Computation of Convex Conjugates in Linear Time using Graph-Matrix Calculus

by

Tasnuva Haque

B.Sc., Rajshahi University of Engineering and Technology (RUET), 2012

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

MASTER OF SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

March 29, 2017

The undersigned certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis entitled:

Computation of Convex Conjugates in Linear Time using Graph-Matrix Calculus

submitted by Tasnuva Haque in partial fulfilment of the requirements of the degree of Master of Science.

Dr. Yves Lucet, Irvin K.Barber School of Arts and Sciences, Unit 5, Computer Science

Supervisor, Professor (please print name and faculty/school above the line)

Dr. Heinz Bauschke, Irvin K.Barber School of Arts and Sciences, Unit 5, Mathematics

Supervisory Committee Member, Professor (please print name and faculty/school above the line)

Dr. Shawn Wang, Irvin K.Barber School of Arts and Sciences, Unit 5, Mathematics

Supervisory Committee Member, Professor (please print name and faculty/school above the line)

Dr. Abbas S. Milani, Faculty of Applied Science, School of Engineering

University Examiner, Professor (please print name and faculty/school above the line)

March 29, 2017

(Date Submitted to Grad Studies)

Abstract

Computational Convex Analysis focuses on computing the convex operators which are used very often in convex analysis. The Fenchel conjugate is one of the most frequently used convex operator. The objective of this thesis is to develop an algorithm for computing the conjugate of a bivariate Piecewise Linear-Quadratic (PLQ) function. Although some algorithms already exist for computing the conjugate of a bivariate PLQ function, their worst-case time complexity is not linear. Our challenge is to improve the worst case time complexity to linear.

We use a planar graph to represent the entities of a PLQ function. Each node of the graph contains a GPH matrix which represents an entity of the PLQ function . In addition, we store the adjacency information and type of all entities. We traverse the graph using breadth first search and compute the conjugate of each entity. Moreover we store the information of visited entities using a binary flag. So we never need loop through all entities to check whether it is already visited. As a result we get linear computation time in the worst case.

We perform numerical experiment which confirms that our algorithm is running in linear-time. We provide a comparison of the performance of this algorithm with the previous algorithms. Finally we explained how to extend this algorithm in higher dimensions while keeping the worst-case time complexity linear.

Table of Contents

List of Tables

List of Figures

Acknowledgment

In the name of Allah, the Most Gracious and the Most Merciful.

Alhamdulillah, all praises to Allah for giving me the opportunity, the strength and the patience to complete this thesis and overcome all the challenges and difficulties.

I would like to express my sincere gratitude to my supervisor, Dr. Yves Lucet, for the guidance, encouragement and advice he has provided throughout the progression of my graduate studies. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. His positive outlook and confidence in my research inspired me and gave me confidence.

I would also like to extend my enduring gratitude to the members of my examination committee: Dr. Heinz Bauschke and Dr. Shawn Wang for their insightful comments on my thesis. I am also thankful to Dr. Abbas S. Milani, my university examiner, for spending his valuable time reading my thesis and providing constructive feedback. Thanks to the committee members for letting my defense be an enjoyable moment.

I truly appreciate the support of all amazing colleagues and friends of my research group. I am really proud to be a part of this research group. I would also like to thank the Bangladesh community in Kelowna for their efforts and selfless care.

I would like to take this opportunity to acknowledge and express my gratitude towards my parents (Md. Manowarul Haque and Fouzia Jesmin), parents-in-law, sister (Tasnima Haque) and other family members for their continuous prayers, inspiration and support. I owe thanks to a very special person, my husband, Md. Nobinur Rahman, for his unconditional support and understanding during this journey that made the completion of the thesis possible. Thanks to my baby, Nuayma, for being one of my sources of inspiration in completing this thesis.

Finally, I am also thankful to the University of British Columbia, Okanagan and Natural Sciences and Engineering Research Council of Canada (NSERC) for providing financial support.

Dedication

To my husband
Md. Nobinur Rahman
&
my daughter
Nuayma Mahreen Rahman.

Chapter 1

Introduction

Computational Convex Analysis (CCA) introduces several algorithms for studying operators commonly encountered in Convex Analysis. Among them the addition, the scalar multiplication, the Moreau envelope (also known as Moreau-Yosida approximate or Moreau-Yosida regularization), the (Legendre-Fenchel) conjugate are considered as the core convex operators [GL13, Luc10]. These operators are used to develop new theories and results in convex optimization [Luc13]. One application of CCA is Computer-Aided Convex Analysis, which focuses on visualizing these operators when applied to functions of typically one or two variables.

