Computational Convex Analysis using Parametric Quadratic Programming

by

Khan Md. Kamall Jakee

B.Sc., Bangladesh University of Engineering and Technology, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES

(Interdisciplinary Studies - Optimization)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

September 2013

© Khan Md. Kamall Jakee, 2013
Abstract

The class of piecewise linear-quadratic (PLQ) functions is a very important class of functions in convex analysis since the result of most convex operators applied to a PLQ function is a PLQ function. Although there exists a wide range of algorithms for univariate PLQ functions, recent work has focused on extending these algorithms to PLQ functions with more than one variable. First, we recall a proof in [Convexity, Convergence and Feedback in Optimal Control, Phd thesis, R. Goebel, 2000] that PLQ functions are closed under partial conjugate computation. Then we use recent results on parametric quadratic programming (pQP) to compute the inf-projection of any multivariate convex PLQ function. We implemented the algorithm for bivariate PLQ functions, and modified it to compute conjugates. We provide a complete space and time worst-case complexity analysis and show that for bivariate functions, the algorithm matches the performance of [Computing the Conjugate of Convex Piecewise Linear-Quadratic Bivariate Functions, Bryan Gardiner and Yves Lucet, Mathematical Programming Series B, 2011] while being easier to extend to higher dimensions.
Preface

A part of this thesis (Theorem 3.2) is included in a manuscript co-authored with Dr. Yves Lucet and Bryan Gardiner. The manuscript is submitted for publication in Computational Optimization and Applications [GJL13].
Table of Contents

Abstract .............................................. ii
Preface .............................................. iii
Table of Contents ....................................... iv
List of Figures ......................................... vi
Acknowledgements ....................................... vii
Dedication ............................................... viii
Chapter 1: Introduction ................................. 1

Chapter 2: Convex Analysis Preliminaries ............... 5

Chapter 3: The Partial Conjugates ....................... 13

Chapter 4: Convex Parametric Quadratic Programming .... 18
  4.1 Piecewise quadratic optimization problem ............... 18
  4.2 Parametric piecewise quadratic optimization problems .... 26
    4.2.1 Critical region ...................................... 27
    4.2.2 Calculation of the solution .......................... 29
    4.2.3 Convex parametric quadratic program (pQP) ........... 30
  4.3 Adaptation of pQP in computational convex analysis ....... 34
    4.3.1 Conjugate computation ............................... 34
    4.3.2 Moreau envelope computation .......................... 41
    4.3.3 Proximal average computation ....................... 42

Chapter 5: Algorithm ..................................... 44
  5.1 Data structure ....................................... 44
    5.1.1 List representation of bivariate PLQ functions ....... 44
**TABLE OF CONTENTS**

5.1.2 Face-constraint adjacency representation of a bivariate PLQ function ................................................. 45
5.1.3 The data structures to store lines, rays, segments and vertexes ................................................................. 46
5.1.4 The data structure to store entity-face and entity-constraint adjacency information ................................. 48
5.1.5 The Half-Edge data structure [CGA, Ope] ......................... 50
5.2 The Algorithms ........................................................................................................................................ 52
5.3 Complexity Analysis ................................................................................................................................. 61

Chapter 6: Examples ....................................................................................................................................... 65

Chapter 7: Conclusion ..................................................................................................................................... 75

Bibliography ................................................................................................................................................... 76
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Example of normal cones</td>
<td>11</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Polyhedral decomposition of $D$</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Polyhedral subdivision of $S$</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>The index set of $x$</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>The polyhedral decomposition of $D$</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>The partitions of $D$</td>
<td>28</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>Illustration of $\kappa(x_3) = \kappa(x_4)$</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Illustration of $G_{J_8}$ and $D_{J_8}$</td>
<td>29</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Domain of the function, $f(x_1, x_2) =</td>
<td>x_1</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Domain of the function, $f(x_1, x_2) =</td>
<td>x_1</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>The half-edge data structure.</td>
<td>51</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>The representation of the domain of $f(x_1, x_2) =</td>
<td>x_1</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Computation of the vertexes.</td>
<td>55</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Computation of the remaining entities by using computed vertexes.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>2D Energy function.</td>
<td>63</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>The $l_1$ norm and its conjugate.</td>
<td>66</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>2D Energy function.</td>
<td>66</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>The function from Example 6.3 and its conjugate</td>
<td>67</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>The function from Example 6.4 and its conjugate</td>
<td>67</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>The function from Example 6.5 and its conjugate</td>
<td>68</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>The function from Example 6.6 and its conjugate</td>
<td>69</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>The function from Example 6.7 and its conjugate</td>
<td>71</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>2D Energy function.</td>
<td>72</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>2D Energy function.</td>
<td>73</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>2D Energy function.</td>
<td>74</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>The approximation of conjugate of $f(x_1, x_2) = x_1^4 + x_2^4$.</td>
<td>74</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Yves Lucet for his continuous support throughout my thesis with his patience and knowledge.

Beside my supervisor, I would like to thank my thesis committee members for spending time to read this thesis and agreeing to be a part of this memorable moment of my life.

A special thank to Dr. Warren Hare for the course Convex Optimization and Non-smooth Analysis which helps me a lot in my research.

I would like to thank my current and previous lab mates: Shahadat Hossain, Sukanto Mondal, Muhammad Mahmudul Hasan Razib and Dessalegn Amenu Hirpa for their helping hands.

I thank my parents and my wife for their support and encouragement in my life.

The research is funded by a Discovery Grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. It was performed in the Computer-Aided Convex Analysis (CA$^2$) lab funded through a Canadian Foundation for Innovation Leaders Opportunity Fund.
Dedication

To my wife, Sadia Sharmin.
Chapter 1

Introduction

Convex analysis is a branch of Mathematics which focuses on the study of the properties of convex sets and functions. Numerous applications in engineering and science use convex optimization, a subfield of optimization that studies the minimization of convex functions over convex sets.

Convex analysis provides numerous operators, like the Legendre-Fenchel conjugate (also known as the conjugate in convex analysis), the Moreau envelope (Moreau-Yosida approximate or Moreau-Yosida regularization), and the proximal average. Computational convex analysis focuses on the computation of these operators. It originated with the work of Moreau [Mor65, Paragraph 5c]. It was significantly studied with the introduction of the fast Legendre transform algorithm [Bre89, Cor96, Luc96, NV94, SAF92].

Piecewise linear-quadratic (PLQ) functions have very interesting properties. The class of PLQ functions is closed under the conjugate [RW98, Theorem 11.14] and Moreau envelope [RW98, Example 11.28] operators. Moreover, the partial conjugate of a PLQ function is also a PLQ function. When the closed form of a function does not exist or it is very complex, the piecewise linear-quadratic approximation is a convenient way to manipulate it. Algorithms based on PLQ functions [Luc06] do not suffer the resulting numerical errors due to the composition of several convex operators with unbounded domains that affects algorithms based on piecewise linear functions [Luc06]. These algorithms have been restricted to univariate functions until very recently.

Symbolic computation algorithms have been given to compute the Legendre-Fenchel conjugate and related transforms [BM06, BH08, Ham05]. It is possible to handle univariate [BM06] and multi-variate [BH08] functions with these algorithms. When symbolic computation is not applicable, the need for numerical algorithms arises.

The Fast Legendre Transform (FLT) [Bre89] was the beginning of the development of efficient numerical algorithms. Its worst-case time complexity was log-linear which was improved by the development of the Linear-time Legendre transform (LLT) algorithm [Luc97]. A piecewise linear model is used in the LLT algorithm and this model could be applied to multivari-
Chapter 1. Introduction

The Moreau envelope could be computed from the conjugate. Some other linear-time fast algorithms have been developed in [HUL07, Luc06] based on this relation. Beside the computation of the conjugate and the Moreau envelope, the fast algorithms are used to compute the proximal average [BLT08], which combines several fundamental operations of convex analysis like addition, scalar multiplication, conjugation, and regularization with the Moreau envelope. Other transforms like the proximal hull transforms, the inf-convolution of convex functions as well as the deconvolution of convex functions [HUL93, RW98], could be computed by combining existing fast algorithms.

The FLT and LLT algorithms have been applied beyond computational convex analysis. The LLT is used in network communication [HOVT03], robotics [Koo02], numerical simulation of multiphasic flows [Hel05], pattern recognition [Luc05] and analysis of the distribution of chemical compounds in the atmosphere [LPBL05]. The LLT algorithm has also explicit application in the computation of distance transforms [Luc05]. The addition and inf-convolution of convex functions are also related to the Linear Cost Network Flow problem on Series-Parallel Networks [TL96]. For a survey of applications, see [Luc10].

The linear dependence between the graph of the subdifferential of most convex operators and the graph of the function gives rise to graph-matrix-calculus (GPH) algorithms with linear time complexity [GL11b]. These algorithms store the subdifferential data for reducing the conjugate computation to a matrix multiplication. Since GPH algorithms are optimized for matrix operations, these algorithms are very suitable and efficient for implementation in matrix-based mathematical software like Scilab and Matlab.

The conjugate computation algorithm of bivariate convex PLQ functions [GL11a] is the first effective algorithm to compute the conjugate of convex bivariate PLQ functions. Its worst case running time is log-linear. This algorithm stores the function domain in a planar arrangement data structure provided by CGLAB [CGL]. A toolbox for convex PLQ functions based on this algorithm, has also been developed. The necessary operations to build this toolbox, like addition, scalar multiplication are implemented.

The full conjugate of a PLQ function could be computed from the partial conjugates (the conjugate with respect to one of the variables). Due to the similarities between the full conjugate and the partial conjugate, an algorithm to compute the partial conjugate of bivariate PLQ functions has been developed based on the conjugate computation algorithm of bivariate convex PLQ functions [GJL13]. Although partial conjugates are PLQ, they are not convex. In fact, they are saddle functions [Roc70].
Chapter 1. Introduction

Intense research has been done in the sensitivity analysis of solutions to nonlinear programs and variational inequalities during the last three decades [Fia83, RW09, KK02, DR09, FP03]. The sensitivity analysis in an optimization problem studies properties of the solution mapping only on the vicinity of a reference point in the parameter space. Sensitivity analysis is closely related to parametric optimization and sensitivity analysis could be applied in the algorithms for parametric optimization. But unlike sensitivity analysis, parametric optimization computes the solution mapping for every value in the parameter space [BGK+82, PGD07b, PGD07a].

Parametric optimization algorithms for linear and quadratic programs got interest in the last decade [BBM02b, BMDP02, BBM02a]. Ongoing research in parametric programming has resulted in the development of several algorithms for solving convex parametric quadratic programs (pQP) [Bao02, SGD03, TJB03, GBTM04] and parametric linear programs (pLP) [GBTM03]. Research has been done on the continuity properties of the value function and optimal solution set of parametric programs [Fia83, Zha97, BGK+83, Bes72, Hog73a, Hog73b]. The continuity of the optimal set mapping and the stability of the optimization problem are closely related to each other. The stability of quadratic programs is studied in [BC90, PY01]. The stability and continuity of parametric programs are mostly derived from set theory [Ber63, DFS67, Hau57]. The goal in parametric programming is to solve a parametric optimization problem for all possible values of the parameters. The solutions of a parameter dependent optimization problem are piecewise functions which are defined on the partition of the parameter space.

Parametric programming could be used to formulate several problems in control theory. Parametric linear program (pLP), parametric quadratic program (pQP), parametric mixed integer program (pMILP), parametric non-linear program (pNLP), parametric linear complementarity problem (pLCP) are the different types of parametric programs in control theory. We could get the exact and the explicit solution of dynamic programming equations of various classes of problems using parametric optimization. In [BBM02a, DB02], dynamic programming coupled with parametric programming is applied for min-max optimal control problems of constrained uncertain linear systems with a polyhedral performance index. It is also applied in the finite or infinite horizon optimal control of piecewise affine systems with polyhedral cost index [BCM06, CBM05]. Dynamic programming coupled with parametric programming is also used for the finite inf-sup optimal control of piecewise affine systems with polyhedral cost index [KM02, SKMJ09], and for problems with a quadratic cost index [BBBM09].
Chapter 1. Introduction

Parametric programming could be used beyond control theory. Due to the link between parametric programming and geometry, it has applications in computational geometry, i.e., parametric programming could be used to compute Voronoi Diagrams and Delaunay triangulations [RGJ04]. In addition, it has applications in convex analysis.

The parametric piecewise linear-quadratic optimization problem [PS11] enables us to use parametric programming in computational convex analysis. As long as the function to be optimized in an optimization problem is convex PLQ, that optimization problem could be modeled using parametric programming. Due to this fact, it is possible to model the conjugate computation problem as well as the Moreau envelope computation problem of any multi-variate convex PLQ function as a parametric optimization problem.

Although there is an effective algorithm to compute the conjugate of convex bivariate PLQ functions [GL11a], no work has been done to compute the convex conjugate of bivariate PLQ functions based on parametric programming. We study the first such algorithm. It relies on explicitly storing the partitions of the domain space as well as the associated linear-quadratic functions. The algorithm also stores the adjacency information of the partitions. The conjugate computation problem is modeled as a parametric piecewise quadratic optimization problem and convex parametric piecewise quadratic optimization is applied to compute the convex conjugate which is another bivariate PLQ function. PLQ functions have a simple structure and those functions could be represented easily. Although there are several ways to represent a bivariate PLQ function, we discuss different data structures to manipulate bivariate PLQ functions. These data structures are slightly different from the data structure presented in [GJL13], but are efficient and easily manipulable.

The thesis is organized in several chapters. After introducing the thesis in the current chapter, we discuss the notations and some basic definitions in the following chapter. Chapter 3 recalls partial conjugates. After that, we recall the recent results on convex parametric quadratic programming (pQP) in Chapter 4. We also discuss the adaptation of pQP in computational convex analysis in Chapter 4. Chapter 5 proposes several data structures to represent a convex bivariate PLQ function. It also presents an algorithm based on pQP to compute conjugate and Moreau envelope of any convex bivariate PLQ function. By using the algorithm discussed in Chapter 5, we generate some examples of conjugate and Moreau envelope computation. These examples are given in Chapter 6. Chapter 7 concludes the thesis.
Chapter 2
Convex Analysis
Preliminaries

We start with the notations we will use in the present thesis. We apply the standard arithmetic used in Convex Analysis as described in [RW98], and [BC11]. We note $\mathbb{R} \cup \{+\infty\}$ is the set of extended real numbers. The inner product of two column vectors $x, y \in \mathbb{R}^n$ is expressed by either $\langle x, y \rangle$ or $x^T y$. The notation $x^T$ denotes the transpose of $x$ i.e. the row vector $[x_1, \ldots, x_n]$. We use $\|x\|$ to denote the Euclidean norm of any vector $x \in \mathbb{R}^n$.

**Definition 2.1.** [Open set, closed set] A set $S$ is open if for any $x \in S$ there exists $\epsilon > 0$ such that $B_\epsilon(x) \subseteq S$, where $B_\epsilon(x) = \{y : \|y - x\| < \epsilon\}$.

A set $S$ is closed if for any sequence of points $x_1, x_2, \ldots$ such that $x_i \in S$ and $\lim_{i \to \infty} x_i = \pi$ we have $\pi \in S$.

**Definition 2.2.** [Interior of a set, closure of a set] The interior of a set $S$ is the largest open set inside $S$ and is denoted by $\text{Int}(S)$.

The closure of a set $S$ is the smallest closed set containing $S$. It is expressed by $\text{cl}(S)$.

**Definition 2.3.** [Convex set, convex hull, affine set, affine hull] A set $S$ is a convex set if given any two points $x_1, x_2 \in S$, and any $\theta \in [0, 1]$ we have $\theta x_1 + (1 - \theta)x_2 \in S$. The convex hull of a set $S \subseteq \mathbb{R}^n$ is the smallest convex set containing $S$. We use $\text{co} S$ to denote it.

A set $S$ is an affine set if given any two points $x_1, x_2 \in S$, and any $\theta \in \mathbb{R}$ we have $\theta x_1 + (1 - \theta)x_2 \in S$. The affine hull of a set $S \subseteq \mathbb{R}^n$ is the smallest affine set containing $S$. We denote it by $\text{aff}(S)$.

**Definition 2.4.** [Open ball] The open ball is defined by

$$B_\epsilon(x) = \{y : \|x - y\| < \epsilon\},$$

where $x, y \in \mathbb{R}^n$ and $\epsilon \in \mathbb{R}^+$. 


Chapter 2. Convex Analysis Preliminaries

Definition 2.5. [Relative interior] The relative interior of a set $S \subseteq \mathbb{R}^n$ is the interior relative to the affine hull of $S$. We denote it by $\text{ri} S$, i.e.

$$\text{ri} S = \{ y \in S : \exists \epsilon > 0, B_\epsilon(y) \cap \text{aff}(S) \subseteq S \}.$$ 

Definition 2.6. [Half-space] Given $a \in \mathbb{R}^n$, $a \neq 0$ and $b \in \mathbb{R}$ the half-space $H_{ab}$ is defined as

$$H_{ab} = \{ x : \langle a, x \rangle \leq b \}.$$ 

Definition 2.7. [Hyperplane] Given $a \in \mathbb{R}^n$, $a \neq 0$ and $b \in \mathbb{R}$ the hyperplane $h_{ab}$ is defined as

$$h_{ab} = \{ x : \langle a, x \rangle = b \}.$$ 

Definition 2.8. [Polyhedral set] A polyhedral set $S \subseteq \mathbb{R}^n$ is a set which can be expressed as an intersection of some finite collection of closed half-spaces. In other words, it can be specified by finitely many linear constraints, i.e., constraints $f_i(x) \leq 0$ or $f_i(x) = 0$ where $f_i$ is a affine function of $x \in \mathbb{R}^n$ and $i = \{1, 2 \ldots m\}$.

Definition 2.9. [Dimension of the affine hull of a set] Let $S$ be a non-empty set. The linear space by $S$ is $\text{lin} S = \{ \lambda x + \mu y : x, y \in S \text{ and } \lambda, \mu \in \mathbb{R} \}$ and has dimension $n = \text{dim}(\text{lin} S)$. For any $\lambda, \mu \in \mathbb{R}$ with $\lambda + \mu = 1$, the affine space generated by $S$ is $\text{aff}(S) = \{ \lambda x + \mu y : x, y \in S \}$ and has dimension $\text{dim}(\text{aff} S) = \text{dim}(\text{lin} S)$.

Definition 2.10. [Dimension of a set] The dimension of a set $S \subseteq \mathbb{R}^n$ is the dimension of $\text{aff}(S)$. It is denoted by $\text{dim}(S)$. If $\text{dim}(S) = n$, then $S$ is said to be full dimensional.

Definition 2.11. [Set difference, set complement] Assume $A$ and $B$ are two sets. The set difference of $A$ and $B$ is denoted by $A - B$ where $A - B = \{ x \in A : x \notin B \}$.

Let $S \subseteq U$ be a set. The complement of $S$ with respect to $U$ is denoted by $S^c$ with $S^c = U - S$. 

6
Chapter 2. Convex Analysis Preliminaries

Definition 2.12. [Edge, vertex, face] An edge is defined by the set $E = \{x \in \mathbb{R}^n : x = \lambda_1 x_1 + \lambda_2 x_2, \lambda_1 + \lambda_2 = 1\}$ with $x_1, x_2 \in \mathbb{R}^n$ and $x_1 \neq x_2$. It becomes a segment when $\lambda_1, \lambda_2 \geq 0$. An edge is called a ray if $\lambda_1 \geq 0$ but $\lambda_2 \notin \mathbb{R}$, or a line when $\lambda_1, \lambda_2 \in \mathbb{R}$.

A vertex is a starting point of a ray or it is one of the end points of a segment or it is an isolated point.

A face is a polyhedral set with nonempty interior.

Definition 2.13. [Indicator function] For any set $S \subseteq \mathbb{R}^n$, $I_S$ denotes the indicator function, i.e.

$$I_S(x) = \begin{cases} 0, & \text{if } x \in S; \\ \infty & \text{otherwise}. \end{cases}$$

Definition 2.14. [Effective domain] The effective domain of a function $f$ is the set of all points where $f$ takes a finite value. It is denoted by $\text{dom}(f)$.

Definition 2.15. [Proper function] Let $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ be a function. It is proper if it has nonempty domain and is greater than $-\infty$ everywhere.

Definition 2.16. [Epigraph] The epigraph of a function $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ is defined as

$$\text{epi}(f) = \{(x, \alpha) : \alpha \geq f(x)\}.$$ 

It is a set in $\mathbb{R}^{n+1}$.

