative process of solving optimization problem (2.21). The algorithm is divided into two stages by the value of $\|f'(x_k)\|_{x_k}^*$. The splitting parameter β guarantees quadratic convergence rate for the second part of the algorithm. The parameter β is chosen from interval $(0, \bar{\lambda})$, where

$$\bar{\lambda} = \frac{3 - \sqrt{5}}{2}, \tag{2.22}$$

which is a root of the equation

$$\frac{\lambda}{(1-\lambda)^2} = 1. \tag{2.23}$$

The first while loop (lines 2 – 5) represents damped Newton method, where at each iteration we have

$$f(x_k) - f(x_{k+1}) \geq \beta - \ln(1+\beta) \text{ for } k \geq 0, \tag{2.24}$$

Algorithm 2.1. Newton method for minimization of self-concordant functions.

Input:

f a self-concordant function to minimize
$x_0 \in \operatorname{dom} f$ a starting point
$\beta \in (0, \bar{\lambda})$ a parameter of size of the region of quadratic convergence
ε a precision

Output:

x^* an optimal solution to the minimization problem (2.21)

```
1: $k \leftarrow 0$
2: while $\|f'(x_k)\|^*_{x_k} \geq \beta$ do
3:     $x_{k+1} \leftarrow x_k - \frac{1}{1+\|f'(x_k)\|^*_{x_k}} f''(x_k)^{-1} f'(x_k)$
4:     $k \leftarrow k + 1$
5: end while
6: while $\|f'(x_k)\|^*_{x_k} > \varepsilon$ do
7:     $x_{k+1} \leftarrow x_k - f''(x_k)^{-1} f'(x_k)$
8:     $k \leftarrow k + 1$
9: end while
10: return $x^* \leftarrow x_k$
```

where

$$\beta - \ln(1+\beta) \quad > \quad 0 \text{ for } \beta > 0, \tag{2.25}$$

and therefore the global convergence of the algorithm is ensured. It can be shown that the local convergence rate of the damped Newton method is also quadratic, but the presented switching strategy is preferred as it gives better complexity bounds.

The second while loop of the algorithm (lines 6 – 9) is the standard Newton method with quadratic convergence rate.

The algorithm terminates when the required precision ε is reached.

2.3.2. Self-concordant barriers

To be able to introduce self-concordant barriers, let us denote $\operatorname{Dom} f = \operatorname{cl}(\operatorname{dom} f)$.

Definition 2.3 (Self-concordant barrier). Let $F(x)$ be a standard self-concordant function. We call it a ν-self-concordant barrier for set $\operatorname{Dom} F$, if

$$\sup_{u\in\mathbb{R}^n} \left(2u^\top F'(x) - u^\top F''(x)u\right) \quad \leq \quad \nu \tag{2.26}$$

for all $x \in \operatorname{dom} F$. The value ν is called the parameter of the barrier.

The inequality (2.26) can be rewritten into the following equivalent matrix notation:

$$F''(x) \quad \succeq \quad \frac{1}{\nu} F'(x)F'(x)^\top. \tag{2.27}$$

In Definition 2.3, the hessian $F''(x)$ is not required to be nondegenerate. However, in case that $F''(x)$ is nondegenerate, the inequality (2.26) is equivalent to

$$F'^\top(x)F''(x)^{-1}F'(x) \quad \leq \quad \nu. \tag{2.28}$$

Let us explore, which basic functions are self-concordant barriers.

Example 2.2.

1. Linear functions.

$$F(x) = \alpha + a^\top x, \ \operatorname{dom} F = \mathbb{R}^n \tag{2.29}$$
$$F''(x) = 0 \tag{2.30}$$

From (2.27) and for $a \neq 0$ follows, that linear functions are not self-concordant barriers.

2. Convex quadratic functions.
For $A = A^\top \succ 0$:

$$F(x) = \alpha + a^\top x + \frac{1}{2} x^\top A x, \ \operatorname{dom} F = \mathbb{R}^n \tag{2.31}$$
$$F'(x) = a + Ax \tag{2.32}$$
$$F''(x) = A \tag{2.33}$$

After substitution into (2.28) we obtain

$$(a + Ax)^\top A^{-1}(a + Ax) = a^\top A^{-1} a + 2a^\top x + x^\top A x, \tag{2.34}$$

which is unbounded from above on $\mathbb{R}^n$. Therefore, quadratic functions are not self-concordant barriers.