Computational Convex Analysis has numerous applications in the field of science and engineering such as network flow, electrical networks, robot navigations, image processing, computer vision, partial differential equations, network communications etc. For example, in image processing, convex operators are used to compute the Euclidian distance transform, generalized distance transform and morphological operators [Luc10]. The algorithms for computing the Euclidian distance transform are developed based on the concept of the Legendre-fenchel transform and the Moreau envelope. These algorithms achieve linear-time complexity by combining separability of the Euclidian distance transform with convex properties [Luc09]. See [Luc10] for more details on how CCA facilitates all of these applications.

Convex analysis has several advantages over nonconvex analysis. One of the advantages of a convex function is that the global optimality is guaranteed by local optimality. Nonsmooth functions can be manipulated using convex calculus. Moreover several efficient numerical algorithms are available.

To calculate the convex operators and perform several transforms by implementing convex algorithms, a Computational Convex Analysis (CCA) toolbox has been developed. The current version of the CCA toolbox has been built by using Scilab [SCI], a free and open source software. The CCA toolbox is almost complete for univariate functions.

1.1 Motivation

In this thesis we compute the convex conjugate which is one of the most frequently used convex operator. Convex conjugate functions play a significant role in *duality*. Consider the primal optimization problem

$$p = \inf_{x \in \mathbb{R}^d} [f(x) + g(Ax)].$$

The dual optimization problem is

$$d = \sup_{y \in \mathbb{R}^d} [-f^*(A^T y) - g^*(-y)].$$

Here f^* is the *Fenchel conjugate* of f. Fenchel's duality theorem states that under some constraint qualification conditions, p=d and the solution of one problem can be computer from the solution of the other. Computing the conjugate of a function is one of the key step to solve a dual optimization problem [Her16].

There is a close relationship between the conjugate and the Moreau envelope [Luc06, HUL07]. The Moreau envelope of a convex function is

$$M_\lambda(s) = \inf_{x \in \mathbb{R}^d} [f(x) + \frac{||x - s||^2}{2\lambda}].$$

One of the important properties of the Moreau envelope is regularization property [Luc10]. The Moreau envelope regularize a nonsmooth function while keeping the same local minima [PW16]. The set of point at which the infimum is found is called the proximal mapping which is the basis of many convex and nonconvex optimization techniques e.g. the bundle method.

The application of conjugate function discussed above inspired us to develop an algorithm to compute the convex conjugate. Although some algorithms are available for computing the conjugate of the univariate or bivariate convex functions, we are interested in improving their performance. In Section 1.2 we discuss the performance of available algorithms for the univariate function.

1.2 Algorithms for computing the conjugate of an univariate convex function

The field of Computational Convex Analysis was born with developing the algorithms to compute the Legendre-Fenchel transform, see also references in [Luc10] for a more complete history. The Fenchel conjugate operator, one of the most fundamental convex operators, is used to calculate the dual properties of a convex function [Luc10]. Symbolic computation algorithms have been developed for computing the conjugate operator of univariate or multivariate convex functions. These algorithms involve calculating the Fenchel conjugate by differentiating a function under the supremum and computing an equation which is satisfied by all the critical points. However, the problem is that sometimes the computations of the critical points involves a nonlinear equation that is difficult or impossible to solve symbolically e.g. polynomial of degree 5 or higher degree.

The fast Legendre transform (FLT) algorithms were developed for computing the conjugate using piecewise-linear approximations [Luc97, Luc13]. The FLT algorithms are also used to compute the Moreau envelope and the proximal average. The Moreau envelope can be computed from the conjugate operator and it converts a nonsmooth optimization problem into a smooth problem without changing the minimum of the problem [Luc06].

The worst-case time complexity of FLT algorithms was log-linear. The time complexity of the FLT algorithms was later improved by the Linear-time Legendre transform (LLT) algorithm, which runs in linear time [Luc97].

The LLT algorithms have a wide range of applications in network communication, robotics, numeric simulation of multiphasic flows, pattern recognition, and computation of distance transforms [Luc10]. Unfortunately, because of the curse of dimensionality the application of FLT and LLT algorithms are limited to a small number of variables [GL13].

The Parabolic Envelope (PE) algorithm was developed as an alternative to the LLT algorithms to compute the conjugate and the Moreau envelope [FH12, Luc10]. However the efficiency of the PE and the LLT algorithms are same.