Definition 2.17. [Convex function] Let $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$ be a proper function. Then it is convex if

$$f((1 - \lambda)x_1 + \lambda x_2) \leq (1 - \lambda)f(x_1) + \lambda f(x_2),$$

for all $x_1, x_2 \in \text{dom}(f)$, $\lambda \in [0, 1]$.

Definition 2.18. [Strictly convex function] A function $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ is strictly convex if

$$f((1 - \lambda)x_1 + \lambda x_2) < (1 - \lambda)f(x_1) + \lambda f(x_2),$$

for all $x_1, x_2 \in \text{dom}(f)$, $x_1 \neq x_2$, $\lambda \in ]0, 1[$.
**Definition 2.19.** [Concave function] A function $f$ is said to be a concave function if the function $-f$ is convex.

**Definition 2.20.** [Saddle function] Let $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{-\infty, +\infty\}$ be a function. It is called a convex-concave function if $f(x_1, x_2)$ is a convex function of $x_1 \in \mathbb{R}^m$ for each $x_2 \in \mathbb{R}^n$ and a concave function of $x_2 \in \mathbb{R}^n$ for each $x_1 \in \mathbb{R}^m$. Convex-concave functions are defined similarly. Both kinds of functions are called saddle functions.

**Definition 2.21.** [Additively separable function, additively non-separable function] Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty, +\infty\}$ be a function of $n$ variables. It is called an additively separable function if it can be written as $f(x_1, \ldots, x_n) = f(x_1) + \ldots + f(x_n)$ for some single variable functions $f(x_1), \ldots, f(x_n)$. Otherwise $f$ is an additively non-separable function.

**Definition 2.22.** [Convex subdifferential] Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a convex function. The convex subdifferential of $f$ at $x \in \mathbb{R}^n$ is defined by

$$\partial f(x) = \{ s : f(y) \geq f(x) + \langle s, y-x \rangle, \forall y \in \mathbb{R}^n \}.$$

**Definition 2.23.** [Partial Subdifferential] Let $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{+\infty\}$ be a proper convex function. The partial subdifferential of $f(\cdot, x_2)$ at $x_1 \in \mathbb{R}^m$ is defined by

$$\partial_1 f(x_1, x_2) = \{ s : f(x_1', x_2) \geq f(x_1, x_2) + \langle s, x_1' - x_1 \rangle, \forall x_1' \in \mathbb{R}^m \},$$

where $x_2 \in \mathbb{R}^n$, $s \in \mathbb{R}^m$.

**Definition 2.24.** [Subdifferential of saddle function] Let $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{-\infty, +\infty\}$ be a concave-convex function. We define $\partial_1 f(x_1, x_2)$ to be the set of all subgradients of the concave function $f(\cdot, x_2)$ at $x_1$, i.e.

$$\partial_1 f(x_1, x_2) = \{ s_1 : f(x_1', x_2) \leq f(x_1, x_2) + \langle s_1, x_1' - x_1 \rangle, \forall x_1' \in \mathbb{R}^m \},$$

where $x_1 \in \mathbb{R}^m$, $x_2 \in \mathbb{R}^n$ and $s_1 \in \mathbb{R}^m$. Similarly we define $\partial_2 f(x_1, x_2)$ to be the set of all subgradients of the convex function $f(x_1, \cdot)$ at $x_2$, i.e.

$$\partial_2 f(x_1, x_2) = \{ s_2 : f(x_1, x_2') \geq f(x_1, x_2) + \langle s_2, x_2' - x_2 \rangle, \forall x_2' \in \mathbb{R}^n \},$$

where $s_2 \in \mathbb{R}^n$. Then, the subdifferential of the saddle function $f$ at $(x_1, x_2)$ is defined as

$$\partial f(x_1, x_2) = \partial_1 f(x_1, x_2) \times \partial_2 f(x_1, x_2).$$
Chapter 2. Convex Analysis Preliminaries

Definition 2.25. [Fenchel conjugate (also named Legendre-Fenchel transform, Young-Fenchel transform, Legendre-Fenchel conjugate, or convex conjugate)] Let \( f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\} \) be a function. The Fenchel conjugate of \( f \) is defined by

\[
    f^*(s) = \sup_{x \in \mathbb{R}^n} \langle s, x \rangle - f(x),
\]

where \( s \in \mathbb{R}^n \).

Definition 2.26. [Moreau envelope (or Moreau-Yosida approximation)] Given \( \lambda > 0 \) the Moreau envelope of a function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\} \) is defined as

\[
    e_\lambda f(s) = \inf_{x \in \mathbb{R}^n} \left[ f(x) + \frac{1}{2\lambda} \| x - s \|^2 \right],
\]

where \( s \in \mathbb{R}^n \).

Proposition 2.27. ([Luc06, Proposition 3]) Let \( f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\} \) be a function and assume \( \lambda > 0 \). Then

\[
    e_\lambda f(s) = \frac{\| s \|^2}{2\lambda} - \frac{1}{\lambda} g_\lambda^*(s),
\]

where \( g_\lambda(x) = \frac{\| x \|^2}{2} + \lambda f(x) \).

Proof.

\[
    e_\lambda f(s) = \inf_{x \in \mathbb{R}^n} \left[ f(x) + \frac{1}{2\lambda} \| x - s \|^2 \right] \\
= \frac{\| s \|^2}{2\lambda} + \inf_{x \in \mathbb{R}^n} \left[ -\frac{1}{\lambda} \langle s, x \rangle + \frac{\| x \|^2}{2\lambda} + f(x) \right] \\
= \frac{\| s \|^2}{2\lambda} + \inf_{x \in \mathbb{R}^n} \left[ -\frac{1}{\lambda} \langle s, x \rangle + \frac{1}{\lambda} \left( \| x \|^2 \right) + f(x) \right] \\
= \frac{\| s \|^2}{2\lambda} + \inf_{x \in \mathbb{R}^n} \left[ -\frac{1}{\lambda} \langle s, x \rangle + g_\lambda^*(s) \right] \\
= \frac{\| s \|^2}{2\lambda} - \frac{1}{\lambda} \sup_{x \in \mathbb{R}^n} \langle s, x \rangle - g_\lambda(x) \\
= \frac{\| s \|^2}{2\lambda} - \frac{1}{\lambda} g_\lambda^*(s). \]
**Definition 2.28.** [Partial conjugate] Assume $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is a function of $n$ variables. The partial conjugate of $f$ is defined as taking the conjugate with respect to a single variable while the other variables remain constant i.e.

$$f^*(x_1, \ldots, x_{i-1}, s_i, x_{i+1}, \ldots, x_n) = \sup_{x_i \in \mathbb{R}} [s_i x_i - f(x_1, \ldots, x_n)].$$

**Fact 2.29.** ([RW98, Proposition 11.48]) Assume $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{+\infty\}$ is a proper convex function. Then,

$$(s_1, s_2) \in \partial f(x_1, x_2) \iff \begin{cases} x_1 \in \partial_1 f^*(s_1, x_2), \\ s_2 \in \partial_2 (-f^*(s_1, x_2)). \end{cases}$$

**Definition 2.30.** [Piecewise linear-quadratic (PLQ) function] Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a function. Then, $f$ is called a piecewise linear-quadratic (PLQ) function if $\text{dom } f$ can be represented as a union of finitely many polyhedral sets, relative to each of which $f(x)$ is defined by an expression of the form $\frac{1}{2} \langle x, Ax \rangle + \langle a, x \rangle + \sigma$ for some scalar $\sigma \in \mathbb{R}$, vector $a \in \mathbb{R}^n$ and symmetric matrix $A \in \mathbb{R}^{n \times n}$.

**Definition 2.31.** [Pieces of PLQ function] Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a PLQ function. Assume $\text{dom } f = \bigcup_{k=1}^K P_k$, where $P_k$ are polyhedral sets. We define $f$ on each $P_k$ by $f_k(x) = \frac{1}{2} \langle x, Ax \rangle + \langle a, x \rangle + \sigma$ so that

$$f(x) = \begin{cases} f_1(x), & \text{if } x \in P_1; \\ f_2(x), & \text{if } x \in P_1; \\ \vdots \\ f_k(x), & \text{if } x \in P_k. \end{cases}$$

Then the function $\overline{f_k} : P_k \to \mathbb{R} \cup \{+\infty\}$ is a piece of $f$ with

$$\overline{f_k}(x) = \begin{cases} f_k(x), & \text{if } x \in P_k; \\ +\infty, & \text{otherwise}. \end{cases}$$

**Definition 2.32.** [Lower semicontinuous (lsc) function] A proper function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is lower semicontinuous (lsc) at a point $\tilde{x} \in \text{dom } f$ if
for every $\epsilon > 0$ there exist a neighborhood $U$ of $\tilde{x}$ such that $f(x) \geq f(\tilde{x}) - \epsilon$ for all $x \in U$.

A function is lower semicontinuous if it is lower semicontinuous at every point of its domain.

**Fact 2.33. (Conjugate of convex PLQ functions[RW98, Theorem 11.14])** Assume $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is a proper lsc convex function. Then, $f$ is PLQ if and only if $f^*$ is PLQ.

**Definition 2.34. (Proximal average of two PLQ functions)** Let $f_1 : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ and $f_2 : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be two PLQ functions. The proximal average of $f_1$ and $f_2$ for any $\lambda \in [0, 1]$ is defined as

$$P(f_1, \lambda, f_2) = \left(1 - \lambda \right) \left(f_1 + \frac{1}{2} \|x\|^2 \right)^* + \lambda \left(f_2 + \frac{1}{2} \|x\|^2 \right)^* - \frac{1}{2} \|x\|^2.$$

**Definition 2.35. (Normal cone to convex set)** The normal cone to a convex set $C$ at $x \in C$ is defined by

$$N_C(x) = \{ z : \langle z, x - \bar{x} \rangle \leq 0 \text{ for all } x \in C \}.$$  

The normal cone $N_C(y) = \{0\}$ if $y \notin C$.

**Example 2.36. (Example of normal cones to a convex set)** Consider the rectangle as shown in Figure 2.1. The normal cone at the point $x$, $N_C(x) = \{(0, 0)\}$. For the point $y$, the normal cone is $N_C(y) = \{(x_1, x_2) : x_1 \leq 0, x_2 = 0\}$. The normal cone for the point $z$ is $N_C(z) = \{(x_1, x_2) : x_1 \leq 0, x_2 \leq 0\}$.

**Figure 2.1: Example of normal cones**

**Definition 2.37. (Inequality and equality parts of normal cone)** A normal cone to a polyhedral set $C = \{x \in \mathbb{R}^n : Ax = 0, Bx \leq 0\}$ at a point can be...
Chapter 2. Convex Analysis Preliminaries

defined as a set of polyhedral inequalities. Assume a normal cone \( N_C(x) \) to the polyhedral set \( C \) at a point \( x \in \mathbb{R}^n \) is defined as

\[
N_C(x) = \{ x \in \mathbb{R}^n : \begin{bmatrix} \mathcal{L}_E \\ \mathcal{L}_I \end{bmatrix} x \leq 0 \},
\]

with \( \mathcal{L}_E x = 0 \) and \( \mathcal{L}_I x \leq 0 \) where \( \mathcal{L}_E \) and \( \mathcal{L}_I \) are \( m_1 \times n \) and \( m_2 \times n \) matrices. Then \( \mathcal{L}_E \) is the equality part and \( \mathcal{L}_I \) is the inequality part of \( N_C(x) \).

**Definition 2.38.** (Active set [NW06, Definition 12.1], LICQ [NW06, Definition 12.4]) Let a set \( C \) be defined by

\[
C = \left\{ x \in \mathbb{R}^n : \begin{array}{l}
g_i(x) \leq 0 \text{ if } i \in I; \\
g_i(x) = 0 \text{ if } i \in E.
\end{array} \right\}
\]

The active set \( A(x) \) at any point \( x \in C \) is defined as

\[
A(x) = E \cup \{ i \in I : g_i(x) = 0 \}.
\]

The linear independence constraint qualification (LICQ) holds at \( x \) if the set of active constraint gradients \( \{ \nabla g_i(x), i \in A(x) \} \) is linearly independent.

**Fact 2.39.** ([NW06, Lemma 12.9]) Let a set \( C \) be defined by

\[
C = \left\{ x \in \mathbb{R}^n : \begin{array}{l}
g_i(x) \leq 0 \text{ if } i \in I; \\
g_i(x) = 0 \text{ if } i \in E.
\end{array} \right\},
\]

where \( g_i \in C^1 \) for \( i \in I \cup E \). Suppose that \( C \) is convex and \( \bar{x} \in C \). Assume LICQ holds at \( \bar{x} \), then

\[
N_C(\bar{x}) = \{ z : z = \sum_{i \in A(\bar{x})} \lambda_i \nabla g_i(\bar{x}), \text{with } \lambda_i \geq 0 \text{ for } i \in A(\bar{x}) \cap I \}.
\]

**Definition 2.40** (Degree of a vertex in a graph). Let \( G = (V, E) \) be a non-empty graph. The degree of a vertex \( v \in V \) is the total number of edges in \( v \). It is denoted by \( \text{deg}(v) \).

**Fact 2.41.** ([Die12, Page 5]) Let \( G = (V, E) \) be a non-empty graph. We have \( \sum_{v \in V} \text{deg}(v) = 2n_E \), where \( n_E \) is the total number of edges in \( G \).

**Fact 2.42.** (Euler’s formula [Ger07]) In a planar graph with \( n \) nodes, \( a \) arcs and \( r \) regions, we have \( n - a + r = 2 \).
Chapter 3

The Partial Conjugates

In this chapter we recall Goebel’s proof [Goe00, Proposition 3, page 17] that the partial conjugates of convex PLQ functions are also PLQ.

Definition 3.1. [Partial conjugate] Assume that $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{+\infty\}$ is a function of $m+n$ variables. The partial conjugate of $f$ with respect to a variable $x_1 \in \mathbb{R}^m$ is defined as

$$f^{*1}(s_1, x_2) = \sup_{x_1} [\langle s_1, x_1 \rangle - f(x_1, x_2)].$$

Similarly the partial conjugate of $f$ with respect to another variable $x_2 \in \mathbb{R}^n$ is defined by

$$f^{*2}(x_1, s_2) = \sup_{x_2} [\langle s_2, x_2 \rangle - f(x_1, x_2)].$$

Theorem 3.2. Let $f : \mathbb{R}^{m+n} \to \mathbb{R} \cup \{+\infty\}$ be a proper convex PLQ function of $m+n$ variables. Assume $f^{*1}$ is the partial conjugate of $f$ with respect to the first $m$ variables. Then $f^{*1}$ is PLQ.

The proof is found in [Goe00, Proposition 3, page 17]. For the sake of completeness, we reproduce it here using our notations.

Proof. Since $x_1 \mapsto f^{*2}(x_1, s_2)$ is concave and $s_2 \mapsto f^{*2}(x_1, s_2)$ is convex, $f^{*2}(x_1, s_2)$ is a saddle function. The convex parent of a saddle function $h(x, y)$ is defined as [Goe00, Formula 2.6, page 10]

$$C_p(p, y) = \sup_x [h(x, y) - \langle p, x \rangle].$$

By putting $p = -s_1$, $y = s_2$, $x = x_1$ and replacing $f^{*2}$ with $h$, we have

$$C_p(-s_1, s_2) = \sup_{x_1} [f^{*2}(x_1, s_2) + \langle s_1, x_1 \rangle].$$

13
Using the definition of $f^*$ we deduce that
\[
C_p(-s_1, s_2) = \sup_{x_1} \left[ \sup_{x_2} \{s_2x_2 - f(x_1, x_2)\} + s_1x_1 \right] \\
= \sup_{x_1, x_2} \left[ s_1x_1 + s_2x_2 - f(x_1, x_2) \right] \\
= f^*(s_1, s_2)
\]

Since $f$ is PLQ if, and only if, $f^*$ is PLQ [Fact 2.33], we deduce $C_p$ is PLQ.

Consequently, there exists polyhedral sets $P_k$ with $\text{dom } C_p = \bigcup_{k=1}^s P_k$ and for $(-s_1, s_2) \in P_k$, $C_p$ is defined by
\[
C_p(-s_1, s_2) = \frac{1}{2} \langle (-s_1, s_2), A_k(-s_1, s_2) \rangle + \langle a_k, (-s_1, s_2) \rangle + b_k.
\]

Since $C_p$ is PLQ, $\partial C_p$ is piecewise polyhedral [RW98, Proposition 12.30]. Note that $\text{gph } \partial C_p$ is piecewise polyhedral [RW98, Example 9.57]. For any $(-x_1, x_2) \in \partial C_p(-s_1, s_2)$, we have $(-s_1, s_2, -x_1, x_2) \in \text{gph } \partial C_p$.

Note $\text{gph } \partial C_p = \bigcup_{l=1}^s D_l$ where each $D_l$ is a polyhedral set such that the image of $D_l$ under the projection $(-s_1, s_2, -x_1, x_2) \mapsto (-s_1, s_2)$ is contained in some $P_k$. We have
\[
(-s_1, s_2, -x_1, x_2) \in \text{gph } \partial C_p \\
\iff (-x_1, x_2) \in \partial C_p(-s_1, s_2) \\
\iff \begin{cases} -s_1 \in \partial_1 C_{s_1}^*(x_1, s_2) \\ x_2 \in \partial_2(C_{s_1}^*)(-x_1, s_2) \end{cases} \quad \text{[Fact 2.29].}
\]

For any saddle function $h(x, y)$ on $\mathbb{R}^m \times \mathbb{R}^n$ which is concave in $x$ and convex in $y$, the subdifferential is defined in [Roc70, page 374] as $\partial_1 h(x, y) = \partial_y h(x, y)$. It is the set of all subgradients of the concave function $h(\cdot, y)$ at $x$, i.e. the set of all vectors $x^* \in \mathbb{R}^m$ such that
\[
h(x', y) \leq h(x, y) + \langle x^*, x' - x \rangle, \forall x' \in \mathbb{R}^m.
\]
Similarly $\partial_2 h(x, y) = \partial_y h(x, y)$ is the set of all subgradients of the convex function $h(x, \cdot)$ at $y$, i.e. the set of all vectors $y^* \in \mathbb{R}^n$ such that
\[
h(x, y') \geq h(x, y) + \langle y^*, y' - y \rangle, \forall y' \in \mathbb{R}^n.
\]
The elements $(x^*, y^*)$ of the set $\partial h(x, y) = \partial_1 h(x, y) \times \partial_2 h(x, y)$ are defined as the subgradients of $h$ at $(x, y)$.
Chapter 3. The Partial Conjugates

We have

\[-s_1 \in \partial_1 C_p^*(x_1, s_2)\]
\[\iff C_p^*(-x_1', s_2) \geq C_p^*(-x_1, s_2) + (-s_1, -x_1' - (-x_1)), \forall (-x_1')\]
\[\iff C_p^*(-x_1', s_2) \leq -C_p^*(-x_1, s_2) + (s_1, -x_1' - (-x_1)), \forall (-x_1')\]

and

\[x_2 \in \partial_2(-C_p^*)(-x_1, s_2)\]
\[\iff -C_p^*(-x_1, s_2') \geq -C_p^*(-x_1, s_2) + \langle x_2, s_2' - s_2 \rangle, \forall s_2'\]

Therefore, we obtain

\[(s_1, x_2) \in \partial(-C_p^*)(-x_1, s_2)\]
\[\iff (-x_1, s_2, s_1, x_2) \in \text{gph}(-C_p^*).\]

We note

\[M = \begin{bmatrix} 0 & 0 & -I & 0 \\ 0 & I & 0 & 0 \\ I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix},\]

where \(I\) is the \(m \times m\) or \(n \times n\) identity matrix. We also note that \(M\) is invertible with

\[M^{-1} = \begin{bmatrix} 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \\ -I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix}.\]

By applying \(M\) to both sides of \((-x_1, s_2, s_1, x_2) \in \text{gph}(-C_p^*)\), we obtain

\[M(-x_1, s_2, s_1, x_2) \in M \text{gph}(-C_p^*)\]
\[\iff (-s_1, s_2, -x_1, x_2) \in M \text{gph}(-C_p^*).\]

Therefore, we deduce

\[\text{gph} \partial C_p = M \text{gph} \partial(-C_p^*)\]
\[\iff \text{gph} \partial(-C_p^*) = M^{-1} \text{gph} \partial C_p.\]

Define the polyhedral set \(E_t = M^{-1}D_t\). We have \(\text{gph} \partial(-C_p^*) = \bigcup_{t=1}^\infty E_t\).