3. Logarithmic barrier for a ray.

$$F(x) = -\ln x, \ \operatorname{dom} F = \left\{ x \in \mathbb{R} \mid x > 0 \right\} \tag{2.35}$$
$$F'(x) = -\frac{1}{x} \tag{2.36}$$
$$F''(x) = \frac{1}{x^2} \tag{2.37}$$

From (2.28), when $F'(x)$ and $F''(x)$ are both scalars, we get

$$\frac{F'(x)^2}{F''(x)} = \frac{x^2}{x^2} = 1. \tag{2.38}$$

Therefore, the logarithmic barrier for a ray is a self-concordant barrier with parameter $\nu = 1$ on domain $\left\{ x \in \mathbb{R} \mid x > 0 \right\}$.

Now, let us focus on the main properties of self-concordant barriers.

Theorem 2.3. Let $F(x)$ be a self-concordant barrier. Then the function $c^\top x + F(x)$ is a self-concordant function on $\operatorname{dom} F$.

Theorem 2.4. Let F_i be ν_i-self-concordant barriers, $i = 1, 2$. Then the function

$$F(x) = F_1(x) + F_2(x) \tag{2.39}$$

is a self-concordant barrier for convex set

$$\operatorname{Dom} F = \operatorname{Dom} F_1 \cap \operatorname{Dom} F_2 \tag{2.40}$$

with the parameter

$$\nu = \nu_1 + \nu_2. \tag{2.41}$$

Theorem 2.5. Let $F(x)$ be a ν-self-concordant barrier. Then for any $x \in \operatorname{Dom} F$ and $y \in \operatorname{Dom} F$ such that

$$(y - x)^\top F'(x) \quad \geq \quad 0, \tag{2.42}$$

we have

$$\|y - x\|_x \quad \leq \quad \nu + 2\sqrt{\nu}. \tag{2.43}$$

There is one special point of convex set, which is important for solving convex minimization problem. It is called analytic center of convex set and we will focus on its properties.

Definition 2.4. Let $F(x)$ be a ν-self-concordant barrier for the set $\operatorname{Dom} F$. The point

$$x_F^* \quad = \quad \arg \min_{x \in \operatorname{Dom} F} F(x) \tag{2.44}$$

is called the analytic center of convex set $\operatorname{Dom} F$, generated by the barrier $F(x)$.

Theorem 2.6. Assume that the analytic center of a ν-self-concordant barrier $F(x)$ exists. Then for any $x \in \operatorname{Dom} F$ we have

$$\|x - x_F^*\|_{x_F^*} \quad \leq \quad \nu + 2\sqrt{\nu}. \tag{2.45}$$

This property clearly follows from Theorem 2.5 and the fact, that $F'(x_F^*) = 0$.

Thus, if $\operatorname{Dom} F$ contains no straight line, then the existence of x_F^* (which leads to nondegenerate $F''(x_F^*)$) implies, that the set $\operatorname{Dom} F$ is bounded.

Now, we describe the algorithm and its properties for obtaining an approximation to the analytic center. To find the analytic center, we need to solve the minimization problem (2.44). For that, we will use the standard implementation of the damped Newton method with termination condition

$$\|F'(x_k)\|_{x_k}^* \quad \leq \quad \beta, \text{ for } \beta \in (0, 1). \tag{2.46}$$

The pseudocode of the whole minimization process is shown in Algorithm 2.2.

Algorithm 2.2. Damped Newton method for analytic centers.

Input:

F a ν-self-concordant barrier
$x_0 \in \operatorname{Dom} F$ a starting point
$\beta \in (0, 1)$ a centering parameter

Output:

x_F^* an approximation of the analytic center of the set $\operatorname{Dom} F$

```
1: $k \leftarrow 0$
2: while $\|F'(x_k)\|_{x_k}^* > \beta$ do
3:     $x_{k+1} \leftarrow x_k - \frac{1}{1+\|F'(x_k)\|_{x_k}^*} F''(x_k)^{-1} F'(x_k)$
4:     $k \leftarrow k + 1$
5: end while
6: return $x_F^* \leftarrow x_k$
```