A new class of algorithm based on Piecewise Linear-Quadratic (PLQ) functions was developed to remove the restrictions of the symbolic algorithm and the fast algorithms [LBT09]. A function is called a PLQ function if its domain can be partitioned into a finite number of polyhedral sets, on each of which the function is either linear or quadratic [RW09]. PLQ functions have some interesting properties. Consider a convex univariate PLQ function $f(x)$ with k pieces,

$$f(x) = \begin{cases} Q_0x^2 + q_0x + \alpha_0, & \text{if } x \le x_0, \\ Q_1x^2 + q_1x + \alpha_1, & \text{if } x_0 \le x \le x_1, \\ \vdots & \vdots \\ Q_{k-1}x^2 + q_{k-1}x + \alpha_{k-1}, & \text{if } x_{k-2} \le x \le x_{k-1}, \\ Q_kx^2 + q_kx + \alpha_k, & \text{otherwise,} \end{cases}$$

where, Q is the quadratic term, q is the linear term and α is the constant term. It is stored using a PLQ matrix as follows

$$\text{PLQ matrix of } f(x) = \begin{bmatrix} x_0 & Q_0 & q_0 & \alpha_0 \\ x_1 & Q_1 & q_1 & \alpha_1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{k-1} & Q_{k-1} & q_{k-1} & \alpha_{k-1} \\ +\infty & Q_k & q_k & \alpha_k \end{bmatrix}.$$

If convex operators like the conjugate, the Moreau envelope or the proximal average are applied to a PLQ function then the resulting function is also a PLQ function [Kha13]. PLQ functions are closed under all convex operators because they allow quadratic pieces [Luc13, Luc10, GKL14]. Another advantage of the PLQ functions is that the representation and the evaluation of a PLQ function is simple. PLQ algorithms are as efficient as the fast algorithms but provide greater precision. Usually PLQ based algorithms have less numeric errors than fast algorithms especially with unbounded domain. In addition, efficient PLQ algorithms are available for the standard convex univariate operators.

Another class of algorithms for PLQ functions, called GPH algorithms, is based on graph-matrix calculus [Goe08]. The idea of GPH algorithms comes from the linear relation between the graph of the subdifferential and the graph of the function of a convex operator [GL11, Kha13]. GPH algorithms are used to compute the graph of a transform. These algorithms store the graph of the subdifferential and the function value at a finite set of points [Luc13].

The GPH representation of $f(x)$ is,

$$\text{GPH matrix of } f(x) = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_k & x_{k+1} \\ s_0 & s_1 & s_2 & \cdots & s_k & s_{k+1} \\ y_0 & y_1 & y_2 & \cdots & y_k & y_{k+1} \end{bmatrix}$$

where x_i are the points in the domain of f, s_i are the subgradients of f at x_i and y_i is the function value at x_i.

The advantage of using the GPH matrix over the PLQ matrix is that a GPH matrix stores the graph of the subdifferential of a PLQ function which simplifies the computation of the most common convex operators [GL11]. The performance of GPH algorithms is similar to PLQ algorithms (linear-time) but they are simpler. The GPH algorithms become especially efficient when implemented in matrix-based mathematical software like MATLAB or Scilab. For example, the computation of the convex conjugate operator becomes a matrix multiplication operation. Consider the following GPH matrix representation of $f(x)$,

$$\text{GPH matrix of } f(x) = \begin{bmatrix} x \\ s \\ y \end{bmatrix}. \tag{1.1}$$

According to Goebel's Graph-Matrix calculus rules [Goe08], the graph of the subdifferential of the conjugate of f is defined by the following formula,

$$\text{gph } \partial(f^*) = \begin{bmatrix} 0 & Id \\ Id & 0 \end{bmatrix} \text{gph } \partial(f).$$

Applying this formula, the GPH matrix of the conjugate of f, denoted by f^*, is,

$$\begin{aligned} \text{GPH matrix of } f^*(x) &= \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} x \\ s \end{bmatrix} \\ s^T x - y \end{bmatrix}, \\ &= \begin{bmatrix} s \\ x \\ s^T x - y \end{bmatrix}, \end{aligned} \tag{1.2}$$

where $*$ is used for the standard matrix multiplication.

The performance of the CCA toolbox can be extended by including the algorithms for computing convex operators in higher dimension. For example, algorithms for computing the conjugate of a bivariate convex function are already available . In the next section we briefly discuss their performance and the limitations.