By applying \(M^{-1}\) to both sides of \((-s_1, s_2, -x_1, x_2) \in D_t\), we deduce

\[M^{-1}(-s_1, s_2, -x_1, x_2) \in M^{-1}D_t\]
\[\iff (-x_1, s_2, s_1, x_2) \in E_t.\]
Chapter 3. The Partial Conjugates

Let $F_l$ be the image of $E_l$ under the projection $(-x_1, s_2, s_1, x_2) \mapsto (-x_1, s_2)$. Now by [RW98, Proposition 3.55], the sets $E_l$ and $F_l$ are polyhedral, and $\text{dom} \partial(-C_p^{*1}) = \bigcup_{l=1}^{q} F_l$. Since $\text{dom} \partial(-C_p^{*1})$ is dense in $\text{dom}(-C_p^{*1})$ [Roc70, Theorem 37.4], we deduce that $\text{dom}(-C_p^{*1}) = \bigcup_{l=1}^{q} F_l$.

It remains to show that $-C_p^{*1}$ is linear or quadratic on each $F_l$. To prove that $-C_p^{*1}$ is linear or quadratic on $F_l$ it is necessary and sufficient to prove that $-C_p^{*1}$ is linear or quadratic relative to every line segment in $F_l$ [RW98, Lemma 11.15].

Pick any two points $(-x_1^0, s_2^0), (-x_1^1, s_2^1)$ in $F_l$ and any $(s_1^0, x_2^0) \in \partial(-C_p^{*1})(-x_1^0, s_2^0)$, with $(s_1^1, x_2^1) \in \partial(-C_p^{*1})(-x_1^1, s_2^1)$. We have

$$(-x_1^0, s_2^0, s_1^0, x_2^0), (-x_1^1, s_2^1, s_1^1, x_2^1) \in E_l.$$ 

Now we define for $\tau \in [0, 1]$,

$$(-x_1^\tau, s_2^\tau, s_1^\tau, x_2^\tau) = (1 - \tau)(-x_1^0, s_2^0, s_1^0, x_2^0) + \tau(-x_1^1, s_2^1, s_1^1, x_2^1).$$

Since polyhedral sets are convex, we deduce

$$(-x_1^\tau, s_2^\tau, s_1^\tau, x_2^\tau) \in E_l, \forall \tau \in [0, 1]. \quad (3.1)$$

We have

$$-C_p^{*1}(-x_1^\tau, s_2^\tau) = -\sup_{-s_1} [(-s_1, -x_1^\tau) - C_p(-s_1, s_2^\tau)]$$

$$= \inf_{-s_1} [C_p(-s_1, s_2^\tau) + (-s_1, x_1^\tau)].$$

Define $s_1'$ such that

$$-x_1^\tau \in \partial_1 C_p(-s_1', s_2^\tau)$$

$$\iff C_p(-s_1, s_2^\tau) \geq C_p(-s_1', s_2^\tau) + (-x_1^\tau, -s_1 + s_1'), \forall (-s_1)$$

$$\iff C_p(-s_1, s_2^\tau) + (-s_1, x_1^\tau) \geq C_p(-s_1', s_2^\tau) - (s_1', x_1^\tau), \forall (-s_1)$$

$$\iff \inf_{-s_1} [C_p(-s_1, s_2^\tau) + (-s_1, x_1^\tau)] \geq C_p(-s_1', s_2^\tau) - (s_1', x_1^\tau).$$

By definition

$$\inf_{-s_1} [C_p(-s_1, s_2^\tau) + (-s_1, x_1^\tau)] \leq C_p(-s_1', s_2^\tau) - (s_1', x_1^\tau).$$

Therefore, we deduce that

$$\inf_{-s_1} [C_p(-s_1, s_2^\tau) + (-s_1, x_1^\tau)] = C_p(-s_1', s_2^\tau) - (s_1', x_1^\tau)$$

16
Chapter 3. The Partial Conjugates

for all $s'_1$ such that $-x'_1 \in \partial_1 C_p(-s'_1, s'_2)$.

Since $(-x'_1, s'_2, s'_1, x'_2) \in E_l$, we have $(-x'_1, s'_2, s'_1, x'_2) \in \text{gph}(-C_p^{*1})$.

From $\text{gph} \partial C_p = M \text{gph} \partial (-C_p^{*1})$, we deduce

$$M(-x'_1, s'_2, s'_1, x'_2) \in \text{gph} \partial C_p$$
$$\iff (-s'_1, s'_2, -x'_1, x'_2) \in \text{gph} \partial C_p$$
$$\iff (-x'_1, x'_2) \in \partial C_p(-s'_1, s'_2)$$
$$\iff -x'_1 \in \partial_1 C_p(-s'_1, s'_2).$$

Consequently, we have

$$-C_p^{*1}(-x'_1, s'_2) = C_p(-s'_1, s'_2) + \langle -s'_1, x'_1 \rangle.$$

By applying $M$ to both sides of (3.1), we get

$$\forall \tau \in [0, 1], M(-x'_1, s'_2, s'_1, x'_2) \in ME_l$$
$$\iff \forall \tau \in [0, 1], (-s'_1, s'_2, -x'_1, x'_2) \in D_l$$
$$\implies \exists k, \forall \tau \in [0, 1], (-s'_1, s'_2) \in P_k.$$

Therefore, we have

$$-C_p^{*1}(-x'_1, s'_2) = \frac{1}{2} \langle (-s'_1, s'_2), A_k(-s'_1, s'_2) \rangle + \langle a_k, (-s'_1, s'_2) \rangle + b_k + \langle -s'_1, x'_1 \rangle.$$

The later expression is linear or quadratic in $\tau$, so the function $-C_p^{*1}(-x'_1, s'_2)$ is PLQ, which implies that $-C_p^{*1}$ is PLQ.

In summary, we proved that

$$f \text{ is PLQ} \implies (f^*)^{*1 \text{ is PLQ.}} \quad (3.2)$$

By Fact 2.33,

$$f \text{ is PLQ} \iff f^* \text{ is PLQ.} \quad (3.3)$$

From the previous two implications we derive,

$$f^* \text{ is PLQ} \implies (f^*)^{*1 \text{ is PLQ.}}$$

By applying the previous implication to $g = f^*$, we deduce

$$g \text{ is PLQ} \implies g^{*1 \text{ is PLQ,}}$$

which concludes the proof.
Chapter 4

Convex Parametric Quadratic Programming

In this chapter, we recall recent results on parametric minimization of convex piecewise quadratic functions [PS11], and adapt them to compute the conjugate.

4.1 Piecewise quadratic optimization problem

Definition 4.1. ([PS11, Definition 1]) Let \( \zeta = \{C_k : k \in K\} \) be a collection of nonempty sets where \( K \) is a finite index set.

1. If the union of all sets in \( \zeta \) is \( D \subseteq \mathbb{R}^d \) and the sets are mutually disjoint, then \( \zeta \) is a partition of \( D \).

2. If the members of \( \zeta \) are polyhedral sets and (i) \( \bigcup_{k \in K} C_k = D \), (ii) \( \dim C_k = \dim D \) for all \( k \in K \), (iii) \( \text{ri} \, C_k \cap \text{ri} \, C_l = \phi \), for \( k, l \in K, k \neq l \), then it is called a polyhedral decomposition of \( D \subseteq \mathbb{R}^d \).

3. If \( \zeta \) is a polyhedral decomposition and the intersection of any two members of \( \zeta \) is either empty or a common proper face of both, then it is called a polyhedral subdivision.

Example 4.2. [Polyhedral decomposition, polyhedral subdivision] Figure 4.1 illustrates a collection of polyhedral set \( \zeta_1 = \{D_1, D_2, D_3\} \) such that \( D_1 \cup D_2 \cup D_3 = D \). The collection \( \zeta_1 \) is a polyhedral decomposition of \( D \). But \( \zeta \) is not a polyhedral subdivision of \( D \) because the intersection of \( D_1 \) and \( D_3 \) is not a common face of both. Similarly the intersection of \( D_2 \) and \( D_3 \) is not a common face of both too.

Assume another collection of polyhedral sets \( \zeta_2 = \{S_1, S_2, S_3\} \) such that \( S_1 \cup S_2 \cup S_3 = S \). Figure 4.2 shows this collection and it is a subdivision of \( S \).

Definition 4.3. ([PS11, Definition 2]) A piecewise linear-quadratic function (PLQ) is a function \( f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\} \) such that there exists a polyhedral
4.1. Piecewise quadratic optimization problem

\[ \zeta = \{ C_k : k \in K \} \]

Figure 4.1: Polyhedral decomposition of \( D \)

Figure 4.2: Polyhedral subdivision of \( S \)

decomposition \( \zeta = \{ C_k : k \in K \} \) of \( \text{dom} \ f \), \( f(x) = f_k(x) \) if \( x \in C_k, k \in K \), where \( f_k(x) = (1/2)x^TQ_kx + q_k^T x + \sigma_k \) with \( \sigma_k \in \mathbb{R} \), \( q_k \in \mathbb{R}^n \) and \( Q_k \in \mathbb{R}^{n \times n} \) is a symmetric matrix.

One can always build a polyhedral decomposition from a polyhedral subdivision. Without loss of generality it is assumed that the dimension of \( \text{dom} \ f \) is \( n \). Each \( C_k, k \in K \) is defined by \( C_k = \{ x \in \mathbb{R}^n : A^k x \leq b^k \} \) with \( A^k \in \mathbb{R}^{m_k \times n} \).

**Definition 4.4.** ([PS11, Definition 3]) For any \( x \in \text{dom} \ f \) the index set \( \kappa(x) \) is defined by

\[ \kappa(x) = \{ k \in K : x \in C_k \}. \]

**Example 4.5.** Figure 4.3 illustrates the domain of a PLQ function \( f \) with \( \text{dom} \ f = \{ C_1, C_2, C_3, C_4, C_5 \} \) and a point \( x \in \text{dom} \ f \). Here, \( \kappa(x) = \{ C_2, C_3 \} \).

**Definition 4.6.** ([PS11, Page 5]) Consider a non-empty polyhedral set \( C = \{ x \in \mathbb{R}^n : Ax \leq b \} \) with \( b \in \mathbb{R}^m \). Let the collection of all non-empty polyhedral faces be denoted by \( \mathcal{F}(C) \) and the collection of the relative interiors of all non-empty faces of \( C \) be denoted by \( \mathcal{G}(c) \), i.e. \( \mathcal{G}(c) = \{ \text{ri} \mathcal{F} : \mathcal{F} \in \mathcal{F}(C) \} \).
4.1. Piecewise quadratic optimization problem

![Figure 4.3: The index set of $x$](image)

Also let $\mathcal{B}(C) = \{I \subseteq \{1, \ldots, m\} : \exists x \text{ s.t. } A_I(x) = b_I, A_{I^c}(x) < b_{I^c}\}$, where $I^c$ is the complement of $I$.

Consider the sets $\mathcal{F}_I = \{x \in \mathbb{R}^n : A_I(x) = b_I, A_{I^c}(x) \leq b_{I^c}\}$ and $\mathcal{G}_I = \{x \in \mathbb{R}^n : A_I(x) = b_I, A_{I^c}(x) < b_{I^c}\}$, for every index $I \in \mathcal{B}(C)$.

**Example 4.7.** For example, consider a polyhedral set which is defined by

$$C = \left\{ (x_1, x_2) \in \mathbb{R}^2 : \begin{array}{l}
x_1 \leq 1 \\
-x_2 \leq 1 \\
x_1 \leq 1 \\
x_2 \leq 1
\end{array} \right\} = [-1, 1]^2.$$

Therefore, $C = \{(x_1, x_2) \in \mathbb{R}^2 : A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq b\}$ where $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & 1 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$. Now the index set $\mathcal{B}(C) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{2, 3\}, \{3, 4\}, \{1, 4\}\}$.

For the index set $I_1 = \emptyset \in \mathcal{B}(C)$, we deduce $I_1^c = \{1, 2, 3, 4\}$. Now the set $\mathcal{F}_{I_1}$ is defined by

$$\mathcal{F}_{I_1} = \left\{ x \in \mathbb{R}^2 : \begin{array}{l}
x_1 \leq 1 \\
-x_2 \leq 1 \\
x_1 \leq 1 \\
x_2 \leq 1
\end{array} \right\}.$$
4.1. Piecewise quadratic optimization problem

The set $\mathcal{G}_{\mathcal{I}_1}$ is defined as

$$\mathcal{G}_{\mathcal{I}_1} = \left\{ x \in \mathbb{R}^2 \left| \begin{array}{l} x_1 < 1 \\ -x_2 < 1 \\ -x_1 < 1 \\ x_2 < 1 \end{array} \right. \right\}.$$

For any index set $\mathcal{I}_2 = \{1\} \in \mathcal{B}(C)$, $F_{\mathcal{I}_1}$ and $G_{\mathcal{I}_1}$ are defined as follows.

$$F_{\mathcal{I}_2} = \left\{ x \in \mathbb{R}^2 \left| \begin{array}{l} x_1 = 1 \\ -x_2 \leq 1 \\ -x_1 \leq 1 \\ x_2 \leq 1 \end{array} \right. \right\}.$$

$$G_{\mathcal{I}_2} = \left\{ x \in \mathbb{R}^2 \left| \begin{array}{l} x_1 = 1 \\ -x_2 < 1 \\ -x_1 < 1 \\ x_2 < 1 \end{array} \right. \right\}.$$

Similarly we define $F_{\mathcal{I}}$ and $G_{\mathcal{I}}$ for $\mathcal{I} \in \{\{2,\}, \{3,\}, \{4,\}\}$ too. For $\mathcal{I}_6 = \{1, 2\} \in \mathcal{B}(C)$, $F_{\mathcal{I}_6}$ and $G_{\mathcal{I}_6}$ is defined as follows.

$$F_{\mathcal{I}_6} = \left\{ x \in \mathbb{R}^2 \left| \begin{array}{l} x_1 = 1 \\ -x_2 = 1 \\ -x_1 \leq 1 \\ x_2 \leq 1 \end{array} \right. \right\}.$$

$$G_{\mathcal{I}_6} = \left\{ x \in \mathbb{R}^2 \left| \begin{array}{l} x_1 = 1 \\ -x_2 = 1 \\ -x_1 < 1 \\ x_2 < 1 \end{array} \right. \right\}.$$

The sets $F_{\mathcal{I}}$ and $G_{\mathcal{I}}$ for $\mathcal{I} \in \{\{2,3,\}, \{3,4,\}, \{1,4,\}\}$ are defined similarly.

**Fact 4.8.** ([PS11, Proposition 1]) Let $C = \{x \in \mathbb{R}^n | Ax \leq b\}$ be a polyhedral set. Then,

1. $\mathcal{G}(C)$ is a partition of $C$,

2. $\mathcal{F}(C) = \{F_{\mathcal{I}} | \mathcal{I} \in \mathcal{B}(C)\}$,

3. for every $\mathcal{I} \in \mathcal{B}(C), G_{\mathcal{I}} = riF_{\mathcal{I}}$,

4. for any $\mathcal{I}, \mathcal{I}' \in \mathcal{B}(C), \mathcal{I} \cap \mathcal{I}' \in \mathcal{B}(C)$,
5. for any $x_1, x_2 \in G$ and $I \in B(C)$, $N_C(x_1)$ and $N_C(x_2)$ are equal.

The notation $N_C(F_I)$ is used to denote the normal cone of $C$ at any point $x \in ri F_I$.

Consider again Example 4.7. For this example, $F(C) = \{F_1, F_2, \ldots, F_9\}$ and therefore, $G(C) = \{ri F_1, ri F_2, \ldots, ri F_9\}$. By using the previous fact, we deduce that $G_1 = ri F_1, G_2 = ri F_2, \ldots, G_9 = ri F_9$. The sets $G$ is a partition of $C$. Consider two index sets $I_6 = \{1, 2\}$ and $I_7 = \{2, 3\}$. We deduce $I_6 \cap I_7 = \{2\} \in B(C)$. Consider another index set $I_2 = \{1\} \in B(C)$. Take two points $x_1 = (1, 0.5)$ and $x_2 = (1, -0.5) \in G_{I_2}$. Then $N_C(x_1) = N_C(x_2)$.

**Definition 4.9.** ([PS11, Page 5]) Consider a polyhedral decomposition $\zeta = \{C_k : k \in K\}$ of $D \subseteq \mathbb{R}^d$. Assume $F(\zeta) = \{F(C_k)\}_{k \in K}$, $G(\zeta) = \{G(C_k)\}_{k \in K}$ and $B(\zeta) = \{B(C_k)\}_{k \in K}$ for any $x \in D$, $J(x)$ is defined by

$$J(x) = \{G \in G(\zeta) | x \in G\}.$$ 

Assume $\mathcal{Y}$ consists of all sets $J \subseteq G(\zeta)$ such that $J = J(x)$ for some $x \in D$. Let $G_J = \bigcap_{G \in J} G$ and $D_J = \text{cl} G_J$, for any $J \in \mathcal{Y}$.

**Fact 4.10.** ([PS11, Proposition 2]) Let $\zeta = \{C_k : k \in K\}$ be a polyhedral decomposition of $D \subseteq \mathbb{R}^d$. Then,

1. $\mathcal{G} = \{G_J : J \in \mathcal{Y}\}$ is a partition of $D$,

2. $\kappa(x_1) = \kappa(x_2)$, for any $x_1, x_2 \in G_J$ where $J \in \mathcal{Y}$,

3. for any $J \in \mathcal{Y}$, there exists a unique $I_k(J) \in B(C_k)$ where $k \in \kappa(J)$ such that $G_J = \bigcap_{k \in \kappa(J)} G_{I_k(J)}$ and $D_J = \bigcap_{k \in \kappa(J)} F_{I_k(J)}$.