Theorem 2.7. Algorithm 2.2 terminates no later than after N steps, where

$$N = \frac{1}{\beta - \ln(1+\beta)}\big(F(x_0) - F(x_F^*)\big). \tag{2.47}$$

The knowledge of analytic center allows us to solve the standard minimization problem

$$x^* = \arg\min_{x \in Q} c^\top x \tag{2.48}$$

with bounded closed convex set $Q \equiv \operatorname{Dom} F$, which has nonempty interior, and which is endowed with a ν-self-concordant barrier $F(x)$. Denote

$$f(t,x) = tc^\top x + F(x), \text{ for } t \geq 0 \tag{2.49}$$

as a parametric penalty function. Using Theorem 2.3 we can see, that $f(t,x)$ is self-concordant in x. Let us introduce new minimization problem using the parametric penalty function $f(t,x)$

$$x^*(t) = \arg\min_{x \in \operatorname{dom} F} f(t,x). \tag{2.50}$$

This trajectory is called the central path of the problem (2.48). We will reach the solution $x^*(t) \to x^*$ as $t \to \infty$. Moreover, since the set Q is bounded, the analytic center x_F^* of this set exists and

$$x^*(0) = x_F^*. \tag{2.51}$$

From the first-order optimality condition, any point of the central path satisfies equation

$$tc + F'\big(x^*(t)\big) = 0 \tag{2.52}$$

Since the analytic center lies on the central path and can be found by Algorithm 2.2, all we have to do, to find the solution x^*, is to follow the central path. This enables us an approximate centering condition:

$$\|f'(t,x)\|_x^* = \|tc + F'(x)\|_x^* \leq \beta, \tag{2.53}$$

where the centering parameter β is small enough.

Assuming $x \in \operatorname{dom} F$, one iteration of the path-following algorithm consists of two steps:

$$t_+ = t + \frac{\gamma}{\|c\|_x^*}, \tag{2.54}$$

$$x_+ = x - F''(x)^{-1}\big(t_+ c + F'(x)\big). \tag{2.55}$$

Theorem 2.8. Let x satisfy the approximate centering condition (2.53)

$$\|tc + F'(x)\|_x^* \leq \beta \tag{2.56}$$

with $\beta < \bar{\lambda} = \frac{3-\sqrt{5}}{2}$. Then for γ, such that

$$|\gamma| \leq \frac{\sqrt{\beta}}{1+\sqrt{\beta}} - \beta, \tag{2.57}$$

we have again

$$\|t_+ c + F'(x_+)\|_{x_+}^* \leq \beta. \tag{2.58}$$

Algorithm 2.3. Path following algorithm.

Input:

F a ν-self-concordant barrier

$x_0 \in \operatorname{dom} F$ a starting point satisfying $\|F'(x_0)\|^*_{x_0} \leq \beta$, e.g. the analytic center x^*_F of the set $\operatorname{Dom} F$

$\beta \in (0,1)$ a centering parameter

γ a parameter satisfying $|\gamma| \leq \frac{\sqrt{\beta}}{1+\sqrt{\beta}} - \beta$

$\epsilon > 0$ an accuracy

Output:

x^* an optimal solution to the minimization problem (2.48)

```
1: $t_0 \leftarrow 0$
2: $k \leftarrow 0$
3: while $\epsilon t_k < \nu + \frac{(\beta+\sqrt{\nu})\beta}{1-\beta}$ do
4:     $t_{k+1} \leftarrow t_k + \frac{\gamma}{\|c\|^*_{x_k}}$
5:     $x_{k+1} \leftarrow x_k - F''(x_k)^{-1}\big(t_{k+1}c + F'(x_k)\big)$
6:     $k \leftarrow k+1$
7: end while
8: return $x^* \leftarrow x_k$
```

This theorem ensures the correctness of the presented iteration of the path-following algorithm. For the whole description of the path-following algorithm, please see Algorithm 2.3.

Theorem 2.9. Algorithm 2.3 terminates no more than after N steps, where

$$N \leq \mathcal{O}\left(\sqrt{\nu}\ln\frac{\nu\|c\|^*_{x^*_F}}{\epsilon}\right). \tag{2.59}$$

The parameters β and γ in Algorithm 2.2 and Algorithm 2.3 can be fixed. The reasonable values are:

$$\beta = \frac{1}{9}, \tag{2.60}$$

$$\gamma = \frac{\sqrt{\beta}}{1+\sqrt{\beta}} - \beta = \frac{5}{36}. \tag{2.61}$$

The union of Algorithm 2.2 and Algorithm 2.3 can be easily used to solve the standard minimization problem (2.48), supposing we have a feasible point $x_0 \in Q$.