1.3 Algorithms for computing the conjugate of a bivariate convex function

The first algorithm to compute the conjugate of a convex bivariate PLQ function was developed in [GL13]. The algorithm is implemented in Scilab using the Computational Geometry Library CGAL [CGA]. In this algorithm, a *planar arrangement* is used to store the entities (vertices, edges and faces) of a PLQ function. The *dual arrangement* is computed by looping through the entities of the planar arrangement.

The key idea of the algorithm proposed in [GL13] is to loop through the vertices and compute the conjugate of the edges associated with a vertex.

For every primal edge, the algorithm loops through all the gradients associated with an edge and compute its dual vertex with a vertex index. The algorithm never creates duplicate vertices and edges. However one of the challenges is to compute whether a vertex is duplicate or not and assign an index if it is unique.

A binary search tree is used to detect the duplicate points which has $\mathcal{O}(N \log N)$ worst-case time complexity. This algorithm can achieve a linear-time expected-case complexity if it is implemented using a hash table or a linear-time worst-case time complexity for a worst-case trie (assuming bounded bit length).

For a convex bivariate PLQ function, the full conjugate can be computed using the partial conjugate. The partial conjugate is the conjugate with respect to one of the variables. Partial conjugates are also PLQ functions but not necessarily convex [Roc70] e.g. the partial conjugate of the l_1 norm $l_1(x_1, x_2) = |x_1| + |x_2|$ is $(l_1)^*_1(s_1, x_2) = \iota_{[-1,1]}(s_1) - |x_2|$. In [GKL14], the full conjugate of a bivariate PLQ function is determined more easily by using two partial conjugates. However, the complexity of the full conjugate computation algorithm and the partial conjugate computation algorithm is the same.

Another algorithm, based on parametric programming, is developed in [Kha13] to compute the conjugate of convex bivariate PLQ functions. Parametric programming is a method for computing the effect of perturbations in mathematical programs. Optimization problems that depend on one scalar parameter are called parametric programs while problems that depend on several parameters are called multi-parametric programs. Solving them involves computing the solution of an optimization problem for every value in the parameter space [BBM03, PS11]. A parametric programming problem is categorized as a parametric linear program (pLP), a parametric quadratic program (pQP), a parametric non-Linear program (pNLP), a parametric linear complementary problem (pLCP), etc.

The algorithm in [Kha13] combines a Parametric Quadratic programming (pQP) approach with computational convex analysis. The input and output PLQ function is stored using a list representation which is internally converted into a face-constraints adjacency representation. In this algorithm a planar graph is used to store the entities of a PLQ function and adjacency information of the entities is not stored.

The algorithm computes the vertices using Mulmuley's segment intersection algorithm [Mul88] which requires a log-linear time complexity in the worst-case. The time required to compute the remaining entities i.e. rays and segment is log-quadratic because for every vertex, the algorithm loops through all adjacent edges which requires log-quadratic time in the worst-case. However, according to [Kha13, Theorem 5.7], the worst-case time complexity can be improved by using a *half edge* data structure as this data contains the ordering information of the vertices. This algorithm still needs to detect the duplicate entities, which takes log-linear time.

In total, the time required to compute the conjugate in [Kha13] without breaking the similarity of input and output data structure is log-linear.

A summary of the algorithms for computing the conjugate of a bivariate PLQ function is presented in Table 1.1.

Table 1.1: The worst-case performance of the algorithms developed for computing the conjugate of a bivariate PLQ function.

Algorithm	Source	Time	Data Structure	Space
Full conjugate (geometric alg.)	[GL13]	Log-linear	Red-Black Tree	Linear
		Linear	Trie (bit length is bounded)	Linear
		Expected Linear	Hash table	Linear
Full conjugate (parametric opt.)	[Kha13]	Log-linear	List	Linear
Partial conjugate (geometric alg.)	[GKL14]	Log-linear	Red-Black Tree	Linear

From Table 1.1, we can conclude that no algorithm is known so far to compute the conjugate of a bivariate PLQ function in linear-time.

In this thesis, we present a new algorithm to compute the conjugate of a bivariate PLQ function using graph-matrix calculus and fully leveraging the neighbour graph. To our knowledge, this algorithm is the first linear-time algorithm to compute the conjugate of a bivariate PLQ function. We implement our algorithm in Scilab [SCI] for bivariate PLQ functions.