**Example 4.11.** Consider a polyhedral decomposition $\zeta = \{C_1, C_2, C_3, C_4\}$ of $D$. Assume $C_1, C_2, C_3$ and $C_4$ are defined by

$$C_1 = \left\{ (x_1, x_2) \in \mathbb{R}^2 \left| \begin{array}{c} -x_1 \leq 0 \\ -x_2 \leq 0 \end{array} \right. \right\},$$

$$C_2 = \left\{ (x_1, x_2) \in \mathbb{R}^2 \left| \begin{array}{c} x_1 \leq 0 \\ -x_2 \leq 0 \end{array} \right. \right\},$$

$$C_3 = \left\{ (x_1, x_2) \in \mathbb{R}^2 \left| \begin{array}{c} x_1 \leq 0 \\ x_2 \leq 0 \end{array} \right. \right\},$$

$$C_4 = \left\{ (x_1, x_2) \in \mathbb{R}^2 \left| \begin{array}{c} x_1 \leq 0 \\ x_2 \leq 0 \end{array} \right. \right\}.$$
4.1. Piecewise quadratic optimization problem

\[ C_4 = \left\{ (x_1, x_2) \in \mathbb{R}^2 \left| \begin{array}{c}
-x_1 \leq 0 \\
x_2 \leq 0
\end{array} \right. \right\}. \]

Figure 4.4 illustrates this polyhedral decomposition. Now \( B(C_1), B(C_2), B(C_3) \) and \( B(C_4) \) are defined as

\[ B(C_1) = \{ \emptyset^1, \{1^1\}, \{2^1\}, \{1^1, 2^1\} \}, \]
\[ B(C_2) = \{ \emptyset^2, \{1^2\}, \{2^2\}, \{1^2, 2^2\} \}, \]
\[ B(C_3) = \{ \emptyset^3, \{1^3\}, \{2^3\}, \{1^3, 2^3\} \}, \]
\[ B(C_4) = \{ \emptyset^4, \{1^4\}, \{2^4\}, \{1^4, 2^4\} \}. \]

Here superscript 1, 2, 3, or 4 is used to denote whether we are considering \( C_1, C_2, C_3, \) or \( C_4 \). For example, for any \( I \in B(C_1) \), we use superscript 1 in each element of \( I \). Now consider the polyhedral set \( C_1 \). We could write \( F_{\emptyset^1} \) as

\[ F_{\emptyset^1} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 \leq 0, -x_2 \leq 0 \right\}. \]

We could compute \( G_{\emptyset^1} \) as

\[ G_{\emptyset^1} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 < 0, -x_2 < 0 \right\}. \]

Similarly we could write \( F_{\{1^1\}} \) as

\[ F_{\{1^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, -x_2 \leq 0 \right\}. \]

We could compute \( G_{\{1^1\}} \) as

\[ G_{\{1^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, -x_2 < 0 \right\}. \]

We could write \( F_{\{2^1\}} \) as

\[ F_{\{2^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 \leq 0, -x_2 = 0 \right\}. \]

We could compute \( G_{\{2^1\}} \) as

\[ G_{\{2^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 < 0, -x_2 = 0 \right\}. \]

The set \( F_{\{1^1, 2^1\}} \) could be computed as

\[ F_{\{1^1, 2^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, -x_2 = 0 \right\}. \]

The set \( G_{\{1^1, 2^1\}} \) is computed as

\[ G_{\{1^1, 2^1\}} = \left\{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, -x_2 = 0 \right\}. \]
4.1. Piecewise quadratic optimization problem

Therefore, the set $F(C_1)$ could be written as

$$F(C_1) = \{ F_{\emptyset^1}, F_{\{1\}^1}, F_{\{2\}^1}, F_{\{1,2\}^1} \},$$

and

$$G(C_1) = \{ G_{\emptyset^1}, G_{\{1\}^1}, G_{\{2\}^1}, G_{\{1,2\}^1} \}.$$ 

Now consider another polyhedral set $C_2$. For $C_2$ we could compute

$$F_{\emptyset^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 \leq 0, -x_2 \leq 0 \},$$

$$G_{\emptyset^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 < 0, -x_2 < 0 \},$$

$$F_{\{1\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, -x_2 \leq 0 \},$$

$$G_{\{1\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, -x_2 < 0 \},$$

$$F_{\{2\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 \leq 0, -x_2 = 0 \},$$

$$G_{\{2\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 < 0, -x_2 = 0 \},$$

$$F_{\{1,2\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, -x_2 = 0 \},$$

and

$$G_{\{1,2\}^2} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, -x_2 = 0 \}.$$ 

The set $F(C_2)$ could be computed as

$$F(C_2) = \{ F_{\emptyset^2}, F_{\{1\}^2}, F_{\{2\}^2}, F_{\{1,2\}^2} \},$$

and

$$G(C_2) = \{ G_{\emptyset^2}, G_{\{1\}^2}, G_{\{2\}^2}, G_{\{1,2\}^2} \}.$$ 

Consider the polyhedral set $C_3$. We have

$$F_{\emptyset^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 \leq 0, x_2 \leq 0 \},$$

$$G_{\emptyset^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 < 0, x_2 < 0 \},$$

$$F_{\{1\}^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 \leq 0 \},$$

$$F_{\{2\}^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 \leq 0, x_2 = 0 \},$$

$$F_{\{1,2\}^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 = 0 \},$$

and

$$G_{\{1,2\}^3} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 = 0 \}.$$
4.1. Piecewise quadratic optimization problem

\[ \mathcal{G}_{(1^3)} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 < 0 \}, \]

\[ \mathcal{F}_{(2^3)} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 \leq 0, x_2 = 0 \}, \]

\[ \mathcal{G}_{(2^3)} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 < 0, x_2 = 0 \}, \]

\[ \mathcal{F}_{(1^3, 2^3)} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 = 0 \}, \]

\[ \mathcal{G}_{(1^3, 2^3)} = \{ (x_1, x_2) \in \mathbb{R}^2 : x_1 = 0, x_2 = 0 \}, \]

\[ \mathcal{F}(C_3) = \{ \mathcal{F}_{\emptyset^3}, \mathcal{F}_{(1^3)}, \mathcal{F}_{(2^3)}, \mathcal{F}_{(1^3, 2^3)} \}, \]

and

\[ \mathcal{G}(C_3) = \{ \mathcal{G}_{\emptyset^3}, \mathcal{G}_{(1^3)}, \mathcal{G}_{(2^3)}, \mathcal{G}_{(1^3, 2^3)} \}. \]

Finally, consider the remaining polyhedral set \( C_4 \). We compute

\[ \mathcal{F}_{\emptyset^4} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 \leq 0, x_2 \leq 0 \}, \]

\[ \mathcal{G}_{\emptyset^4} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 < 0, x_2 < 0 \}, \]

\[ \mathcal{F}_{(1^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, x_2 \leq 0 \}, \]

\[ \mathcal{G}_{(1^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, x_2 < 0 \}, \]

\[ \mathcal{F}_{(2^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 \leq 0, x_2 = 0 \}, \]

\[ \mathcal{G}_{(2^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 < 0, x_2 = 0 \}, \]

\[ \mathcal{F}_{(1^4, 2^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, x_2 = 0 \}, \]

\[ \mathcal{G}_{(1^4, 2^4)} = \{ (x_1, x_2) \in \mathbb{R}^2 : -x_1 = 0, x_2 = 0 \}, \]

\[ \mathcal{F}(C_4) = \{ \mathcal{F}_{\emptyset^4}, \mathcal{F}_{(1^4)}, \mathcal{F}_{(2^4)}, \mathcal{F}_{(1^4, 2^4)} \}, \]
and
\[ S(C_4) = \{ G_{01}, G_{12}, G_{24}, G_{12,24} \}. \]

In this example, we have \( F_{\{11\}} = F_{\{12\}}, F_{\{21\}} = F_{\{24\}}, F_{\{22\}} = F_{\{23\}}, F_{\{13\}} = F_{\{14\}} \) and \( F_{\{11,21\}} = F_{\{12,22\}} = F_{\{13,23\}} = F_{\{14,24\}} \). We also have \( G_{\{11\}} = G_{\{12\}}, G_{\{21\}} = G_{\{24\}}, G_{\{22\}} = G_{\{23\}}, G_{\{13\}} = G_{\{14\}} \) and \( G_{\{12,23\}} = G_{\{12,24\}} = G_{\{13,24\}} = G_{\{13,21\}} \). Now we could write \( S(\zeta) \) as
\[ S(\zeta) = \{ G_{01}, G_{02}, G_{03}, G_{04}, F_{\{11\}}, F_{\{12\}}, F_{\{21\}}, F_{\{22\}}, F_{\{13\}} \}; \]
and \( S(\gamma) \) as
\[ S(\gamma) = \{ G_{01}, G_{02}, G_{03}, G_{04}, G_{\{11\}}, G_{\{21\}}, G_{\{22\}}, G_{\{13\}} \}, G_{\{12,21\}} \}. \]

Consider the point \( x_1 \) of Figure 4.4, we have \( S(x_1) = \{ G_{02} \} \). For the point \( x_2 \), we have \( S(x_2) = \{ G_{\{21\}} \} \). Now we could write \( S \) as
\[ S = \{ \{ G_{01} \}, \{ G_{02} \}, \{ G_{03} \}, \{ G_{04} \}, \{ G_{\{11\}} \}, \{ G_{\{21\}} \}, \{ G_{\{22\}} \}, \{ G_{\{13\}} \}, \{ G_{\{12,21\}} \} \}. \]

Assume \( J_1 = \{ G_{01} \}, J_2 = \{ G_{02} \}, J_3 = \{ G_{03} \}, J_4 = \{ G_{04} \}, J_5 = \{ G_{\{11\}} \}, J_6 = \{ G_{\{21\}} \}, J_7 = \{ G_{\{22\}} \}, J_8 = \{ G_{\{13\}} \}, J_9 = \{ G_{\{12,21\}} \} \). Figure 4.5 illustrates \( G_{J_1}, G_{J_2}, G_{J_3}, G_{J_4}, G_{J_5}, G_{J_6}, G_{J_7}, G_{J_8}, G_{J_9} \). The set \( Y \) is a partition of \( D \), where \( Y \) is defined by
\[ Y = \{ G_{J_1}, G_{J_2}, G_{J_3}, G_{J_4}, G_{J_5}, G_{J_6}, G_{J_7}, G_{J_8}, G_{J_9} \}. \]

Now consider Figure 4.6. It illustrates two points \( x_3, x_4 \in G_{J_6} \). We have \( \kappa(x_3) = \kappa(x_4) = \{ 1, 4 \} \). Consider again \( J_6 \in Y \). We have \( \kappa(J_6) = \{ 1, 4 \} \). There exist unique \( I_1(J_6) = \{ 2^4 \} \) and \( I_4(J_6) = \{ 2^4 \} \) so that \( G_{J_6} = G_{\{21\}} \cap G_{\{24\}} \) and \( D_{J_6} = F_{\{21\}} \cap F_{\{24\}} \). Figure 4.7 illustrates \( G_{J_6} \) and \( D_{J_6} \).

### 4.2 Parametric piecewise quadratic optimization problems

Define \( V(p) = \inf_x f(x, p) \) and \( X(p) = \arg\min_x f(x, p) \), where \( f : \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R} \cup \{ +\infty \} \) is a proper convex PLQ function s.t. \( f(x, p) = f_k(x, p) \) if
4.2. Parametric piecewise quadratic optimization problems

Figure 4.4: The polyhedral decomposition of \( D \) (x,p) \in C_k$. Here,

\[
f_k(x,p) = \frac{1}{2} \begin{bmatrix} x^T & \sigma_k \end{bmatrix} \begin{bmatrix} Q_k & R_k & S_k \\ R_k^T & p \end{bmatrix} \begin{bmatrix} x \\ p \end{bmatrix} + \begin{bmatrix} q_k^T \\ r_k \end{bmatrix} + \sigma_k,
\]

\( C_k = \{(x,p) | A_kx \leq B^k p + b^k \} \) and \( \zeta = \{ C_k | k \in \mathcal{K} \} \) is a polyhedral decomposition of dom \( f \).

**Fact 4.12.** ([PS11, Proposition 5]) Let \( f : \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\} \) be a proper convex PLQ function. Then,

1. \( V \) is also a proper convex PLQ function,
2. \( X \) is a polyhedral multifunction,
3. if \( f \) is strictly convex then \( X \) is single valued and piecewise affine.
4. dom \( X \) is defined as

\[
dom X = \{ p : \exists x, (x,p) \in \text{dom } f \} \cap \{ p : \exists \lambda_k, x_k \text{ s.t. } \nabla_x f_k(x_k,p) + A^k \lambda_k = 0, \forall k \in \mathcal{K} \}.
\]

4.2.1 Critical region

**Definition 4.13.** (Critical region [PS11]). For any index set \( \mathcal{J} \in \mathcal{Y} \), the critical region \( \mathcal{R}_\mathcal{J} \) is defined as

\[
\mathcal{R}_\mathcal{J} = \{ p : \exists x \in X(p) \text{ s.t. } \mathcal{J}(x,p) = \mathcal{J} \}.
\]
4.2. Parametric piecewise quadratic optimization problems

Figure 4.5: The partitions of $D$

**Fact 4.14.** ([PS11, Page 8]) For an index set $\mathcal{J} \in \mathcal{Y}$, we have

$$X_{\mathcal{J}}(p) = \left\{ x \mid (0,y) - \nabla f_k(x,p) \in N_C_k(F_{I_k}(\mathcal{J})), k \in \kappa(\mathcal{J}) \right\}$$

and

$$X_{\mathcal{J}}(p) = \left\{ x \mid (0,y) - \nabla f_k(x,p) \in N_C_k(F_{I_k}(\mathcal{J})), k \in \kappa(\mathcal{J}) \right\}.$$

**Fact 4.15.** ([PS11, Theorem 1(b)]) Assume $\overline{\mathcal{R}}_{\mathcal{J}} = \text{cl} \mathcal{R}_{\mathcal{J}}$. The closure of the critical region of the parametric quadratic program $\inf_x \{ f_k(x,p) : (x,p) \in C_k \}$ corresponding to the equality set $\mathcal{I}_k(\mathcal{J}) \in \mathcal{B}(C_k)$, $k \in \kappa(\mathcal{J})$ is defined as

$$\overline{\mathcal{R}}_{\mathcal{I}_k(\mathcal{J})} = \left\{ p \mid \exists (x, \lambda_k), \nabla_x f_k(x,p) + A^T_{\mathcal{I}_k(\mathcal{J})} \lambda_k = 0, \lambda_k \geq 0, \begin{align*}
A^T_{\mathcal{I}_k(\mathcal{J})} x &= B^T_{\mathcal{I}_k(\mathcal{J})} p + b^T_{\mathcal{I}_k(\mathcal{J})} \\
A^T_{\mathcal{I}_k(\mathcal{J})} x &\leq B^T_{\mathcal{I}_k(\mathcal{J})} p + b^T_{\mathcal{I}_k(\mathcal{J})} \end{align*} \right\}. \quad (4.2)$$

28
4.2. Parametric piecewise quadratic optimization problems

For an index set $\mathcal{J} \in \mathcal{Y}$, $\mathcal{R}_J$ is defined by

$$\mathcal{R}_J = \bigcap_{k \in \kappa(J)} \mathcal{R}_{I_k(J)}.$$  

4.2.2 Calculation of the solution

For strictly convex functions, the solution is single valued and affine inside the closure of the critical region $\mathcal{R}_J$. This solution could be calculated by solving the following quadratic program (QP)

$$\min_{x} \left\{ f_k(x, p) : A_{I_k(J)}^k x = B_{I_k(J)}^k p + b_{I_k(J)}^k \right\},$$

for any $k \in \kappa(J)$. If the linear independence constraint qualification (LICQ) holds for $k \in \kappa(J)$, then we could calculate the Lagrange multipliers from
4.2. Parametric piecewise quadratic optimization problems

\[ \nabla_x f_k(x, p) + A_k^T \lambda_k = 0, \] and consequently we obtain \( R_{I_k(J)} \). Then, we need to calculate the intersection of \( R_{I_k(J)} \) for all \( k \in \kappa(J) \) to compute the closure of the critical region \( R_J \). Computing the Moreau envelope is an example of strictly convex pQP.

For merely convex function i.e. the function is not strictly convex, calculation of the critical regions could be done using the normal cone optimality condition. The single valued and piecewise expression of the solution is valid in each of these critical regions and could be computed by solving a strictly convex parametric quadratic program. The rest of the chapter will focus on these computations.

4.2.3 Convex parametric quadratic program (pQP)

Consider the optimization problem defined by

\[ \nabla(p) = \min_{x \in \mathbb{R}^n} f(x, p), \text{ s.t. } Ax \leq Bp + b, \quad (4.3) \]

where

\[ f(x, p) = \frac{1}{2} x^T Q x + p^T R^T x + q, \]

with \( p \in \mathbb{R}^d \). The vector \( x \in \mathbb{R}^n \) is to be optimized for all values of \( p \in P \), where \( P \subseteq \mathbb{R}^d \) is a polyhedral set such that the minimum in (4.3) exists. Here \( R \in \mathbb{R}^{n \times d} \), \( A \in \mathbb{R}^{q \times n} \), \( b \in \mathbb{R}^{q \times 1} \), \( B \in \mathbb{R}^{q \times d} \) and \( Q \in \mathbb{R}^{n \times n} \) is a symmetric matrix. The parametric quadratic program is strictly convex if \( Q \) is positive definite. Otherwise it is convex when \( Q \geq 0 \). For \( Q = 0 \) we get a special type of parametric program, that is a parametric linear program (pLP). It is assumed that \( P \) is a full dimensional set.

**Definition 4.16.** [Feasible set, optimal set] For a given \( p \), the feasible set of the optimization problem (4.3) is defined as

\[ X(p) = \{ x \in \mathbb{R}^n : Ax \leq Bp + b \}. \]

The optimal set of points of (4.3), for any given \( p \), is defined by

\[ X^*(p) = \{ x \in \mathbb{R}^n : Ax \leq Bp + b, f(x, p) = V^*(p) \}, \]

**Definition 4.17.** [Active constraints, inactive constraints, active set, optimal active set] From a given \( p \), assume \( x \) is a feasible point of (4.3) . The active constraints are the constraints that satisfy \( A_i x = B_i p + b_i \). Inactive constraints satisfy \( A_i x < B_i p + b_i \).
4.2. Parametric piecewise quadratic optimization problems

The active set \( \mathcal{A}(x,p) \) is defined as the set of indices of the active constraints, i.e.,

\[
\mathcal{A}(x,p) = \{ i \in 1, 2, \ldots, q : A_ix = B_ip + b_i \}. 
\]

The optimal active set \( \mathcal{A}^*(p) \) is defined by

\[
\mathcal{A}^*(p) = \{ i \in 1, 2, \ldots, q : i \in \mathcal{A}(x,p), \forall x \in X^*(p) \} = \bigcap_{x \in \mathcal{X}^*(p)} \mathcal{A}(x,p).
\]

It is the set of indices of the constraints that are active for all \( x \in X^*(p) \).

**Definition 4.18.** [Critical region of a convex parametric quadratic program]

For a given index set \( \mathcal{A} \subseteq \{1, 2, \ldots, q\} \), the critical region of (4.3) associated with \( \mathcal{A} \) is defined as

\[
P^\mathcal{A} = \{ p \in P : \mathcal{A}^*(p) = \mathcal{A} \}.
\]

This is the set of parameters for which the optimal active set is equal to \( \mathcal{A} \).

**Fact 4.19.** ([STJ07, Theorem 2.1]) Consider the pQP defined in (4.3).

1. There exists a piecewise affine minimizer function \( \varphi : P \to \mathbb{R}^n \) where \( p \mapsto \varphi(p) \in X^*(p) \) and it is defined on a finite set of full dimensional polyhedral sets \( \mathcal{R} = \{ R^1, R^2, \ldots, R^K \} \) with \( P = \bigcup_{k=1}^K R^k \), \( \text{Int} R^i \cap \text{Int} R^j = \emptyset \) for all \( i \neq j \) and \( \varphi(p) \) defined on \( R^k \) is affine for all \( k \in K \).

2. The function \( \overline{V} : P \to \mathbb{R} \) is continuous and PLQ such that it is linear or quadratic on the polyhedral set \( R_k \) for all \( k \in \{1, 2, \ldots, K\} \).

**Fact 4.20.** (Normal cone optimality condition [RW98, Theorem 6.12]) Let \( f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\} \) be a function defined on a closed convex set \( \Omega \subseteq \mathbb{R}^n \). If \( \varphi \) is a local minimizer of \( f \) in \( \Omega \), then

\[
-\nabla f(\varphi) \in N_\Omega(\varphi).
\]

If \( f \) is convex, then \( \varphi \) is a global minimizer.

Assume \( \Omega \) is a polyhedral set which is defined by

\[
\Omega = \{ x : Ax \leq b \},
\]
4.2. Parametric piecewise quadratic optimization problems

where $A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix}$ and $b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$. The normal cone $N_{\Omega}(\bar{x})$ to the polyhedral set $\Omega$ at any point $\bar{x} \in \Omega$ is defined by

$$N_{\Omega}(\bar{x}) = \{y : L_I y \leq 0, L_E y = 0\},$$

where $L_I, L_E$ are matrices representing the inequality and equality parts of the normal cone. We assume $A = \{i : A_i \bar{x} = b_i\}$. Therefore, for any $i \notin A$ we have $A_i \bar{x} < b_i$. The notation $C^\Omega(A)$ is introduced as follow

$$C^\Omega(A) = \{y : L_I y \leq 0, L_E y = 0\} = N_{\Omega}(\bar{x}).$$

By using Fact 4.20, we deduce $-\nabla_x f(\bar{x}) \in \{y : L_I y \leq 0, L_E y = 0\}$. Thus, we have

$$L_I \nabla_x f(\bar{x}) \geq 0, L_E \nabla_x f(\bar{x}) = 0.$$ 

Now we define the normal cone optimality condition explicitly for (4.3) as

$$L_I^A(Qx + Rp + q) \geq 0 \text{ and } L_E^A(Qx + Rp + q) = 0,$$

where $L_I^A$, $L_E^A$ matrixes represent the inequality and equality parts of the normal cone matrix associated with $A$.

**Fact 4.21.** ([STJ07, Lemma 4.1]) Consider the optimization problem (4.3) for a given $p$. Let $A^*(p)$ be the optimal active set and $N_{X(p)}(A^*(p))$ be the corresponding normal cone. Then, we have

$$-(Qx^* + Rp + q) \in N_{X(p)}(A^*(p)),$$

for any optimal solution $x^* \in X^*(p)$.

**Fact 4.22.** ([STJ07, Lemma 4.2]) Consider the optimization problem (4.3). For any optimal active set $A$, there exists an associated optimal set mapping $\bar{X}^A(p) : P^A \to 2^{\mathbb{R}^n}$ which is defined by

$$\bar{X}^A(p) = \{x \in \mathbb{R}^n : L_E^A(Qx + Rp + q) = 0, L_I^A(Qx + Rp + q) \geq 0, A_\mathcal{N}x = B_A p + b_A, A_{\mathcal{N}} x \leq B_{\mathcal{N}} p + b_{\mathcal{N}}\},$$

where $\mathcal{N} = \{1, 2, \ldots, q\} - A$, $L_E^A$ and $L_I^A$ are the normal cone equality and inequality matrices associated with the optimal active set $A$. 