2.3.3. Barrier function for semidefinite programming

In this section, we are going to show, how to find a self-concordant barrier for the semidefinite program (2.4), so that we can use Algorithm 2.2 and Algorithm 2.3 to solve it. For the purpose of this section, we are interested only in the constrains of the problem. The constrains are defining us the feasibility set Q:

$$Q = \left\{y \in \mathbb{R}^m \mid A_0 + \sum_{i=1}^{m} A_i y^{(i)} \succeq 0\right\}, \tag{2.62}$$

where $A_0, \dots, A_m \in \mathcal{S}^n$. Let us denote $X(y) = A_0 + \sum_{i=1}^{m} A_i y^{(i)}$. If the matrix $X(y)$ is block diagonal

$$X(y) = \begin{bmatrix} X_1(y) & 0 & \cdots & 0 \\ 0 & X_2(y) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X_k(y) \end{bmatrix} \tag{2.63}$$

with $X_j(y) \in \mathcal{S}^{n_j}$ for $j = 1, \dots, k$ and $\sum_{j=1}^{k} n_j = n$, then the feasibility set Q can be expressed as

$$Q = \big\{ y \in \mathbb{R}^m \mid X_j(y) \succeq 0,\ j = 1, \dots, k \big\}. \tag{2.64}$$

This rule allows us to easily add or remove some constraints without touching the others and to keep the sizes of the used matrices small, which can significantly speed up the computation.

Instead of the set Q, which is parametrized by y, we can directly optimize over the set of positive semidefinite matrices. This set $\mathcal{P}^n$ is defined as

$$\mathcal{P}^n = \big\{ X \in \mathcal{S}^n \mid X \succeq 0 \big\} \tag{2.65}$$

and it is called the cone of positive semidefinite $n \times n$ matrices. This cone is a closed convex set, which interior is formed by positive definite matrices and on its boundary lie matrices, which have at least one eigenvalue equal to zero.

Let us denote the function $F(X)$:

$$F(X) = -\ln \prod_{i=1}^{n} \lambda_i(X), \tag{2.66}$$

where $X \in \operatorname{int} \mathcal{P}^n$ and $\big\{\lambda_i(X)\big\}_{i=1}^{n}$ is the set of eigenvalues of the matrix X. To avoid the computation of eigenvalues, the function $F(X)$ can be also expressed as:

$$F(X) = -\ln \det(X). \tag{2.67}$$

Theorem 2.10. Function $F(X)$ is an n-self-concordant barrier for $\mathcal{P}^n$.

Note, that $\operatorname{Dom} F \supseteq \mathcal{P}^n$ because $\det(X) \geq 0$ when the number of negative eigenvalues of X is even. Therefore, the set $\operatorname{Dom} F$ is made by separated subsets, which one of them is $\mathcal{P}^n$. As Algorithm 2.2 and Algorithm 2.3 are interior point algorithms, when the starting point is from $\operatorname{int} \mathcal{P}^n$, then we never leave $\mathcal{P}^n$ during the execution of the algorithms and the optimal solution is found.

Similarly, the self-concordant barrier function for the set Q is a function

$$F(y) = -\ln \det \big(X(y)\big). \tag{2.68}$$

Example 2.3. To make it clearer, what is the difference between the set Q and $\operatorname{Dom} F(y)$, we have prepared this example. Let

$$X(y) = \begin{bmatrix} y_2 & y_1 \\ y_1 & y_2 \end{bmatrix}, \tag{2.69}$$

where $y = \begin{bmatrix} y_1 & y_2 \end{bmatrix}^\top$. The equation

$$z = \det(X(y)) = y_2^2 - y_1^2 \tag{2.70}$$

represents a hyperbolic paraboloid, which you can see in Figure 2.1. Therefore, the equation $z = 0$ is a slice of it, denoted by the purple color in Figure 2.2. The domain of the self-concordant barrier function is

$$\operatorname{Dom} F(y) \quad = \quad \Big\{ y \mid \det\big(X(y)\big) \geq 0 \Big\} \tag{2.71}$$

and is shaded by the blue color. We can see, that the set $\operatorname{Dom} F(y)$ consists of two disjoint parts. One of them is the set, where $X(y) \succeq 0$ (denoted by the orange color) and the second part is an area, where both eigenvalues of $X(y)$ are negative. Therefore, one has to pick his starting point x_0 from the interior of the set Q to obtain the optimal solution from the set Q.