This thesis is organized as follows. We discuss the background and motivation of computing the convex conjugate operator in Chapter 1. All the basic notations and definitions required to explain our algorithm are included in Chapter 2. Our proposed algorithm is explained with an example in Chapter 3. In this chapter, we also include a detailed explanation of the data structure we used and the complexity of the algorithm. Results of the numerical experiments are included in Chapter 4. Chapter 5 concludes the thesis.

Chapter 2

Preliminaries

In this chapter, we discuss some basic definitions, properties and examples required to explain our algorithm.

Definition 2.1. [Effective domain] Consider $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$. The effective domain of a function f, denoted by dom f is,

$$\text{dom } f = \{x \in \mathbb{R}^d : f(x) < +\infty\}.$$

Definition 2.2. [Proper function] A function f is said to be a proper function if dom$f \neq \emptyset$ and $f(x) > -\infty$ for all $x \in \mathbb{R}^d$.

Definition 2.3. [Convex function] A proper function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is convex if its domain is a convex set and for any two points $x_1, x_2 \in \text{dom } f$ and $\theta \in [0, 1]$

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta) f(x_2).$$

Figure 2.1 illustrates an example of a convex function in which the line segment passing through any two points of dom f lies above or on the graph of the function.

Definition 2.4. [Convex Hull] Consider a set $C \subset \mathbb{R}^d$. The closed convex hull of C is the smallest closed convex set that contains C.

The closed convex hull of C is denoted by co (C) and can be computed as the closure of

$$\text{co } (C) = \{\sum_{i=0}^{n} \lambda_i x_i : x_i \in C, \lambda_i \geq 0, \sum_{i=0}^{n} \lambda_i = 1, n \geq 0\}.$$

Definition 2.5. [Lower semi-continuous function] A function $f : \mathbb{R}^d \to \mathbb{R} \cup \{-\infty, +\infty\}$ is said to be lower semi-continuous (l.s.c) if,

$$\lim_{x \to \bar{x}} \inf f(x) \geq f(\bar{x}) \text{ for all } \bar{x} \in \mathbb{R}^d.$$

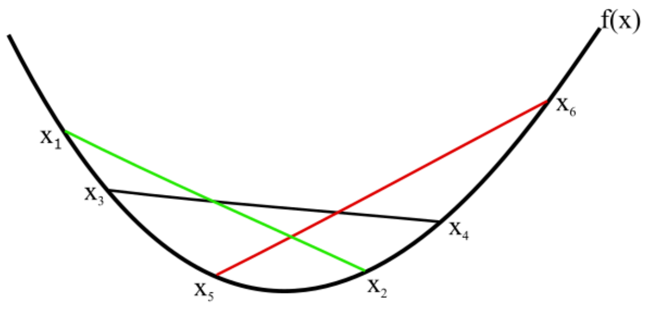


Figure 2.1: A convex function.

Definition 2.6. [Additively separable function] A function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ of n variables is called an additively separable function if for some single variable functions $f_1(x_1), f_2(x_2), \cdots, f_n(x_n)$ it can be represented as

$$f(x_1, x_2, \cdots, x_n) = f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n).$$

Otherwise the function f is said to be an additively non-separable function.

2.1 Polyhedral set

Definition 2.7. [Polyhedral set] A polyhedral set in $\mathbb{R}^d$ is the intersection of finitely many half-spaces.

It is usually denoted as $C = \{x \in \mathbb{R}^d : {a_i}^T x \leq b_i\}$ where $a_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}, i = 1, \cdots, m$. We have the following definitions regarding $\mathcal{C}$.

Definition 2.8. [Relative interior] The relative interior of a set C, denoted by ri C, is the interior of the set C with respect to the smallest affine set containing C.

Definition 2.9. [Affinely independent] A set $X \subseteq \mathbb{R}^d, X \neq \emptyset$ is said to be affinely independent if no vector $x \in X$ is expressible as an affine combination of the vectors in $X \setminus \{x\}$ and can not be represented as $\sum_{i=1}^{n} \alpha_i x_i = 0$ where the constants α_i are not all zeros.

Definition 2.10. [Dimension of a polyhedral set] The dimension of a polyhedral set C, denoted by $\dim C$, is the maximum number of affinely independent points in C minus 1.

For example, if C consists of a single point then $\dim C = 0$ and if C is full dimensional then $\dim C = d$.