32
We assume $X^A : \text{cl} P^A \to 2^{\mathbb{R}^n}$ denotes the extension of the mapping $X^A(p)$ that is defined on the closure of $P^A$.

**Fact 4.23.** ([STJ07, Lemma 4.3]) The minimizer function $y^* : \text{cl}(P^A) \to \mathbb{R}^n$ of the following strictly convex $pQP$ is unique, continuous and piecewise affine.

$$\min_{y \in \mathbb{R}^n} \frac{1}{2} y^T y, \text{ where } y \in X^A(p).$$ \hspace{1cm} (4.4)

We could write (4.4) as

$$\min_{y \in \mathbb{R}^n} \frac{1}{2} y^T y, \text{ where } \tilde{A}y \leq \tilde{B}p + \tilde{b}, p \in \text{cl}(P^A).$$ \hspace{1cm} (4.5)

We let $t$ denote the number of constraints in (4.5). The parametric quadratic program defined in (4.5) is a function of the optimal active set $A$ of (4.3). Consequently the optimizer $y^*(p)$ and the optimal active set for (4.5) are functions of $A$. We use $B^*(p)$ to denote the optimal active set for (4.5).

**Fact 4.24.** ([STJ07, Lemma 4.4]) Given the optimal active sets $A$ for (4.3) and $B$ for (4.5), assume $\tilde{L}_E^B$ and $\tilde{L}_I^B$ are the equality and inequality part of the normal cone to $\{y \in \mathbb{R}^n : \tilde{A}y \leq \tilde{B}p + \tilde{b}\}$ defined by $B$. Assume the function $X^*(p)$ is continuous on $P$. Let the function $y^{A,B}(\cdot)$ be the unique solution to the systems of equations

$$\tilde{L}_E^B y = 0, \tilde{A}y = \tilde{B}p + \tilde{b}. \hspace{1cm} (4.6)$$

Then $y^{A,B}(\cdot)$ is optimal for (4.3) and (4.5) when it is restricted to

$$R^{A,B} = \text{cl}\{p \in P : A = A^*(p), B = B^*(p)\} = \{p \in P : \tilde{A}_N y^{A,B}(p) \leq \tilde{B}_N p + \tilde{b}_N, \tilde{L}_I^B y^{A,B}(p) \geq 0\},$$

where $N = \{1, 2, \ldots, t\} \setminus B$.

**Definition 4.25.** ([STJ07]) We define the mapping $\pi : P \to \mathbb{R}^n$ in Fact 4.19 as

$$\pi(p) = y^{A,B}(p), \text{ if } p \in R^{A,B}.\hspace{1cm}$$

The set of polyhedral sets on which $\pi(\cdot)$ is defined is represented by

$$\mathcal{R} = \{R^{A,B} : \dim(P \cap R^{A,B}) = s\}.$$
4.3 Adaptation of pQP in computational convex analysis

We adapt pQP for computing the conjugate of bivariate PLQ functions.

**Definition 4.26.** (parametric quadratic optimization problem) We define $V(p) = \inf_x \psi(x, p)$ and $X(p) = \arg\min_x \psi(x, p)$, where $\psi : \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$ is a proper PLQ function s.t. $\psi(x, p) = \psi_k(x, p)$ if $(x, p) \in C_k$. Here,

$$\psi_k(x, p) = \frac{1}{2} x^T Q_k x + \frac{1}{2} p^T R_k p + \frac{1}{2} p^T R_k x + \frac{1}{2} p^T R_k x + q_k^T x + r_k^T p + \alpha_k,$$

$C_k = \{(x, p) : \mathbf{A}^k x \leq \mathbf{b}^k p + \mathbf{b}^k\}$ and $\zeta = \{C_k : k \in K\}$. The set $\zeta$ is a polyhedral decomposition of $\text{dom}\psi$. Now we could write $\psi_k(x, p)$ as

$$\psi_k(x, p) = \frac{1}{2} x^T Q_k x + \frac{1}{2} p^T S_k p + \frac{1}{2} p^T R_k x + \frac{1}{2} p^T R_k x + q_k^T x + r_k^T p + \alpha_k.$$

Let $f : \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$ be a convex bivariate PLQ function such that there exists a polyhedral decomposition $P = \{C_k : k \in K\}$ of $\text{dom} f$, $f(x) = f_k(x)$ if $x \in C_k, k \in K$, where $f_k(x) = (1/2)x^T Q_k x + q_k^T x + \sigma_k$ with $\sigma_k \in \mathbb{R}, q_k \in \mathbb{R}^2$ and $Q_k \in \mathbb{R}^{2 \times 2}$ is a symmetric matrix. Assume for any $k \in K, C_k = \{x \in \mathbb{R}^n : \mathbf{A}^k x \leq \mathbf{b}^k\}$.

### 4.3.1 Conjugate computation

**Modeling**

Define a function $g : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$ so that $g(x, s) = g_k(x, s)$ when $x \in C_k$ and $g_k(x, s)$ is defined by

$$g_k(x, s) = f_k(x) - \langle s, x \rangle.$$

Now the conjugate of $f$ could be written as

$$f^*(s) = \sup_x [-g(x, s)]$$

$$= -\inf_x [g(x, s)].$$

Assume $V(s) = \inf_x [g(x, s)]$. Therefore, $f^*(s) = -V(s)$. Note that $g$ is not jointly convex. So we could not apply convex pQP results directly.

To describe the modelling more precisely, we note $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$. We also assume that $f_k(x_1, x_2) = a_k x_1^2 + b_k x_2^2 + c_k x_1 x_2 + d_k x_1 + e_k x_2 + \sigma_k$. Some text is repeated here.
4.3. Adaptation of pQP in computational convex analysis

for all \( k \in K \). Therefore, \( g_k(x_1, x_2; s_1, s_2) = f_k(x_1, x_2) - s_1 x_1 - s_2 x_2 \). Now we define \( g_k \) for any \( k \in K \) as

\[
g_k(x_1, x_2; s_1, s_2) = \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2a_k & c \\ c & 2b_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}^T \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + d^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \sigma_k.
\]

Then \( C_k \) could be expressed as

\[
C_k = \left\{ (x_1, x_2, s_1, s_2) : A_k \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq B_k \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + b_k \right\}.
\]

If we look at the decomposition of \( \text{dom } g \), we see that we have used the matrix \( B_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \) in (4.7) to define each \( C_k \) from a set of \( (x_1, x_2) \) to \( (x_1, x_2, s_1, s_2) \). Thus, in our problem model, considering the index sets for \( \text{dom } g \) means considering the index sets for \( \text{dom } f \). Therefore, for any index set \( J \in \mathcal{Y} \), the definition of the critical region which is defined in (4.13) becomes \( \mathcal{R}_J = \{ s : \exists x \in X(s) \text{ s.t. } J(x) = J \} \).

Note that \( g_k(x, s) \) is not convex. We have \( g_k(x, s) = f_k(x) - \langle s, x \rangle \). We define \( \tilde{f}_k(x) = f_k(x) + I_{C_k}(x) \). We assume \( \tilde{g}_k(x, s) = f_k(x) + I_{C_k}(x) - \langle s, x \rangle \) so that \( g(x, s) = \tilde{g}_k(x, s) \) when \( x \in C_k \). The conjugate of \( f \) could be written as

\[
f^*(s) = -\inf_x g(x, s).
\]

Now we write

\[
V(s) = \inf_x g(x, s), \quad X(s) = \arg\min_x g(x, s).
\]
4.3. Adaptation of pQP in computational convex analysis

**Theorem 4.27.** For any index set $\mathcal{J} \in \mathcal{Y}$ of the optimization problem defined in (4.8), the critical region could be written as

$$
\mathcal{R}_\mathcal{J} = \bigcap_{k \in \kappa(\mathcal{J})} \left\{ s \left| \begin{array}{c}
\exists (x, \lambda_k), 0 = \nabla_x g_k(x, s) + A_{I_k(\mathcal{J})}^k \lambda_k, \lambda_k \geq 0 \\
A_{I_k(\mathcal{J})}^k x = b_{I_k(\mathcal{J})}^k \\
A_{I_k(\mathcal{J})}^k x < b_{I_k(\mathcal{J})}^k
\end{array} \right\}.
\right.
$$

**Proof.** For any index set $\mathcal{J} \in \mathcal{Y}$, the optimality condition for the problem defined in (4.8) is

$$
0 \in \nabla_x f_k(x) + \partial I_{C_k}(x) - s, \forall k \in \kappa(\mathcal{J}). \quad (4.9)
$$

We have

$$
\partial I_{C_k}(x) = \{ v : I_{C_k}(x) \geq I_{C_k}(x) + \langle v, \bar{x} - x \rangle, \forall \bar{x} \in C_k \}.
$$

Therefore, for any $x \in C_k$ we have

$$
\partial I_{C_k}(x) = \{ v : I_{C_k}(x) \geq \langle v, \bar{x} - x \rangle, \forall \bar{x} \in C_k \}
= \{ v : 0 \geq \langle v, \bar{x} - x \rangle, \forall \bar{x} \in C_k \}
= N_{C_k}(x).
$$

Therefore the optimality condition in (4.9) becomes

$$
0 \in \nabla_x g_k(x, s) + N_{C_k}(x), \forall k \in \kappa(\mathcal{J}). \quad (4.10)
$$

We have

$$
\mathcal{R}_\mathcal{J} = \{ s : \exists x \in X(s) \text{ s.t. } \mathcal{J}(x) = \mathcal{J} \}. \quad \text{From Definition 4.6 we know that, } \mathcal{J}(x) = \mathcal{J} \text{ if and only if } x \in G_\mathcal{J}. \quad \text{Moreover, for any } x \in G_\mathcal{J} \text{ we have } x \in X(s) \text{ if and only if (4.10) holds. Therefore, we have}
$$

$$
\mathcal{R}_\mathcal{J} = \left\{ s \left| \exists x, \text{ s.t. } x \in G_\mathcal{J}, \right. \right. \left. \left. 0 \in \nabla_x g_k(x, s) + N_{C_k}(x), k \in \kappa(\mathcal{J}) \right\}.
\right.
$$

Since $G_\mathcal{J} = \bigcap_{k \in \kappa(\mathcal{J})} G_{I_k(\mathcal{J})}$ [Definition 4.9], for any $x \in G_\mathcal{J}$ we have $x \in G_{I_k(\mathcal{J})}$ for all $k \in \kappa(\mathcal{J})$. We know that $G_{I_k(\mathcal{J})} = \text{ri} F_{I_k(\mathcal{J})}$. For any $x \in \text{ri} F_{I_k(\mathcal{J})}$, we have $N_C(x) = N_C(F_{I_k(\mathcal{J}))}$ [Fact 4.8]. Thus, we have

$$
\mathcal{R}_\mathcal{J} = \left\{ s \left| \exists x, \text{ s.t. } x \in G_\mathcal{J}, \right. \right. \left. \left. 0 \in \nabla_x g_k(x, s) + N_C(F_{I_k(\mathcal{J)}), k \in \kappa(\mathcal{J}) \right\}.
\right.
$$

Since $G_\mathcal{J} = \bigcap_{k \in \kappa(\mathcal{J})} \text{ri} F_{I_k(\mathcal{J})}$, we deduce

$$
\mathcal{R}_\mathcal{J} = \bigcap_{k \in \kappa(\mathcal{J})} \left\{ s \left| \exists x, \text{ s.t. } x \in \text{ri} F_{I_k(\mathcal{J)}}, \right. \right. \left. \left. 0 \in \nabla_x g_k(x, s) + N_C(F_{I_k(\mathcal{J)})) \right\}.
\right.$$
4.3. Adaptation of pQP in computational convex analysis

For any $x \in \text{ri} \mathcal{F}_{I_k}(J) = \mathcal{G}_{I_k}(J)$, we have $A^k_{I_k}(J)x = b^k_{I_k}(J)$ and $A^k_{\tilde{I}_k}(J)x < b^k_{\tilde{I}_k}(J)$. Now by using Fact 2.39, we obtain

$$N_C(\mathcal{F}_{I_k}(J)) = A^k_{I_k}(J)^T \lambda_k,$$
with $\lambda_k \geq 0$.

Therefore, we deduce

$$R_J = \bigcap_{k \in \kappa(J)} \left\{ s \mid \exists (x, \lambda_k), 0 \in \nabla x g_k(x, s) + A^k_{I_k}(J)^T \lambda_k, \lambda_k \geq 0 \right\}.$$

\[ \square \]

Computation of the full dimensional critical regions

We assume that dom $f$ is defined on a polyhedral subdivision. After modeling, we go through each of the index sets for dom $g$ to compute the full dimensional i.e., in our case two dimensional critical regions and associated value functions. We explain this computation by using an example. Consider again Example 4.11. Assume a PLQ function $f$ is defined on the polyhedral decomposition $\zeta = \{ C_1, C_2, C_3, C_4 \}$ of $D$. Note that $\zeta$ is also a subdivision.

The index sets are $J_1, J_2, \ldots, J_9$.

We first assume $g(x, s)$ is a strictly convex function with respect to $x$. Consider an index set $J \in \mathcal{Y}$. For all $k \in \kappa(J)$, we need to solve the following equations for computing the values of $x$ and $\lambda_k$.

$$\nabla x g_k(x, s) + A^k_{I_k}(J)^T \lambda_k = 0 \quad (4.11)$$

$$A^k_{I_k}(J)x = b^k_{I_k}(J) \quad (4.12)$$

Assume the solution gives us the function $X_{I_k}(J)(s)$ and $\lambda_{I_k}(J)(s)$. Using this solution we get $\overline{R}_{I_k}(J)$ by using (4.2) as

$$\overline{R}_{I_k}(J) = \left\{ s \in \mathbb{R}^2 \mid \begin{array}{c} \lambda_{I_k}(J)(s) \geq 0 \\ A^k_{\tilde{I}_k}(J)X_{I_k}(J)(s) \leq b^k_{\tilde{I}_k}(J) \end{array} \right\}.$$ 

Consequently, we calculate the closure of the critical region which is defined as

$$\overline{R}_J = \bigcap_{k \in \kappa(J)} \overline{R}_{I_k}(J).$$
4.3. Adaptation of pQP in computational convex analysis

By using \( x = X_{I_k(J)}(s) \) in \( g_k(x, s) \) for any \( k \in \kappa(J) \), we get the value function \( V_J(s) \) which is defined on \( \overline{R}_J \).

For example consider the index set \( J_1 \) in Example 4.11. Now, \( \kappa(J_1) = \{1\} \). We need to solve the equations (4.11) and (4.12) for \( k = 1 \) to get \( X_{I_1(J)}(s) \) and \( \lambda_{I_1(J)}(s) \). Consequently we calculate \( \overline{R}_{I_1(J)} \). Then, we compute \( \overline{R}_{J_1} \) as

\[
\overline{R}_{J_1} = \bigcap_{k \in \{1\}} \overline{R}_{I_k(J_1)} = \overline{R}_{I_1(J_1)}.
\]

By substituting \( x = X_{I_1(J_1)}(s) \) in \( g_1(x, s) \), we get the value function \( V_{J_1}(s) \) which is defined on \( \overline{R}_{J_1} \). Similarly we compute \( \overline{R}_{J_2}, V_{J_2}(s), \overline{R}_{J_3}, V_{J_3}(s), \overline{R}_{J_4}, V_{J_4}(s) \).

Consider the index set \( J_5 \) in Example 4.11. Now, \( \kappa(J_5) = \{1, 2\} \). We need to solve equations (4.11) and (4.12) for \( k = 1 \) to compute \( X_{I_1(J_5)}(s) \) and \( \lambda_{I_1(J_5)}(s) \). Using \( X_{I_1(J_5)}(s) \) and \( \lambda_{I_1(J_5)}(s) \) we deduce \( \overline{R}_{I_1(J_5)} \). Similarly we compute \( \overline{R}_{I_2(J_5)} \). Then, the computation of \( \overline{R}_{J_5} \) is as follows

\[
\overline{R}_{J_5} = \bigcap_{k \in \{1, 2\}} \overline{R}_{I_k(J_5)} = \overline{R}_{I_1(J_5)} \cap \overline{R}_{I_2(J_5)}.
\]

By substituting \( x = X_{I_1(J_5)}(s) \) in \( g_1(x, s) \) or \( x = X_{I_2(J_5)}(s) \) in \( g_2(x, s) \), we deduce the value function \( V_{J_5}(s) \) which is defined on \( \overline{R}_{J_5} \). Similarly we compute \( \overline{R}_{J_6}, V_{J_6}(s), \overline{R}_{J_7}, V_{J_7}(s), \overline{R}_{J_8}, V_{J_8}(s) \).

Now consider the index set \( J_9 \) in Example 4.11. We have \( \kappa(J_9) = \{1, 2, 3, 4\} \). Therefore, we need to solve equations (4.11) and (4.12) for \( k \in \{1, 2, 3, 4\} \) to compute \( X_{I_1(J_9)}(s), \lambda_{I_1(J_9)}(s), X_{I_2(J_9)}(s), \lambda_{I_2(J_9)}(s), X_{I_3(J_9)}(s), \lambda_{I_3(J_9)}(s), X_{I_4(J_9)}(s), \lambda_{I_4(J_9)}(s) \). Consequently, we get \( \overline{R}_{I_1(J_9)}, \overline{R}_{I_2(J_9)}, \overline{R}_{I_3(J_9)}, \overline{R}_{I_4(J_9)} \). Therefore, we deduce \( \overline{R}_{J_9} \) as,

\[
\overline{R}_{J_9} = \overline{R}_{I_1(J_9)} \cap \overline{R}_{I_2(J_9)} \cap \overline{R}_{I_3(J_9)} \cap \overline{R}_{I_4(J_9)}.
\]

To compute \( V_{J_9}(s) \) which is defined on \( \overline{R}_{J_9} \), we use \( x = X_{I_1(J_9)}(s) \) in \( g_1(x, s) \) or \( x = X_{I_2(J_9)}(s) \) in \( g_2(x, s) \) or \( x = X_{I_3(J_9)}(s) \) in \( g_3(x, s) \) or \( x = X_{I_4(J_9)}(s) \) in \( g_4(x, s) \).

Now we assume that \( g(x, s) \) is convex but it is not strictly convex with respect to \( x \). In this case, we need to consider the optimization problem
which is defined by
\[ m_k(s) = \inf_x g_k(x, s) \text{ s.t. } x \in C_k \] (4.13)
as an optimization problem defined in (4.3). Assume the minimum exists in (4.13). Since we are considering that \( \dim(\text{dom } f) = 2 \), the associated element with any index set \( \mathcal{J} \in \mathcal{Y} \) is a relative interior of a two dimensional face, or a relative interior of an edge or a vertex. For any \( \mathcal{F}_{\mathcal{I}_k(\mathcal{J})} \in C_k \) where \( k \in \kappa(\mathcal{J}) \), we get an optimal active set \( \mathcal{A} \) of (4.13). Actually for any \( k \in \kappa(\mathcal{J}) \), the optimal active set \( \mathcal{A} \) for \( \mathcal{F}_{\mathcal{I}_k(\mathcal{J})} \) is \( \mathcal{I}_k(\mathcal{J}) \). For example, consider the index set \( \mathcal{J}_5 \). For \( \mathcal{F}_{\mathcal{I}_1(\mathcal{J})} \), we get the optimal active set \( \mathcal{A} = \{1\} \) of (4.13) with \( k = 1 \) and we have \( \mathcal{I}_1(\mathcal{J}_5) = \{1\} \).