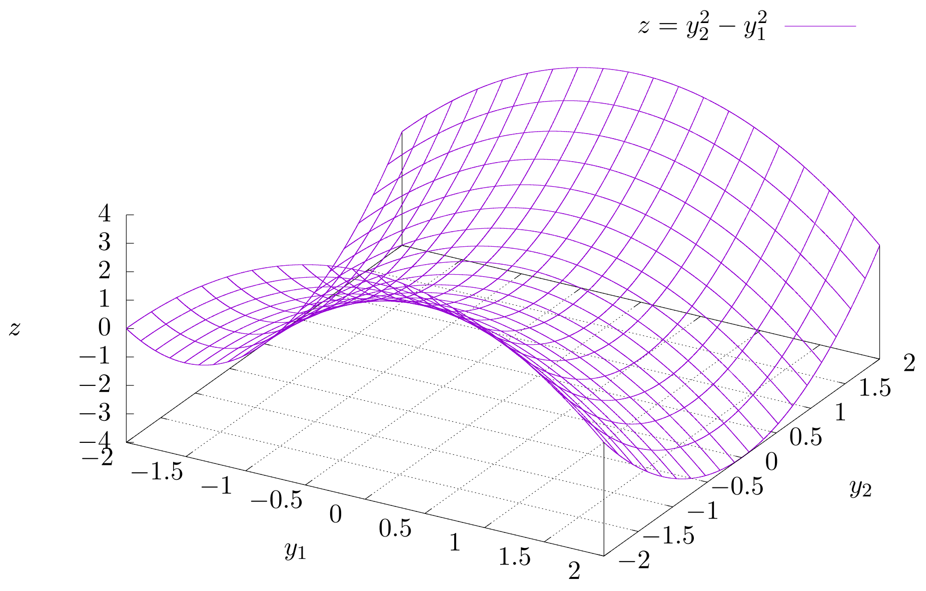

Figure 2.1. Hyperbolic paraboloid $z = y_2^2 - y_1^2$.

When the matrix X has the block diagonal form (2.63), we can rewrite the barrier function (2.68) into summation form

$$F(y) \quad = \quad -\sum_{j=1}^{k} \ln\det\big(X_j(y)\big). \tag{2.72}$$

For the purposes of Algorithm 2.2 and Algorithm 2.3, we need the first and the second partial derivatives of this function. Let us denote $X_j(y) = A_{j,0} + \sum_{i=1}^{m} A_{j,i} y^{(i)}$ for $j = 1, \ldots, k$, then the derivatives are:

$$\frac{\partial F}{\partial y^{(u)}}(y) \quad = \quad -\sum_{j=1}^{k} \operatorname{tr}\big(X_j(y)^{-1} A_{j,u}\big), \tag{2.73}$$

$$\frac{\partial^2 F}{\partial y^{(u)} \partial y^{(v)}}(y) \quad = \quad \sum_{j=1}^{k} \operatorname{tr}\Big(\big(X_j(y)^{-1} A_{j,u}\big)\big(X_j(y)^{-1} A_{j,v}\big)\Big), \tag{2.74}$$

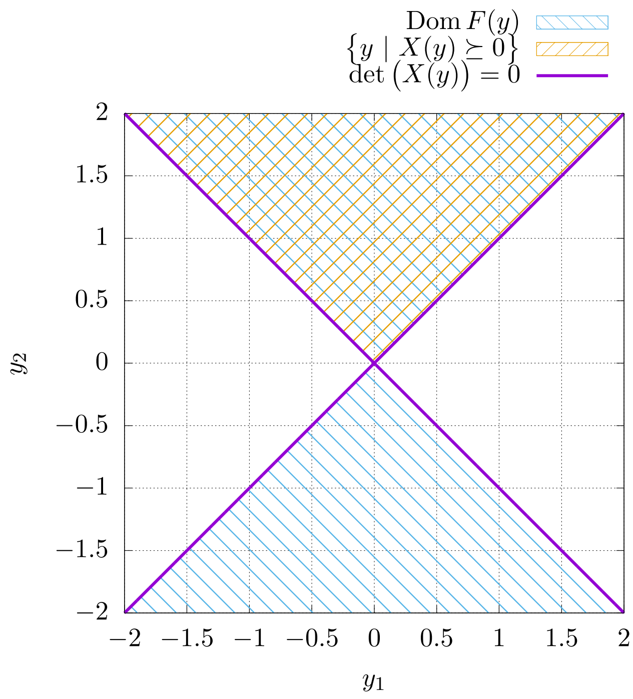

Figure 2.2. Illustration of the sets $\mathrm{Dom}\, F(y)$ and $\big\{y \mid X(y) \succeq 0\big\}$.

for $u, v = 1, \ldots, m$.

The computation of the derivatives is the most expensive part of each step of Algorithm 2.2 and Algorithm 2.3. Therefore, the estimated number of arithmetic operations of computation of the derivatives is also the complexity of each step in the algorithms. The number of arithmetic operations for j-th constraint in form $\big\{y \mid X_j(y) \succeq 0\big\}$ is:

- the computation of $X_j(y) = A_{j,0} + \sum_{i=1}^{m} A_{j,i} y^{(i)}$ needs mn^2 operations,
- the computation of the inversion $X_j(y)^{-1}$ needs n^3 operations,
- to compute all matrices $X_j(y)^{-1} A_{j,u}$ for $u = 1, \ldots, m$ is needed mn^3 operations,
- to compute $\mathrm{tr}\big(X_j(y)^{-1} A_{j,u}\big)$ for $u = 1, \ldots, m$ is needed mn operations,
- the computation of $\mathrm{tr}\Big(\big(X_j(y)^{-1} A_{j,u}\big)\big(X_j(y)^{-1} A_{j,v}\big)\Big)$ for $u, v = 1, \ldots, m$ needs $m^2 n^2$ operations.

The most expensive parts requires mn^3 and $m^2 n^2$ arithmetic operations on each constraint. Typically, the value k, the number of constraints, is small and it keeps constant, when the semidefinite programs are generated as subproblems, when solving more complex problems, e.g. polynomial optimization. Therefore, we can say, that k is constant and we can omit it from the complexity estimation. To sum up, one step of Algorithm 2.2 and Algorithm 2.3 requires

$$\mathcal{O}\big(m(m+n)n^2\big) \tag{2.75}$$

arithmetic operations.

2.4. Implementation details

To be able to study the algorithms described previously in this section, we have implemented them in the programming language Python [7]. The full knowledge of the code allows us to trace the algorithms step by step and inspect their behaviors. Instead of using some state of the art toolboxes for semidefinite programming, e.g. SeDuMi [6] and MOSEK [4], which are more or less black boxes for us, the knowledge of the used algorithms allows us to decide, if the chosen algorithm is suitable for the given semidefinite problem or not. Moreover, if we would like to create some specialized solver for some class of semidefinite problems, we can easily reuse the code, edit it as required and build the solver very quickly. On the other hand, we can not expect that our implementation will be as fast as the implementation of some state of the art toolboxes, as much more time and people was used for developing them.

The implementation is compatible with Python version 3.6 and higher. The package NumPy is used for linear algebra computations. Please refer to the installation guide of NumPy for your system to ensure, that it is correctly set to use the linear algebra libraries, e.g. LAPACK [1], ATLAS [8] and BLAS [3]. The incorrect setting of these libraries causes significant drop of the performance. Other Python packages are required as well, e.g. SymPy and SciPy, but theirs settings are not so crucial for the performance of this implementation.

2.4.1. Package installation

The package with implementation of Algorithm 2.2 and Algorithm 2.3 is named polyopt, as the semidefinite programming part of this package is only a tool, which is used for polynomial optimization, which will be described in Section ??. The newest version of the package is available at `http://cmp.felk.cvut.cz/~trutmpav/master-thesis/polyopt/`. To install the package on your system, you have to clone and checkout the Git repository with the source codes of the package. To install other packages that are required, the preferred way is to use the pip[1] installer. The required packages are listed in the `requirements.txt` file. Then, install the package using the script `setup.py`. For the exact commands for the whole installation process, please see Listing 2.1.

Listing 2.1. Installation of the package polyopt.

```
git clone https://github.com/PavelTrutman/polyopt.git
cd polyopt
pip3 install -r requirements.txt
python3 setup.py install
```

To check, whether the installation was successful, run command `python3 setup.py test` and the predefined tests will be automatically run. If no error emerges, then the package is installed and ready to use.

2.4.2. Usage

The polyopt package is created to be able to solve semidefinite programs in a form

$$\begin{aligned} y^* &= \arg\min_{y\in\mathbb{R}^m} \quad c^\top y \\ &\quad \text{s.t.} \qquad A_{j,0} + \sum_{i=1}^{m} A_{j,i} y^{(i)} \succeq 0 \text{ for } j = 1, \ldots, k, \end{aligned} \tag{2.76}$$

[1]The PyPA recommended tool for installing Python packages. See `https://pip.pypa.io`.

where $A_{j,i} \in \mathcal{S}^{n_j}$ for $i = 0, \dots m$ and $j = 1, \dots, k$, $c \in \mathbb{R}^m$ and k is the number of constraints. In addition, a strictly feasible point $y_0 \in \mathbb{R}^m$ must be given.