For solving the convex case (not strictly convex), we have to go through every index sets like the strictly convex case. Consider an index set \( \mathcal{J} \in \mathcal{Y} \). For any \( k \in \kappa(\mathcal{J}) \), we need to solve (4.13) by applying Fact 4.22. To solve it, at first we need to solve the equations
\[ L_{E_k(\mathcal{J})}(Q_k x + R_k s + q_k) = 0 \] (4.14)
and
\[ A_{I_k(\mathcal{J})}^k x = b_{I_k(\mathcal{J})}^k \] (4.15)
to get \( x \). Assume the solution gives us \( X_{I_k(\mathcal{J})}(s) \). Then, we apply \( x = X_{I_k(\mathcal{J})}(s) \) in the following inequalities to get \( \mathcal{R}_{I_k(\mathcal{J})} \)
\[ L_{I_k(\mathcal{J})}(Q_k x + R_k s + q_k) \geq 0, \] (4.16)
\[ A_{I_k(\mathcal{J})}^k x \leq b_{I_k(\mathcal{J})}^k. \] (4.17)
We define \( \mathcal{R}_{I_k(\mathcal{J})} \) as
\[ \mathcal{R}_{I_k(\mathcal{J})} = \left\{ s \in \mathbb{R}^2 \mid L_{I_k(\mathcal{J})}(Q_k X_{I_k(\mathcal{J})}(s) + R_k s + q_k) \geq 0, \quad A_{I_k(\mathcal{J})}^k X_{I_k(\mathcal{J})}(s) \leq b_{I_k(\mathcal{J})}^k \right\}. \]

Actually we have used the definition of optimal set mapping for any optimal active set \( \mathcal{A} \) which is defined in Fact 4.22. This definition gives us the information about optimal set mapping and the critical region in which the set mapping is defined. As discussed earlier, an optimal active set \( \mathcal{A} = \mathcal{I}_k(\mathcal{J}) \) is associated with a set \( \mathcal{F}_{I_k(\mathcal{J})} \) which is a vertex or an edge or a face.
4.3. Adaptation of pQP in computational convex analysis

Case 1. If an optimal active set is associated with the set $\mathcal{F}_{\mathcal{I}_k(\mathcal{J})}$ which is a face of a polyhedral region $C_k \subseteq \mathbb{R}^2$, then the normal cone to $C_k$ at any point $\overline{x} \in \text{ri} \mathcal{F}_{\mathcal{I}_k(\mathcal{J})}$ is

$$N_{C_k}(\mathcal{F}_{\mathcal{I}_k(\mathcal{J})}) = \{(0, 0)\}. \quad (4.18)$$

Therefore we get the unique solution $x \in \mathbb{R}^2$ by using Equation (4.14) only. Since we are considering the relative interior of a face of a polyhedral region, there is no active constraint.

Case 2. If an optimal active set is associated with the set $\mathcal{F}_{\mathcal{I}_k(\mathcal{J})}$ which is an edge of a polyhedral region $C_k \subseteq \mathbb{R}^2$, then the normal cone to $C_k$ at any point $\overline{x} \in \text{ri} \mathcal{F}_{\mathcal{I}_k(\mathcal{J})}$ is defined as,

$$N_{C_k}(\mathcal{F}_{\mathcal{I}_k(\mathcal{J})}) = \{y \in \mathbb{R}^2 : L_{E}^{\mathcal{I}_k(\mathcal{J})} y = 0, L_{I}^{\mathcal{I}_k(\mathcal{J})} y \leq 0\}.$$  

Since we are considering an edge, we have an active constraint. Therefore, we could calculate the unique solution $x = X_{\mathcal{I}_k(\mathcal{J})}(s)$ by solving equations (4.14) and (4.15).

Case 3. Now assume an optimal active set is associated with a vertex $v$ of a polyhedral region $C_k \subseteq \mathbb{R}^2$. Since we are considering a vertex, we have two active constraints. Thus, we get the unique solution $x = X_{\mathcal{I}_k(\mathcal{J})}(s)$ by solving Equation (4.15).

For each of these three cases, we get the unique solution $x = X_{\mathcal{I}_k(\mathcal{J})}(s)$ after applying Fact 4.22. Since we are not getting a set of solution, we do not apply Fact 4.23 and Fact 4.24.

Case 4. As a special case we may have $Q_k = 0$ and the optimal active set is associated with a face. In this case we get a set of solutions but the solutions are defined on a critical region which is not full dimensional i.e., two dimensional. It happens due to the following equation which we get by using $Q_k = 0$ in equation (4.14).

$$L_{E}^{\mathcal{I}_k(\mathcal{J})} (R_k s + q_k) = 0.$$ 

Now we may proceed from here by applying facts 4.23, 4.24 and Definition 4.25. Since the set of solutions are defined on a critical region which is not full dimensional, by applying facts 4.23 and 4.24 we will get unique solutions defined on critical regions which are not full dimensional. Since Definition 4.25 defined the solution mapping only on the full dimensional i.e., two dimensional polyhedral regions, we actually do not need to apply facts 4.23, 4.24 and Definition 4.25. We can give another explanation of not proceeding from here. That is, for computing the conjugate, it is enough for us if we can compute the value function defined on a critical region and the
value function can be computed if we need to consider an unique solution or a set of solution. Similar scenario happens when \( Q_k = 0 \) and the optimal active set is associated with an edge.

After computing \( X_{I_k}(J)(s) \) and \( \mathcal{R}_{I_k}(J) \) for all \( k \in \kappa(J) \), we apply \( \mathcal{R}_J = \bigcap_{k \in K(J)} \mathcal{R}_{I_k}(J) \) to compute \( \mathcal{R}_J \). Consequently, we deduce the value function \( V_J(s) \) which is defined on \( \mathcal{R}_J \) as we do for the strictly convex case. The computed conjugate could be written as

\[
\tilde{f}^*(s) = -V_J(s) \quad \text{where} \quad s \in \mathcal{R}_J,
\]

with \( J \in \mathcal{Y} \).

### 4.3.2 Moreau envelope computation

**Modeling**

Define another function \( h : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R} \cup \{+\infty\} \) so that \( h(x, s) = h_k(x, s) \) when \( x \in C_k \) and \( h_k(x, s) \) is defined by

\[
h_k(x, s) = f_k(x) + \frac{1}{2\lambda} \| s - x \|^2,
\]

where \( \lambda > 0 \). Note that \( h_k(x, s) \) is strictly convex. Now the Moreau envelope of \( f \) could be written as

\[
e_\lambda f(s) = \inf_x h(x, s).
\]

We have \( h_k(x, s) = f_k(x_1, x_2) + \frac{1}{2\lambda}[(x_1 - s_1)^2 + (x_2 - s_2)^2] \). Therefore, we express \( h_k \) as

\[
h_k(x_1, x_2; s_1, s_2) = \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2(a_k + \frac{1}{2\lambda}) & c \\ c & 2(b_k + \frac{1}{2\lambda}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}^T \begin{bmatrix} \frac{1}{\lambda} & 0 \\ 0 & \frac{1}{\lambda} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \\
+ \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}^T \begin{bmatrix} -\frac{1}{\lambda} & 0 \\ 0 & -\frac{1}{\lambda} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} d \\ e \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \sigma_k.
\]

For any \( k \in K, C_k \) could be expressed similarly as expressed in Equation (4.7). We could write

\[
e_\lambda f(s) = \inf_x h(x, s), X_\lambda(s) = \arg\min_x h(x, s). \quad (4.19)
\]

Now we apply Fact 4.15 for the optimization problem defined in (4.19).
4.3. Adaptation of pQP in computational convex analysis

Computation of full dimensional critical regions

The computation of full dimensional critical regions could be done similarly to the way it is done for conjugate computation. Replace \( g \) with \( h \). During conjugate computation, \( g \) may be strictly convex or only convex relative to \( x \). Since \( h \) is strictly convex relative to \( x \), the computation is limited to the strictly convex case only.

4.3.3 Proximal average computation

We could also compute the proximal average of two convex bivariate PLQ functions. By using Definition 2.34 of the proximal average, we need to solve three parametric quadratic programs for three conjugate computation as well as we have to compute a PLQ addition for computing the proximal average of two convex bivariate PLQ functions. But there is another way to compute it by using another formulation of the proximal average. Consider the following definition of the proximal average of two functions.

**Definition 4.28.** [BGLW08, Definition 4.1] Let \( f_i : \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\} \) be functions where \( i \in \{1, 2\} \). For \( \lambda_i > 0 \) with \( \sum_{i \in \{1, 2\}} \lambda_i = 1 \) and for any \( \mu > 0 \), the proximal average of the functions \( f_i \) is defined as

\[
p_{\mu}(f, \lambda)(y) = \frac{1}{\mu} \left[ -\frac{1}{2} \|y\|^2 + \inf_{\sum_{i \in \{1, 2\}} y_i = y} \left( \lambda_i \left( \mu f_i(y_i/\lambda_i) + \frac{1}{2} \|y_i/\lambda_i\|^2 \right) \right) \right].
\]

(4.20)

By using the previous definition, we just need to solve a parametric quadratic program for computing the proximal average of two convex bivariate PLQ functions. We have

\[
\inf_{\sum_{i \in \{1, 2\}} y_i = y} \left( \lambda_i \left( \mu f_i(y_i/\lambda_i) + \frac{1}{2} \|y_i/\lambda_i\|^2 \right) \right)
= \inf_{y_1 + y_2 = y} \left( \lambda_1 \left( \mu f_1(y_1/\lambda_1) + \frac{1}{2} \|y_1/\lambda_1\|^2 \right) + \lambda_2 \left( \mu f_2(y_2/\lambda_2) + \frac{1}{2} \|y_2/\lambda_2\|^2 \right) \right)
= \inf_{y_1} \left( \lambda_1 \left( \mu f_1(y_1/\lambda_1) + \frac{1}{2} \|y_1/\lambda_1\|^2 \right) + \lambda_2 \left( \mu f_2((y - y_1)/\lambda_2) + \frac{1}{2} \|(y - y_1)/\lambda_2\|^2 \right) \right)
= \inf_{y_1} g(y_1, y).
\]

Since \( (y - y_1)/\lambda_2 \) is an affine function and \( f_2 \) is convex, \( f_2((y - y_1)/\lambda_2) \) is a convex function [RW98, Exercise 2.20(a)]. Similarly \( f_1(y_1/\lambda_1) \) is a convex function also. Since \( \|y_1/\lambda_1\|^2 \) and \( \|(y - y_1)/\lambda_2\|^2 \) are convex functions and
convex functions are closed under addition and scalar multiplication, \( g(y_1, y) \) is also a convex function. We define a parametric piecewise quadratic optimization problem as

\[
V(y) = \inf_{y_1} g(y_1, y), \quad X(y) = \arg\min_{y_1} g(y_1, y).
\] (4.21)

Now we apply Fact 4.15 to solve the optimization problem defined in (4.21). Consequently we compute the proximal average defined in Equation (4.20).
Chapter 5

Algorithm

In this chapter we discuss the full algorithm based on parametric programming to compute the convex operators (conjugate and Moreau envelope) of any convex bivariate PLQ function.

5.1 Data structure

Our implementation in Scilab [Sci94] represents a convex bivariate PLQ function using a list of data structures for taking input and giving output. But this data structure is converted to another data structure (face-constraint adjacency data structure) internally. We will discuss both of these data structures but we will focus mainly on the face-constraint adjacency data structure to present the algorithm and compute its complexity.

5.1.1 List representation of bivariate PLQ functions

Recall that each piece of a PLQ function $f$ is a function defined on a polyhedral set

$$C_k = \{ x \in \mathbb{R}^2 | A^k x - b^k \leq 0 \},$$

where $A^k \in \mathbb{R}^{m_k \times 2}$ and $b^k \in \mathbb{R}^{m_k \times 1}$. On $C_k$, $f$ is equal to

$$f^k(x) = (1/2)x^T Q^k x + q^T_k x + \sigma^k,$$

where $\sigma_k \in \mathbb{R}$, $q_k \in \mathbb{R}^2$ and $Q_k \in \mathbb{R}^{2 \times 2}$ is a symmetric matrix.

Using these definitions, we store a piece of a bivariate PLQ function in a structure containing the constraint set $C_k$ and function value $f^k$. Therefore we store the full bivariate PLQ function using a list of such structures. We store $C_k$ and $f^k$ in $m_k \times 3$ and $1 \times 6$ matrixes respectively. Each row $[c_1, c_2, c_3]$ in $C_k$ represents the constraint $c_1 x_1 + c_2 x_2 + c_3 \leq 0$. The row $[f^k_1, f^k_2, f^k_3, f^k_4, f^k_5, f^k_6]$ in $f^k$ stores the function $f^k(x_1, x_2) = f^k_1 x_1^2 + f^k_2 x_2^2 + f^k_3 x_1 x_2 + f^k_4 x_1 + f^k_5 x_2 + f^k_6$.

Example 5.1. For example, $f(x_1, x_2) = |x_1| + |x_2|$ is represented as a list of four structures where each structure represents each individual piece.
5.1. Data structure

The first piece is represented as

\[ C_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, f_1 = [0 0 0 1 1 0], \]

and similarly

\[ C_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}, f_2 = [0 0 0 -1 1 0], \]

\[ C_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, f_3 = [0 0 0 -1 -1 0], \]

\[ C_4 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, f_4 = [0 0 0 1 -1 0]. \]

The structure of the first piece informs us that the function \( x_1 + x_2 \) is defined on the polyhedral set \( \{(x_1, x_2) | x_1 \geq 0, x_2 \geq 0\} \). The other pieces are defined similarly.

5.1.2 Face-constraint adjacency representation of a bivariate PLQ function

In this representation a bivariate PLQ function is defined as \((C, f, E_{f,C})\) where \( C \) is a \( n_c \times 3 \) matrix (where \( n_c \) is the number of distinct constraints) storing each constraint, \( c_1 x_1 + c_2 x_2 + c_3 \leq 0 \) as a row, \([c_1, c_2, c_3] \) in \( C \). We define \( f \) which is a \( n_f \times 6 \) matrix where \( n_f \) is the number of faces. Each row \([f_1^k, f_2^k, f_3^k, f_4^k, f_5^k, f_6^k] \) represents the associated quadratic function with each face, \( f^k(x_1, x_2) = f_1^k x_1^2 + f_2^k x_2^2 + f_3^k x_1 x_2 + f_4^k x_1 + f_5^k x_2 + f_6^k \). The face-constraint adjacency is represented by \( E_{f,C} \) which is a \( n_f \times n_C \) binary matrix.

Example 5.2. For example, \( f(x_1, x_2) = |x_1| + |x_2| \) is represented as

\[
C = \begin{bmatrix}
-1 & 0 & 0 \\
0 & -1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}, f = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 & 0
\end{bmatrix}, E_{f,C} = \begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1
\end{bmatrix}.
\]

From \( C \) we deduce there are four constraints. Each row in \( C \) represents a constraint. For example, the second row \([0, -1, 0] \) in \( C \) represents the constraint with parameters \( c_1 = 0, c_2 = -1 \) and \( c_3 = 0 \) that is \( -x_2 \leq 0 \). Since the matrix \( f \) has four rows, we deduce that there are four faces. The first
5.1. Data structure

row of \( f \) indicates that the function with parameters \( f_1 = 0, f_2^1 = 0, f_3^1 = 0, f_4^1 = 1, f_5^1 = 1 \) and \( f_6^1 = 0 \) is associated with the first face. So \( x_1 + x_2 \) is that associated function. The remaining rows of \( f \) stores other functions associated with other faces. The rows in \( E_{f,C} \) indicates the face-constraint adjacency information for each of the four faces. The entry 1 in \( E_{f,C} \) indicates adjacency while 0 indicates non-adjacency between corresponding face and constraint. For example, the first row in \( E_{f,C} \) is \([1, 1, 0, 0]\) meaning that the boundary of the first face is defined by the first and second constraints. Similarly the fourth row of \( E_{f,C} \), \([1, 0, 0, 1]\) indicates that the boundary of the fourth face is defined by the first and fourth constraints.

5.1.3 The data structures to store lines, rays, segments and vertexes

We store the lines in the domain by using a \( n_L \times 3 \) matrix \( M_L \) which is defined as

\[
M_L = \begin{bmatrix}
  l_1^1 & l_2^1 & l_3^1 \\
  \vdots & \vdots & \vdots \\
  l_n^1 & l_n^2 & l_n^3 \\
\end{bmatrix}
\]

Each row of \( M_L \) represents a line. Consider the first row \([l_1^1, l_2^1, l_3^1]\) of \( M_L \). The row represents the line \( l_1^1 x_1 + l_2^1 x_2 + l_3^1 = 0 \).

Any ray is represented by using a line and a constraint. We use a \( n_R \times 6 \) matrix \( M_R \) to represent \( n_R \) rays in domain. The matrix \( M_R \) is defined by

\[
M_R = \begin{bmatrix}
  r_1^1 & r_2^1 & r_3^1 & r_1^c & r_2^c & r_3^c \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  r_{n_R}^1 & r_{n_R}^2 & r_{n_R}^3 & r_{n_R}^c & r_{n_R}^c & r_{n_R}^c \\
\end{bmatrix}
\]

The first row \([r_1^1, r_2^1, r_3^1, r_1^c, r_2^c, r_3^c]\) of \( M_R \) represents a ray which is defined by \( r_1^1 x_1 + r_2^1 x_2 + r_3^1 = 0 \) and \( r_1^c x_1 + r_2^c x_2 + r_3^c \leq 0 \). Similarly we define the other rows.

For \( n_V \) vertexes in the domain, we use a \( n_V \times 2 \) matrix \( M_V \) to store them. We define \( M_V \) as

\[
M_V = \begin{bmatrix}
  v_1^1 & v_1^2 \\
  \vdots & \vdots \\
  v_{n_V}^1 & v_{n_V}^2 \\
\end{bmatrix}
\]
Each row of $M_V$ represents a vertex. For example, the first row of $M_V$ represents the vertex with $x_1 = v_1^1$ and $x_2 = v_1^2$.

A segment is defined by two vertexes. We store $n_S$ segments in the domain by using a $n_S \times 2$ matrix $M_S$ which is defined as

$$M_S = \begin{bmatrix} r_{v_1}^1 & r_{v_2}^1 \\ \vdots & \vdots \\ r_{v_1}^{n_S} & r_{v_2}^{n_S} \end{bmatrix}.$$  

Each entry in $M_S$ represents a reference to a vertex in $M_V$. The row $[r_{v_1}^1, r_{v_2}^1]$ represents a segment between the two vertexes which are referenced by $r_{v_1}^1$ and $r_{v_2}^1$. Other segments are defined similarly.

**Example 5.3.** For example we consider the domain of 1-norm, the function $f(x_1, x_2) = |x_1| + |x_2|$. It is illustrated in Figure 5.1. The domain has four faces, four rays and one vertex. We store the rays by using a $4 \times 6$ matrix $M_R$ i.e.,

$$M_R = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \end{bmatrix}.$$  

The vertex is stored as

$$M_V = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$  

**Example 5.4.** We consider the domain of indicator function $I_{[-1,1]}$. It is illustrated in Figure 5.2. There are one face, four segments and four vertexes in the domain. The vertexes are stored as

$$M_V = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}.$$  

We store the segments as

$$M_S = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}.$$  

47
5.1. Data structure

Figure 5.1: Domain of the function, \( f(x_1, x_2) = |x_1| + |x_2| \).

5.1.4 The data structure to store entity-face and entity-constraint adjacency information

Assume we have \( n_f \) faces, \( n_L \) lines, \( n_S \) segments, \( n_R \) rays and \( n_V \) vertices. We define a \((n_f + n_L + n_S + n_R + n_V) \times n_f\) binary matrix \( M_{E,f} \) to store entity-face adjacency information. We use \( e_i \) to denote any entity with \( i \in \{1, 2, \ldots, n_f + n_L + n_S + n_R + n_V\} \). We denote the entities \( e_i \) with \( i \in \{1, 2, \ldots, n_f\} \) are faces. Similarly \( e_i \) with \( i \in \{n_f + 1, \ldots, n_f + n_L\} \) are lines. The segments are \( e_i \) with \( i \in \{n_f + n_L + 1, \ldots, n_f + n_L + n_S\} \). The rays are \( e_i \) with \( i \in \{n_f + n_L + n_S + 1, \ldots, n_f + n_L + n_S + n_R\} \). Similarly the vertexes are \( e_i \) where \( i \in \{n_f + n_L + n_S + n_R + 1, \ldots, n_f + n_L + n_S + n_R + n_V\} \).

We define a \((n_f + n_L + n_S + n_R + n_V) \times n_f\) binary matrix \( M_{E,f} \) to store entity-face adjacency information. The matrix \( M_{E,f} \) is defined by

\[
M_{E,f}(i, j) = \begin{cases} 
1, & \text{if } e_i \in e_j; \\
0, & \text{otherwise}.
\end{cases}
\]

where \( i \in \{1, 2, \ldots, n_f + n_L + n_S + n_R + n_V\} \) and \( j \in \{1, 2, \ldots, n_f\} \).
5.1. Data structure

Figure 5.2: Domain of indicator function $I_{[-1,1]}$.

Assume we have $n_C$ constraints. We use $C_i$ with $i \in \{1, 2, \ldots, n_C\}$ to denote a constraint. We define a $(n_f + n_L + n_S + n_R + n_V) \times n_C$ binary matrix $M_{E,C}$ to store the entity-constraint adjacency information which is defined as

$$M_{E,C}(i,j) = \begin{cases} 
1, & \text{if } e_i \text{ is bounded by } C_j; \\
0, & \text{otherwise.}
\end{cases}$$

where $i \in \{1, 2, \ldots, n_f + n_L + n_S + n_R + n_V\}$ and $j \in \{1, 2, \ldots, n_C\}$.

**Example 5.5.** We again consider the domain of the function $f(x_1, x_2) = |x_1| + |x_2|$ which is shown in Figure 5.1. There are nine entities in the domain i.e., four faces, four rays and one vertex. We store computed entity-face adjacency information by using a $9 \times 4$ binary matrix $M_{E,f}$ defined as

$$M_{E,f} = \begin{bmatrix} 
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}.$$
5.1. Data structure

The collection of constraints is represented as

\[
M_C = \begin{bmatrix}
-1 & 0 & 0 \\
0 & -1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}.
\]

There are four constraints in \( M_C \). We store computed entity-constraint adjacency information by using a \( 9 \times 4 \) binary matrix \( M_{E,C} \) which is defined as

\[
M_{E,C} = \begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}.
\]

5.1.5 The Half-Edge data structure [CGA, Ope]

The Half-Edge data structure is an edge centered data structure which could maintain the incident information of vertexes, edges and faces in a two dimensional polyhedral subdivision. Each edge is decomposed into two half-edges with opposite directions. We describe the data structure by using an example. Consider Figure 5.3 where each edge is decomposed into two half-edges.

1. Every vertex references one outgoing half-edge. For example the vertex \( v_1 \) references the half-edge \( h_1 \).

2. Every face references one of its bounding half-edges. For example the face \( f_1 \) references \( h_2 \).

3. Every half-edge provides references to the following elements.
   - The vertex it points to i.e., the half-edge \( h_2 \) provides reference to \( v_1 \).
   - The face in which it belongs. For example the half-edge \( h_2 \) has a reference to the face \( f_1 \).
5.1. Data structure

- The next half-edge in the face in counter-clockwise direction. For example $h_2$ references to $h_3$.
- The opposite half-edge. For example $h_2$ has a reference to $h_4$.
- The previous half-edge in the belonging face. Reference to this element is optional. For example $h_2$ has a reference to $h_5$.

**Example 5.6.** For example consider the 1-norm, $f(x_1, x_2) = |x_1| + |x_2|$. Its domain is illustrated using half-edges in Figure 5.4. A representation of the domain using the half-edge data structure is as follows.

1. The vertex $v_1$ references one outgoing half-edge e.g. $h_1$.
2. The face $f_1$ references one of its bounding edges e.g. $h_1$.
3. The face $f_2$ references one of its bounding edges e.g. $h_3$.
4. The face $f_3$ references one of its bounding edges e.g. $h_5$.
5. The face $f_4$ references one of its bounding edges e.g. $h_7$.
6. The half-edge $h_1$ provides references to the following elements.
5.2. The Algorithms

- The vertex it points to. It is null.
- The face in which it belongs. It is $f_1$.
- The next half-edge in the face in counter-clockwise direction. It is null.
- The opposite half-edge. It is $h_8$.
- The previous half-edge in the belonging face. It is $h_2$.

7. Similarly other half-edges i.e., $h_2, h_3, h_4, h_5, h_6, h_7$ and $h_8$ provide references.

![Diagram](image)

Figure 5.4: The representation of the domain of $f(x_1, x_2) = |x_1| + |x_2|$ using half-edge data structure.

### 5.2 The Algorithms

For an index set $\mathcal{J} \in \mathcal{Y}$, we compute the closure of the critical region $\mathcal{R}_\mathcal{J}$ which is defined as

$$\mathcal{R}_\mathcal{J} = \bigcap_{k \in \mathcal{K}(\mathcal{J})} \mathcal{R}_{I_k(\mathcal{J})},$$
where $R_{I_k(J)}$ for the parametric quadratic program $\inf_x \{ \psi_k(x,p) : (x,p) \in C_k \}$ corresponding to the equality set $I_k(J) \in \mathcal{B}(C_k)$, $k \in \kappa(J)$ is defined as

$$R_{I_k(J)} = \begin{cases} \exists (x, \lambda_k), \nabla_x \psi_k(x,p) + A^k_{I_k(J)} \lambda_k = 0, \lambda_k \geq 0 \\ A^k_{I_k(J)} x = b^k_{I_k(J)} \\ A^k_{\pi_k(J)} x \leq b^k_{\pi_k(J)} \end{cases}.$$ 

An entity is a set in $\mathbb{R}^2$ which is either a vertex, an edge or a face. We use $\psi(x,s)$ to denote $g(x,s)$ or $h(x,s)$. Note that $g(x,s)$ is defined in Subsection 4.3.1 and $h(x,s)$ is defined in Subsection 4.3.2. We need to consider every index set $J \in \mathcal{Y}$ of $\text{dom} \psi$. For each index set $J \in \mathcal{Y}$ we have a $D_J$ which is a vertex or edge or face in the domain. Moreover, for any index set $J \in \mathcal{Y}$ we need to consider all $k \in \kappa(J)$. Due to these facts, the main idea of the algorithm is to loop through each entity in the domain of a bivariate convex PLQ function. For each entity, it is needed to consider each of its adjacent faces. For any index set $J \in \mathcal{Y}$ and for any $k \in \kappa(J)$, we have $F_{I_k(J)}$ which is a vertex, an edge, or a face. Therefore, $F_{I_k(J)}$ represents an entity. We also have $G_{I_k(J)}$ which is the relative interior of the entity $F_{I_k(J)}$. The entity $F_{I_k(J)}$ is the same entity as $D_J$ for any $k \in \kappa(J)$. In our model, for all $k \in \kappa(J)$, $C_k$ are the faces. The entity $F_{I_k(J)}$ is a face when $I_k(J) = \emptyset$. Actually, $F_\emptyset = C_k$ for any $k \in \kappa(J)$.

Our implementation in Scilab [Sci94] to compute entities and adjacency information

In our implementation using Scilab [Sci94], we start with List representation of bivariate PLQ functions which is defined in Subsection 5.1. This data structure does not give us all entities of the domain explicitly except faces. We use a brute-force approach to compute the remaining entities (vertex, ray, segment, line). At first we collect all constraints together and store them in a matrix. Then we compute the vertexes. For each constraint we loop through all other constraints to compute the intersection points. During the computation of vertexes, we get the adjacency information between vertexes and constraints and also between vertexes and faces. We get lines from those constraints which are not adjacent to any vertex. After that, we compute the segments and rays for each face. For each face, we loop through all of its adjacent constraints, and for each constraint we loop through all of its adjacent vertexes to the current face to compute associated segments and rays. We also get the adjacency information of the entities (segments, rays and lines) with faces and constraints.
An improved strategy to compute entities and adjacency information

It is possible to compute all entities and their adjacency information with faces and constraints more efficiently. Assume we start with the face-constraint adjacency representation of any bivariate PLQ function which is defined in Section 5.2. We compute all of the vertexes and also the adjacency information between vertexes and constraints using Mulmuley’s segment intersection algorithm [Mul88, MS92]. This algorithm finds all intersecting pairs of segments in a set of line segments in the plane. Since we have all constraints in $C$, we get all lines corresponding to the constraints. We represent any line from these lines as a line segment by bounding it with large boundaries. The Mulmuly’s algorithm may give some extra intersection points which are not vertexes. We eliminate the unnecessary intersection points by using the planar point location algorithm [Kir83, ST86]. This algorithm determines the polyhedral region of a subdivision in which a query point lies. As a biproduct of unnecessary intersection points elimination, we get vertex-face adjacency information at no extra cost.

For example consider a polyhedral subdivision with three polyhedral sets $P_1, P_2$ and $P_3$ in Figure 5.5. Here the green lines present the boundaries of the polyhedral sets. After applying Mulmuly’s algorithm, we get all intersecting pairs of line segments. The red colored intersection points are the required vertexes and the blue colored intersection points are the extra intersection points. Then we apply the planar point location algorithm for all intersection points to eliminate the unnecessary intersection points. Suppose we are considering the intersection point $v_1$. After applying the planar point location algorithm on it, we will get that it does not belongs to any polyhedral set. Therefore, we eliminate it. Assume we are considering another intersection point $v_2$ whose intersecting pair of line segment is $(l_1, l_3)$. The planar point location algorithm will return the polyhedral set $P_3$. Since $l_3$ is not a boundary line of $P_3$, we eliminate this intersection as an unnecessary intersection point. Now consider the intersection point $v_3$ whose intersecting line segments are $l_1$ and $l_6$. For this intersection point, the planar point location algorithm will return $P_3$. Since both of $l_1$ and $l_6$ are boundary lines of $P_3$, $v_3$ will not be eliminated.

The remaining entities are lines, segments and rays. A line does not contain any vertex, otherwise it will be a ray or segment. Since we have already got the adjacency information between vertexes and constraints, we retrieve the lines in the domain by using those constraints which are not adjacent to any vertex. For example, assume a constraint $c_1x_1 + c_2x_2 + c_3 \leq 0$.
5.2. The Algorithms

is not adjacent to any vertex. Therefore, we get a line \( c_1 x_1 + c_2 x_2 + c_3 = 0 \). The adjacency information between lines and constraints is very trivial to compute because lines come from the constraints.

We compute the segments and rays using the information we have so far. We have retrieved all of the vertexes. We also have the adjacency information between constraints and vertexes. But we have no idea about the vertex order on a line corresponding to a constraint. For example, consider again Figure 5.5. We already know that the vertexes \( v_3, v_4 \) and \( v_5 \) are on the line segment \( l_1 \). But we do not know the vertex order of the vertexes \( v_3, v_4 \) and \( v_5 \) on \( l_1 \).

We need to loop through all of the vertexes and for each vertex we need to loop through its adjacent constraints to compute the associated segments or rays. We have no information about the order of vertexes along the line corresponding to a constraint. Therefore, we may have to find out the vertex from a set of vertexes so that this vertex and another particular vertex belongs to the same polyhedral region. We do this by applying again the planar point location algorithm. Another way to do this by using the parametric representation of the line corresponding to the constraint. For each vertex in the set of vertexes, we calculate the value of parameter in

Figure 5.5: Computation of the vertexes.
the parametric representation of the line. After that we sort the parameter values to compute the nearest vertex.

For example consider Figure 5.6. Suppose we are at vertex $v_4$ and considering the adjacent line segment $l_1$ along the direction from $v_4$ to $v_3$. From the already known adjacency information, we retrieve that the vertexes $v_3$ and $v_6$ are on $l_1$ along this direction. Then we need to compute a vertex from $v_3$ and $v_6$ such that this computed vertex and $v_4$ belong to the same polyhedral region. The computed vertex is $v_3$ and this is the nearest vertex from $v_4$ along the direction from $v_4$ to $v_3$. Consequently we get the segment between $v_4$ and $v_3$.

![Figure 5.6: Computation of the remaining entities by using computed vertexes.](image)

Since we need to loop through all of the adjacent constraints for each vertex to compute corresponding rays or segments, we get the adjacency information between segments or rays and the constraints. Since $E_{f,c}$ gives the face-constraint adjacency information, using $E_{f,c}$ we deduce the adjacency information of all edges (lines, segments, rays) with faces. Algorithm 1 computes the segments and rays.

We already know the adjacency information between entities and faces. We just need to extract this information. Algorithm 2 extracts the adjacency
5.2. The Algorithms

Algorithm 1: Compute segments and rays

input: $C$ (the set of constraints), $V$ (set of vertexes), $E_{f,C}$
(adjacency information between faces and constraints), $E_{V,C}$
(adjacency information between vertexes and constraints)

output: $set\_of\_segments$ (all segments), $set\_of\_rays$ (all rays)

for $v \in V$ do
    foreach $C_i \in C$ adjacent to $v$ do
        compute associated segments and rays;
        merge the segments into $set\_of\_segments$;
        merge the rays into $set\_of\_rays$;
    end
end

In addition to all of the previously computed information, we need the adjacency information between all entities and constraints. The matrix $E_{f,C}$ gives the adjacency information between faces and constraints. The adjacency information between vertexes and constraints is retrieved already. The adjacent constraints for any ray or segment is also known. The adjacency information between lines and constraints is trivial. Algorithm 3 extracts the adjacency information between entities and constraints.

Computation of conjugate or the Moreau-envelope

After having all of the previous computed informations, we loop through all of the different entities in the domain. For each entity, we loop through all of its adjacent faces to compute corresponding critical regions and we also get corresponding solutions. Consequently, for each entity we calculate a critical region and define a value function on it. From these critical regions with associated value functions, we get the conjugate or the Moreau envelope. For example, consider the domain of the function $f(x_1, x_2) = |x_1| + |x_2|$ which is illustrated in 5.1. One of its rays is adjacent to two faces. The single vertex is adjacent to all of the four faces. For every ray, the algorithm computes the critical region as well as the associated solution for each of the two adjacent faces. Then it deduces a critical region and defines a value function on it. Algorithm 4 represents these tasks. For ease of describing this algorithm, we have used some definitions and notations. We define $set\_of\_entities$ as $D = \{D_J : J \in \mathcal{J}\}$. For any $k \in \kappa(J)$, we have $D_J = \mathcal{F}_{I_k(J)}$. For any
Algorithm 2: Extract entity face adjacency information

**input**: $L$ (the set of lines), $V$ (the set of vertexes), $S$ (the set of segments), $R$ (the set of rays), $E_{L,f}$ (the adjacency information between lines and faces), $E_{V,f}$ (the adjacency information between vertexes and faces), $E_{S,f}$ (the adjacency information between segments and faces), $E_{R,f}$ (the adjacency information between rays and faces)

**output**: `entity_face_adjacency` (adjacency information between entities and faces)

```plaintext
foreach $l \in L$ do
    extract the adjacency information between $l$ and faces ($E_{L,f}(l)$);
    merge $E_{L,f}(l)$ into `entity_face_adjacency`;
end

foreach $v \in V$ do
    extract the adjacency information between $v$ and faces ($E_{V,f}(v)$);
    merge $E_{V,f}(v)$ into `entity_face_adjacency`;
end

foreach $s \in S$ do
    extract the adjacency information between $s$ and faces ($E_{S,f}(s)$);
    merge $E_{S,f}(s)$ into `entity_face_adjacency`;
end

foreach $r \in R$ do
    extract the adjacency information between $r$ and faces ($E_{R,f}(r)$);
    merge $E_{R,f}(r)$ into `entity_face_adjacency`;
end
```
5.2. The Algorithms

**Algorithm 3: Extract entity constraint adjacency information**

**input**: \( L \) (the set of lines), \( V \) (the set of vertexes), \( S \) (the set of segments), \( R \) (the set of rays), \( E_{f,C} \) (the adjacency information between faces and constraints), \( E_{L,C} \) (the adjacency information between lines and constraints), \( E_{V,C} \) (the adjacency information between vertexes and constraints), \( E_{S,C} \) (the adjacency information between segments and constraints), \( E_{R,C} \) (the adjacency information between rays and constraints)

**output**: \( \text{entity\_constraint\_adjacency} \) (adjacency information between entities and constraints)

\[
\text{entity\_constraint\_adjacency} \leftarrow E_{f,C};
\]

foreach \( l \in L \) do

- extract the adjacency information between \( l \) and constraints \( (E_{L,C}(l)) \);
- merge \( E_{L,C}(l) \) into \( \text{entity\_constraint\_adjacency} \);
end

foreach \( v \in V \) do

- extract the adjacency information between \( v \) and constraints \( (E_{V,C}(v)) \);
- merge \( E_{V,C}(v) \) into \( \text{entity\_constraint\_adjacency} \);
end

foreach \( s \in S \) do

- extract the adjacency information between \( s \) and constraint \( (E_{S,C}(s)) \);
- merge \( E_{S,C}(s) \) into \( \text{entity\_constraint\_adjacency} \);
end

foreach \( r \in R \) do

- extract the adjacency information between \( r \) and constraints \( (E_{R,C}(r)) \);
- merge \( E_{R,C}(r) \) into \( \text{entity\_constraint\_adjacency} \);
end
5.2. The Algorithms

entity $D_J$, the set of adjacent faces are defined by $F_J = \{F_{\emptyset k} : k \in \kappa(J)\}$. For any $k \in \kappa(J)$, we have $F_{\emptyset k} = C_k$.

**Algorithm 4: Conjugate Computation**

**input**: $C$, $D$ (the set of all entities), $\{F_J\}_{J \in Y}$ (the adjacency information between all entities and faces), entity_constraint_adjacency (the adjacency information between entities and constraints), $f$, is_strictly_convex (whether $\psi(x, p)$ is strictly convex or only convex with respect to $x$)

**output**: $C^*$, $f^*$, $E_{f_c}$

**foreach** $D_J \in D$ **do**

1. $R_J \leftarrow \emptyset$;
2. $X_J \leftarrow \emptyset$;
3. **foreach** $F_{\emptyset k} \in F_J$ **do**
   1. **if** is_strictly_convex **then**
      1. $(\overline{R}_{I_h(J)}, X_{I_h(J)}(s)) \leftarrow \text{Strictly Convex}(\psi(x,p), F_{\emptyset k}, F_{I_h(J)}(J));$
      2. **end**
   2. **else**
      1. $(\overline{R}_{I_h(J)}, X_{I_h(J)}(s)) \leftarrow \text{Convex}(\psi(x,p), F_{\emptyset k}, F_{I_h(J)}(J));$
      2. **end**
   3. $R_J \leftarrow R_J \cap \overline{R}_{I_h(J)}$;
   4. $X_J \leftarrow X_J \cup X_{I_h(J)}(s)$;
5. **end**
6. $V_J(s) = \psi_k(X_{I_h(J)}(s), s)$ for any $k \in \kappa(J)$;

**compute** $C^*$, $f^*$, $E_{f_c}$

Our algorithm works only for convex functions with respect to $x$. We have divided the computation of the closure of critical regions $\overline{R}_{I_h(J)}$ and the associated solution functions $X_{I_h(J)}(s)$ in strictly convex ($\psi(x,p)$ is strictly convex with respect to $x$) and convex ($\psi(x,p)$ is convex but not strictly convex with respect to $x$) cases. Function **Strictly Convex** and Function **Convex** are for the strictly convex and convex cases respectively.

After computing a PLQ function $V(s)$ with $V(s) = V_J(s)$ when $s \in \overline{R}_J$, we need to compute the information to get a face-constraint adjacency representation ($C^*, f^*, E_{f_c}^*$). We already know the set of constraints and
5.3 Complexity Analysis

As we noted earlier, we use a brute force implementation using Scilab for computing entities and adjacency information. For \(O(n_C)\) constraints, our brute-force implementation computes the vertices in \(O(n_C^2)\) time. The computation of segments and rays takes \(O(n_f n_c n_V)\) time for \(O(n_f)\) faces, \(O(n_c)\) constraints and \(O(n_V)\) vertexes.

Now we describe the complexity of the algorithm discussed throughout this chapter which uses an improved strategy to compute entities and ad-
5.3. Complexity Analysis

The space and time complexity of the algorithm is analyzed according to the different parts of the algorithm.

For \( n_C \) different constraints and \( n_f \) different faces, the space complexity of \( C \) and \( f \) is \( \mathcal{O}(n_C) \) and \( \mathcal{O}(n_f) \) respectively. Adjacency list representation [CLRS09, Chapter 22] of \( E_{f,C} \) takes \( \mathcal{O}(n_C + n_f) \) space. For \( \mathcal{O}(n_C) \) line segments and \( \mathcal{O}(n_V) \) vertexes, the Mulmuly’s algorithm takes \( \mathcal{O}(n_C + n_V) \) space. The planar point location algorithm which is used to eliminate the unnecessary constraints, needs \( \mathcal{O}(n_C) \) storage for \( \mathcal{O}(n_C) \) line segments. For \( n_f \) faces, \( n_C \) constraints, and \( n_V \) vertexes, we have \( \mathcal{O}(n_f + n_C + n_V) \) entities. Since in a planar graph with \( v \) vertexes and \( e \) edges, \( e = \mathcal{O}(v^2) \), we deduce that we have \( \mathcal{O}(n_f + n_C) \) entities. As earlier, it is possible to manipulate the adjacency information between entities and faces with \( \mathcal{O}(n_C + n_f) \) space complexity if we use adjacency list representation. Similarly, entity-constraint adjacency is represented in \( \mathcal{O}(n_C) \) space. Therefore, the total space complexity is \( \mathcal{O}(n_C + n_f) \).

The time complexity comes from the three different parts of computation.