The semidefinite program solver is implemented in the class `SDPSolver`. Firstly, the problem is initialized by the matrices $A_{j,i}$ and the vector c. Then, the function `solve` is called with parameter y_0 as the starting point and with the method for the analytic center estimation. A choice from two methods is available, firstly, the method `dampedNewton`, which corresponds to Algorithm 2.2, and secondly, the method `auxFollow`, which is the implementation of the Auxiliary path-following scheme [5]. However, the `auxFollow` method is unstable and it fails in some cases, and therefore it is not recommended to use. The function `solve` returns the optimal solution y^*. The minimal working example is shown in Listing 2.2.

Listing 2.2. Typical use of the class `SDPSolver` of the polyopt package.

```
import polyopt

# supposing the matrices Aij and the vectors c and y0 are already
    defined
problem = polyopt.SDPSolver(c, [[A10, A11, ..., A1m], ..., [Ak0,
    Ak1, ..., Akm]])
yStar = problem.solve(y0, problem.dampedNewton)
```

Detailed info can be printed out during the execution of the algorithm. This option can be set by `problem.setPrintOutput(True)`. Then, in each iteration of Algorithm 2.2 and Algorithm 2.3, the values k, x_k and eigenvalues of $X_j(x_k)$ are printed.

If n, the dimension of the problem, is equal to 2, boundary of the set $\operatorname{Dom} F$ (2.71) and all intermediate points x_k can be plotted. This can be enabled by setting `problem.setDrawPlot(True)`. An example of such a graph is shown in Figure 2.3.

The parameters β and γ are predefined to the same values as in (2.60) and (2.61). These parameters can be set to different values by assigning to the variables `problem.beta` and `problem.gamma` respectively. The default value for the accuracy parameter ϵ is 10^{-3}. This value can be changed by overwriting the variable `problem.eps`.

The function `problem.getNu()` returns the ν parameter of the self-concordant barrier function used for the problem according to Theorem 2.10. When the problem is solved, we can obtain the eigenvalues of $X(y^*)$ by calling `problem.eigenvalues()`. We should observe, that some of them are positive and some of them are zero (up to the numerical precision). The zero eigenvalues mean, that we have reached the boundary of the set Q, because the optimal solution lies always on the boundary of the set Q.

It may happen, that the set $\operatorname{Dom} F$ is not bounded, but the optimal solution is attained. In this case, the analytic center does not exists and the proposed algorithms can not be used. By adding a constrain for $R \in \mathbb{R}$

$$X_{k+1}(y) = \begin{bmatrix} R^2 & y^{(1)} & y^{(2)} & \cdots & y^{(m)} \\ y^{(1)} & 1 & 0 & \cdots & 0 \\ y^{(2)} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y^{(m)} & 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{2.77}$$

we bound the set by the ball with radius R. The constrain (2.77) is equivalent to

$$\|y\|_2^2 \leq R^2. \tag{2.78}$$

This will make the set $\operatorname{Dom} F$ bounded and the analytic center can by found in the standard way by Algorithm 2.2. When optimizing the linear function by Algorithm 2.3, the radius R may be set too small and the optimum may be found on the boundary of the constrain (2.77). Then, the found optimum is not the solution of the original problem and the algorithm has to be run again with bigger value of R. The optimum is found on the boundary of the constrain (2.77), if at least one of the eigenvalues of $X_{k+1}(y^*)$ is zero. In our implementation, the artificial bounding constrain (2.77) can be set by `problem.bound(R)`. When the problem is solved, we can list the eigenvalues of $X_{k+1}(y^*)$ by the function `problem.eigenvalues('bounded')`.

Example 2.4. Let us present a simple example to show a detailed usage of the package polyopt. Let us have semidefinite program in a form

$$\begin{aligned} y^* &= \arg\min_{y\in\mathbb{R}^2} \quad y^{(1)} + y^{(2)} \\ &\quad \text{s.t.} \quad \begin{bmatrix} 1+y^{(1)} & y^{(2)} & 0 \\ y^{(2)} & 1-y^{(1)} & y^{(2)} \\ 0 & y^{(2)} & 1-y^{(1)} \end{bmatrix} \succeq 0 \end{aligned} \tag{2.79}$$

with starting point

$$y_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top. \tag{2.80}$$

Listing 2.3 shows the Python code used to solve the given problem. The graph of the problem is showed in Figure 2.3. The analytic center of the feasible region of the problem is

$$y_F^* = \begin{bmatrix} -0.317 & 0 \end{bmatrix}^\top, \tag{2.81}$$

the optimal solution is attained at

$$y^* = \begin{bmatrix} -0.778 & -0.592 \end{bmatrix}^\top \tag{2.82}$$

and the objective function has value -1.37. The eigenvalues of $X(y^*)$ are

$$\left\{\lambda_i\big(X(y^*)\big)\right\}_{i=0}^{3} = \{2.32\cdot 10^{-4};\ 1.32;\ 2.45\}. \tag{2.83}$$

2.5. Comparison with the state of the art methods

Listing 2.3. Code for solving semidefinite problem stated in Example 2.4.

```python
from numpy import *
import polyopt

# Problem statement
# min c1*y1 + c2*y2
# s. t. A0 + A1*y1 + A2*y2 >= 0
c = array([[1], [1]])
A0 = array([[1,  0,  0],
            [0,  1,  0],
            [0,  0,  1]])
A1 = array([[1,  0,  0],
            [0, -1,  0],
            [0,  0, -1]])
A2 = array([[0,  1,  0],
            [1,  0,  1],
            [0,  1,  0]])

# starting point
y0 = array([[0], [0]])

# create the solver object
problem = polyopt.SDPSolver(c, [[A0, A1, A2]])

# enable graphs
problem.setDrawPlot(True)

# enable informative output
problem.setPrintOutput(True)

# solve!
yStar = problem.solve(y0, problem.dampedNewton)

# print eigenvalues of X(yStar)
print(problem.eigenvalues())
```

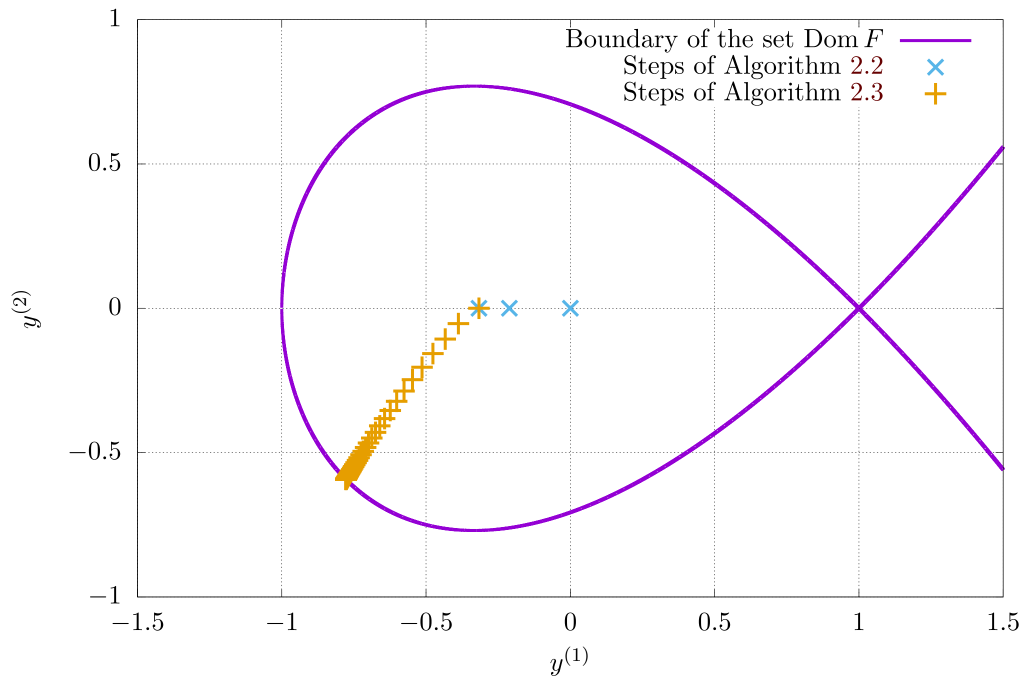

Figure 2.3. Graph of the semidefinite optimization problem stated in Example 2.4.

3. Conclusion

A. Contents of the enclosed CD

```
/
└── thesis
    └── thesis.pdf ...............digital copy of this thesis
```

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition, 1999. 18

[2] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, New York, USA, 2nd edition, 1997. 6

[3] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979. 18

[4] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28)*, 2015. `http://docs.mosek.com/7.1/toolbox/index.html`. 18

[5] Yurii Nesterov. *Introductory lectures on convex optimization : A basic course*. Springer, 2004. 8, 19

[6] Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Version 1.05 available from `http://fewcal.kub.nl/sturm`. 18

[7] Guido van Rossum and Fred L. Drake. *The Python Language Reference Manual*. Network Theory Ltd, 2011. 18

[8] R. Clint Whaley and Antoine Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps` [Online; accessed 2017-04-18]. 18