First part. We compute vertexes using the Mulmuly’s algorithm. It takes \( \mathcal{O}(n_C \log n_C + n_V) \) time with \( n_C \) line segments and \( n_V \) vertexes. For \( \mathcal{O}(n_C) \) line segments, the planar point location algorithm requires \( \mathcal{O}(\log n_C) \) time for each query point. Thus, the elimination of unnecessary constraints takes \( \mathcal{O}(n_V \log n_C) \) time. Therefore, the total time complexity for computing the vertexes is \( \mathcal{O}(n_C \log n_C + n_V + n_V \log n_C) \). The domain of a PLQ bivariate function is represented by a planar graph where vertexes and edges are computed from the constraints. In that planar graph we have \( \mathcal{O}(n_C) \) edges and \( \mathcal{O}(n_V) \) vertexes. Therefore, we have \( n_C = \mathcal{O}(n_V^2) \). So we deduce the total time complexity to compute the vertexes is \( \mathcal{O}(n_C \log n_C) \).

Second part. We compute the remaining entities. If we consider the domain as a planar graph, in this stage we actually need to loop through all adjacent edges for every vertex. For each edge, we may need to apply the planar point location algorithm for a set of vertexes. For \( \mathcal{O}(n_C) \) line segments and \( \mathcal{O}(n_V) \) vertexes this computation takes \( \mathcal{O}(n_V \log n_C) \) time. Therefore, a computation with \( \mathcal{O}(n_V \log n_C) \) time complexity may be needed in each adjacent edge of any vertex. Therefore, the total time complexity for computing the remaining entities is \( \mathcal{O} \left( n_V \log n_C \sum_{i \in \{1,2,...,n_V\}} \deg(v_i) \right) \). By using the Fact 2.41, we deduce that time complexity is \( \mathcal{O}(n_V n_C \log n_C) \).

Third part. We do the computation of conjugate or Moreau envelope using Algorithm 4. For every entity-face adjacency we do a constant time computation. An entity like segment, ray or line, is adjacent to at most
two faces in a polyhedral subdivision. For example consider the polyhedral subdivision which is illustrated in Figure 5.7. Every ray or segment is adjacent to at most two faces. A vertex is adjacent to more than two faces. For example Vertex 2 and Vertex 3 are adjacent to four faces in the polyhedral subdivision illustrated in Figure 5.7. By using the Euler’s formula which is provided in Fact 2.42, we could say this type of vertexes are rare or they are adjacent to approximately a constant number of faces. Due to these facts, we deduce that the time complexity of Algorithm 4 remains linear, which is $O(n_f + n_C)$.

![Figure 5.7: A polyhedral subdivision.](image)

By summing the time complexities from three different parts of computation, we get the overall time complexity as

$$O(n_C \log n_C + n_V n_C \log n_C + n_f + n_C) = O(n_V n_C \log n_C).$$

Finally, the time to build the conjugate data structure is log-linear using the same argument as [GL11a].

Actually we could compute the conjugate or Moreau envelope in linear time by using an enriched data structure but by breaking the data structure similarity between the input and output of computation.

**Theorem 5.7.** We could compute the conjugate or Moreau envelope of any convex bivariate PLQ function in linear time by using an enriched data structure like the Half-Edge data structure for taking input and without building the output in a similar data structure.

**Proof.** The overall complexity of our algorithm is not linear. It is due to a time complexity of $O(n_c \log n_c)$ which arises during the computation of
vertexes, and also for $O(n_V n_C \log n_C)$ time complexity for computing the rays and segments.

Since an enriched data structure like the Half-Edge data structure gives all of the vertexes, we do not need to compute them.

In our algorithm we do not know the ordering information of vertexes along a line corresponding to a constraint. For this reason we may need to do a computation of $O(n_V \log n_C)$ time complexity in each adjacent edge of any vertex during the computation of the rays and segments. In an enriched data structure like the Half-Edge data structure, a half-edge corresponding to an edge has reference to its pointing vertex. Therefore when we are at a vertex, we just need to go through its adjacent edges to get the corresponding rays and segments. Thus, the run time to compute all segments and rays becomes $O\left(\sum_{i \in \{1,2,...,n_V\}} \deg(v_i)\right) = O(n_C)$.

Therefore, by using an enriched data structure we could compute the conjugate or Moreau envelope of any convex bivariate PLQ function in $O(n_C + n_f)$ time.

Equation 5.7 does not consider the complete computation. The cost of building the conjugate data structure is still log-linear [GL11a], and the question of whether there exists a linear-time algorithm taking as input the function $f$ in the half-edge data structure and outputting $f^*$ as a half-edge data format is still open.
Chapter 6

Examples

We present some examples for which we have computed the conjugate by applying our algorithm. The algorithm is implemented in Scilab [Sci94]. We used the SCAT package [BH06, BH08] in the Maple 15 toolbox [MAP] to verify our conjugate computation.

Example 6.1. Consider the $l_1$ norm, $f(x_1, x_2) = |x_1| + |x_2|$. Its conjugate is $f^*(s_1, s_2) = I_{\{(s_1, s_2)|-1 \leq s_1 \leq 1, -1 \leq s_2 \leq 1\}}(s_1, s_2)$. The bivariate PLQ function and its conjugate are illustrated in Figure 6.1.

Example 6.2. We next demonstrate the conjugate of the 2D energy function. The function is $f(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2)$. It has only one piece with unbounded domain which is the full 2D space. Therefore, it has only one entity in the domain. The 2D energy function, illustrated in Figure 6.2, is the only self-conjugate function [Roc70, Page 106].

Example 6.3. Consider the following convex PLQ function which is only quadratic in $x_1$ and has no linear term

$$f(x_1, x_2) = \begin{cases} 
  x_1^2, & \text{if } x_1 \geq 0, x_2 \geq 0; \\
  \infty, & \text{otherwise}.
\end{cases}$$

Its conjugate is

$$f^*(s_1, s_2) = \begin{cases} 
  \frac{1}{4}s_1^2, & \text{if } s_1 \geq 0, s_2 \leq 0; \\
  0, & \text{if } s_1 \leq 0, s_2 \leq 0; \\
  \infty, & \text{otherwise}.
\end{cases}$$

The function and its conjugate are shown in Figure 6.3.

Example 6.4. All of the previously discussed PLQ functions are additively separable functions. Now we consider an additively non-separable PLQ function $f$ defined by

$$f(x_1, x_2) = \begin{cases} 
  x_1^2 + x_2^2 + 2x_1x_2, & \text{if } x_1 \geq 0, x_2 \geq 0; \\
  \infty, & \text{otherwise}.
\end{cases}$$
Chapter 6. Examples

(a) The $l_1$ norm.  
(b) The conjugate of the $l_1$ norm.

Figure 6.1: The $l_1$ norm and its conjugate.

Figure 6.2: 2D Energy function.

The conjugate of this function is,

$$f^*(s_1, s_2) = \begin{cases} 
\frac{1}{4}s_2^2, & \text{if } s_2 \geq 0, s_1 \leq s_2; \\
0, & \text{if } s_1 \leq 0, s_2 \leq 0; \\
\frac{1}{2}s_1^2, & \text{if } s_1 \geq 0, s_1 \geq s_2.
\end{cases}$$

The function and its conjugate are shown in Figure 6.4.

**Example 6.5.** Consider the piecewise function $f$ which is defined by

$$f(x_1, x_2) = \begin{cases} 
\frac{1}{2}(x_1^2 + x_2^2), & \text{if } x_1 \geq 0, x_2 \geq 0; \\
-2x_1 + x_2^2, & \text{if } x_1 \leq 0, x_2 \geq 0; \\
\infty, & \text{otherwise}.
\end{cases}$$
Chapter 6. Examples

(a) The function $f(x_1, x_2)$
(b) The conjugate $f^*(s_1, s_2)$

Figure 6.3: The function from Example 6.3 and its conjugate

(a) The function $f(x_1, x_2)$
(b) The conjugate $f^*(s_1, s_2)$

Figure 6.4: The function from Example 6.4 and its conjugate
Chapter 6. Examples

Figure 6.5: The function from Example 6.5 and its conjugate

It has a strictly convex and merely convex piece. The domain of \( f \) contains two faces, three rays and one vertex. The conjugate of this PLQ function is

\[
f^*(s_1, s_2) = \begin{cases} 
\frac{1}{4}(s_1^2 + s_2^2), & \text{if } s_1 \geq 0, s_2 \geq 0; \\
\frac{1}{4}s_2^2, & \text{if } -2 \leq s_1 \leq 0, s_2 \geq 0; \\
0, & \text{if } -2 \leq s_1 \leq 0, s_2 \leq 0; \\
\frac{1}{4}s_1^2, & \text{if } s_1 \geq 0, s_2 \leq 0; \\
\infty, & \text{otherwise.}
\end{cases}
\]

Its domain has four faces, five rays, one segment and two vertexes. Figure 6.5 illustrates \( f \) and \( f^* \) with the partitions of their domains.

**Example 6.6.** Now we demonstrate another PLQ function with two finite valued pieces whose one piece is quadratic in \( x_1 \) and another one is quadratic.
Chapter 6. Examples

(a) The function $f(x_1, x_2)$

(b) The conjugate $f^*(s_1, s_2)$

Figure 6.6: The function from Example 6.6 and its conjugate

in $x_2$. The function is defined by

$$f(x_1, x_2) = \begin{cases} x_2^2, & \text{if } x_2 \geq 0, x_1 \geq x_2; \\ x_2^2, & \text{if } x_1 \geq 0, x_2 \geq x_1; \\ \infty, & \text{otherwise.} \end{cases}$$

Its conjugate is

$$f^*(s_1, s_2) = \begin{cases} \frac{1}{4}s_1^2 + \frac{1}{4}s_2^2 + \frac{1}{2}s_1 s_2, & \text{if } s_1 \geq 0, s_2 \geq 0; \\ \frac{1}{2}s_2^2, & \text{if } -2 \leq s_1 \leq 0, s_2 \geq 0; \\ 0, & \text{if } s_1 \leq 0, s_2 \leq 0; \\ \frac{1}{4}s_1^2, & \text{if } s_1 \geq 0, s_2 \leq 0. \end{cases}$$

The function and its conjugate are shown in 6.6.

**Example 6.7.** Consider a PLQ function $f$ with four finite valued pieces which is defined by

$$f(x_1, x_2) = \begin{cases} x_1^2 + x_2^2, & \text{if } x_1 \geq 0, x_2 \geq 0; \\ -2x_1 + x_2^2, & \text{if } x_1 \leq 0, x_2 \geq 0; \\ -2x_1 - 2x_2, & \text{if } x_1 \leq 0, x_2 \leq 0; \\ x_1^2 - 2x_2, & \text{if } x_1 \geq 0, x_2 \leq 0. \end{cases}$$

Like the domain of the 1-norm, its domain has four faces, four rays and one
vertex. The conjugate of this PLQ function is defined by

\[ f^*(s_1, s_2) = \begin{cases} 
\frac{1}{4}(s_1^2 + s_2^2), & \text{if } s_1 \geq 0, s_2 \geq 0; \\
\frac{1}{4}s_2^2, & \text{if } -2 \leq s_1 \leq 0, s_2 \geq 0; \\
0, & \text{if } -2 \leq s_1 \leq 0, -2 \leq s_2 \leq 0; \\
\frac{1}{4}s_1^2, & \text{if } s_1 \geq 0, -2 \leq s_2 \leq 0 \\
\infty, & \text{otherwise.}
\end{cases} \]

There are four faces, four rays, four segments and four vertexes in the domain of \( f^* \). Figure 6.7 shows the function and its conjugate. It also illustrates the partitions of the domains.

We also compute the Moreau-envelope of the 2D energy function and the 2D 1-norm with \( \lambda = 1/2 \). For \( \lambda = 1/2 \), the Moreau-envelope of the 2D energy function is written as

\[ e_\lambda f(s_1, s_2) = \frac{1}{3}(s_1^2 + s_2^2). \]

Figure 6.8 illustrates the Moreau-envelope of the 2D energy function. For \( \lambda = 1/2 \), the Moreau-envelope of the 2D 1-norm is computed as

\[ e_\lambda f(s_1, s_2) = \begin{cases} 
s_1 + s_2 - \frac{1}{2}, & \text{if } s_1 \geq \frac{1}{2}, s_2 \geq \frac{1}{2}; \\
-s_1 + s_2 - \frac{1}{2}, & \text{if } s_1 \leq -\frac{1}{2}, s_2 \geq \frac{1}{2}; \\
-s_1 - s_2 - \frac{1}{2}, & \text{if } s_1 \leq -\frac{1}{2}, s_2 \leq -\frac{1}{2}; \\
s_1 - s_2 - \frac{1}{2}, & \text{if } s_1 \geq \frac{1}{2}, s_2 \leq -\frac{1}{2}; \\
s_1^2 + s_2 - \frac{1}{4}, & \text{if } -\frac{1}{2} \leq s_1 \leq \frac{1}{2}, s_2 \geq \frac{1}{2}; \\
-s_1 + s_2^2 - \frac{1}{4}, & \text{if } s_1 \leq -\frac{1}{2}, -\frac{1}{2} \leq s_2 \leq \frac{1}{2}; \\
s_1^2 - s_2 - \frac{1}{4}, & \text{if } -\frac{1}{2} \leq s_1 \leq \frac{1}{2}, s_2 \leq -\frac{1}{2}; \\
s_1^2 + s_2^2, & \text{if } -\frac{1}{2} \leq s_1 \leq \frac{1}{2}, -\frac{1}{2} \leq s_2 \leq \frac{1}{2}.
\end{cases} \]

The Moreau-envelope of 2D 1-norm is shown in Figure 6.9.

Furthermore, we compute the proximal average of 2D 1-norm and 2D energy function with \( \lambda_1 = 1/2 \), \( \lambda_2 = 1/2 \) and \( \mu = 1 \). The proximal average
(a) Partition of the domain of $f$

(b) The function $f(x_1, x_2)$

(c) Partition of the domain of $f^*$

(d) The conjugate $f^*(s_1, s_2)$

Figure 6.7: The function from Example 6.7 and its conjugate
Chapter 6. Examples

Figure 6.8: The Moreau-envelope of 2D energy function with $\lambda = 1/2$.

is computed as

$$p_\mu(f, \lambda)(x_1, x_2) = \begin{cases} 
\frac{1}{4}(x_1^2 + x_2^2) + \frac{1}{4}(x_1 + x_2) - \frac{1}{8}, & \text{if } x_1 \geq \frac{1}{2}, x_2 \geq \frac{1}{2}; \\
\frac{1}{4}(x_1^2 + x_2^2) - \frac{1}{4}(x_1 - x_2) - \frac{1}{8}, & \text{if } x_1 \leq -\frac{1}{2}, x_2 \geq \frac{1}{2}; \\
\frac{1}{4}(x_1^2 + x_2^2) - \frac{1}{8}, & \text{if } x_1 \leq -\frac{1}{2}, x_2 \leq -\frac{1}{2}; \\
\frac{1}{4}(x_1^2 + x_2^2) + \frac{1}{4}(x_1 - x_2) - \frac{1}{8}, & \text{if } x_1 \geq \frac{1}{2}, x_2 \leq -\frac{1}{2}; \\
\frac{1}{4}x_1^2 + \frac{1}{2}x_2^2 + \frac{1}{4}x_1 - \frac{1}{16}, & \text{if } x_1 \geq \frac{1}{2}, -\frac{1}{2} \leq x_2 \leq \frac{1}{2}; \\
\frac{1}{4}x_1^2 + \frac{1}{4}x_2^2 + \frac{1}{4}x_2 - \frac{1}{16}, & \text{if } -\frac{1}{2} \leq x_1 \leq \frac{1}{2}, x_2 \geq \frac{1}{2}; \\
\frac{1}{4}x_1^2 + \frac{1}{4}x_2^2 - \frac{1}{4}x_1 - \frac{1}{16}, & \text{if } x_1 \leq -\frac{1}{2}, -\frac{1}{2} \leq x_2 \leq \frac{1}{2}; \\
\frac{1}{4}x_1^2 + \frac{1}{4}x_2^2 - \frac{1}{4}x_2 - \frac{1}{16}, & \text{if } -\frac{1}{2} \leq x_1 \leq \frac{1}{2}, x_2 \leq -\frac{1}{2}; \\
\frac{1}{2}(x_1^2 + x_2^2), & \text{if } -\frac{1}{2} \leq x_1 \leq \frac{1}{2}, -\frac{1}{2} \leq x_2 \leq \frac{1}{2}. 
\end{cases}$$

Figure 6.10 illustrates the proximal average of of 2D 1-norm and 2D energy function.

All of the above examples deal with PLQ functions with few number of pieces. Now we give an example in which we work with PLQ functions with a large number of pieces. We consider a function $f(x_1, x_2) = x_1^4 + x_2^4$. This is an additively separable function. We assume $f_1(x_1) = x_1^4$ and $f_2(x_2) = x_2^4$. We have $f^*(s_1, s_2) = f_1^*(s_1) + f_2^*(s_2)$. We build a univariate PLQ function
from $f_1$ by using plq.build function of CCA package [CCA] which samples $f_1$ on a grid of points. We use the points from zero to twelve with the interval of one and we get a univariate PLQ function with 13 finite valued pieces. By using this PLQ function we generate a list representation of $f$ as a bivariate PLQ function which has $13^2 = 169$ finite valued pieces. Then we apply our implemented algorithm to compute an approximation of $f^*$ and the computed $f^*$ has 144 finite valued pieces.

To verify our result we also compute an approximation of $f^*$ by using the CCA package. We again use plq.build to generate a univariate PLQ function from $f_1$. Then we apply plq.lft in the CCA package to compute the conjugate $f_1^*$. We obtain $f^*$ which also has 144 finite valued pieces and we store it by using the list representation of bivariate PLQ functions. We compare the coefficients of constrains and function values of the computed conjugate by applying our implemented algorithm and the computed conjugate by applying the CCA package. With a tolerance of $10^{-4}$, this comparison results equality between the computed conjugates. The conjugates are illustrated in Figure 6.11.
Chapter 6. Examples

Figure 6.10: The proximal average of 2D 1-norm and 2D energy function with $\lambda_1 = 1/2$, $\lambda_2 = 1/2$ and $\mu = 1$.

(a) An approximation of conjugate of $f(x_1, x_2) = x_1^4 + x_2^4$ by using our implemented algorithm.

(b) An approximation of conjugate of $f(x_1, x_2) = x_1^4 + x_2^4$ by using the CCA package.

Figure 6.11: The approximation of conjugate of $f(x_1, x_2) = x_1^4 + x_2^4$. 
Chapter 7

Conclusion

We presented Goebel’s proof [Goe00, Proposition 3, page 3] for proving partial conjugates of convex PLQ functions are also PLQ. We focused on adapting parametric quadratic programming (pQP) in computational convex analysis. We studied parametric quadratic optimization problem from [PS11] and we modeled the conjugate and Moreau envelope computation problem of bivariate PLQ functions as a parametric quadratic optimization problem. We proposed the data structures to represent bivariate PLQ functions. We studied an algorithm based on pQP for computing conjugates and Moreau envelopes of convex bivariate PLQ functions and we analyzed the space and time complexity of the algorithm. We also implemented our algorithm in Scilab [Sci94] and we produced some examples of conjugate and Moreau envelope computation.

Although the recent results in parametric minimization of convex PLQ functions work for convex PLQ functions with \( n \) variables, we applied it only for convex bivariate PLQ functions. An interesting future work would be to consider extending the algorithm to functions of more than two variables. The data structure is one of the main issues to be considered. Always new type of entities are present in the domain a \( n \) variate PLQ function which are not present in the domain of a \( n - 1 \) variate PLQ function. One of the challenging parts will be to give an efficient data structure to represent a PLQ function with \( n \) variables.

We know if a function \( f \) is proper convex PLQ then its conjugate \( f^* \) is also a proper convex PLQ function [RW98, Proposition 11.32]. But assume \( f \) is proper PLQ but separately convex. What can we say about its conjugate \( f^* \) and partial conjugate \( f^{*1} \)?

We computed conjugate and Moreau envelope of PLQ functions which are convex. To compute the conjugate, we solved an optimization problem which was not jointly convex. Future research can focus on computing conjugates of multivariate PLQ functions which are not convex.
Bibliography


Bibliography


Bibliography


[CGL] CGLAB a Scilab toolbox for geometry based on CGAL. http://cglab.gforge.inria.fr/. → pages 2


78
Bibliography


[Hau57] F. Hausdorff. Set theory. 1957. \(\rightarrow\) pages 3


Bibliography


81
Bibliography