Definition 2.11. [Proper face, Improper face] A face of a convex set C is a nonempty subset, F, of C with the property that if $x_1, x_2 \in C, \theta \in (0,1)$ and $\theta x_1 + (1-\theta)x_2 \in F$ then $x_1, x_2 \in F$. A face, F, that is strictly smaller than C is called a proper face. Otherwise F is said to be an improper face.

Definition 2.12. [Polyhedral decomposition] The set $\mathcal{C} = \{C_k : k \in \mathcal{K}\}$, where $\mathcal{K}$ is a finite index set, is called a polyhedral decomposition of $\mathcal{D} \subseteq \mathbb{R}^d$ if it satisfies the following conditions

(i) all of its members C_k are polyhedral sets,

(ii) $\bigcup_{k\in\mathcal{K}} C_k = \mathcal{D}$,

(iii) for all $k \in \mathcal{K}$, $\dim C_k = \dim \mathcal{D}$,

(iv) ri $C_{k_1} \cap$ ri $C_{k_2} = \emptyset$, where $k_1, k_2 \in \mathcal{K}, k_1 \neq k_2$.

Definition 2.13. [Polyhedral subdivision] The set $\mathcal{C}$ is a polyhedral subdivision if $\mathcal{C}$ is a polyhedral decomposition and the intersection of any two members of $\mathcal{C}$ is either empty or a common proper face of both.

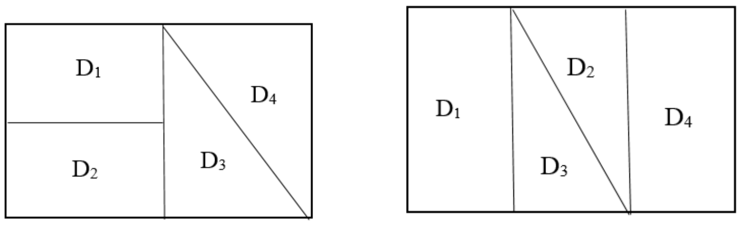
Example 2.14. Let $\mathcal{C} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$ with $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4 = D$ be a collection of polyhedral sets. In Figure 2.2(a), the collection $\mathcal{C}$ is a polyhedral decomposition but not a polyhedral subdivision because $\mathcal{D}_1 \cap \mathcal{D}_3$ or $\mathcal{D}_2 \cap \mathcal{D}_3$ is not empty or a common proper face of both. In Figure 2.2(b), the collection $\mathcal{C}$ is a polyhedral decomposition and a polyhedral subdivision of $\mathcal{D}$.

2.2 Piecewise Linear-Quadratic function

Definition 2.15. [Piecewise Linear-Quadratic (PLQ) function] A function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is a Piecewise Linear-Quadratic (PLQ) function if its domain can be represented as the union of finitely many polyhedral sets and on each polyhedral set the function is either linear or quadratic [RW09].

The domain of a PLQ function can be decomposed into a polyhedral decomposition and on each piece, $f(x) = f_k(x)$ if $x \in C_k, k \in \mathcal{K}$, where

$$f_k(x) = \frac{1}{2}x^T Q_k x + {q_k}^T x + \alpha_k \tag{2.1}$$



(a) A polyhedral decomposition but not a polyhedral subdivision.

(b) A polyhedral subdivision.

Figure 2.2: Examples of a polyhedral decomposition and polyhedral subdivision.

with $Q_k \in \mathbb{R}^{d\times d}$ a symmetric matrix, $q_k \in \mathbb{R}^d$ and $\alpha_k \in \mathbb{R}$. For each piece of the PLQ function f, we associate the function $\tilde{f}_k = f_k + \delta_{C_k}$ where δ_{C_k} is the indicator function defined as

$$\delta_{C_k} = \begin{cases} 0, & \text{if } x \in C_k; \\ \infty, & \text{if } x \notin C_k. \end{cases}$$

Example 2.16. Consider the l_1 norm function,

$$\begin{aligned} f(x_1, x_2) =& |x_1| + |x_2| \\ =& \begin{cases} x_1 + x_2, & \text{if } -x_1 \le 0, -x_2 \le 0; \\ -x_1 + x_2, & \text{if } x_1 \le 0, -x_2 \le 0; \\ -x_1 - x_2, & \text{if } x_1 \le 0, x_2 \le 0; \\ x_1 - x_2, & \text{if } -x_1 \le 0, x_2 \le 0; \end{cases} \end{aligned}$$

This example is illustrated in Figure 2.3. The domain of the l_1 norm function is partitioned into four polyhedral set and on each polyhedral set the function is linear.

Definition 2.17. [Partition] A family of sets $\mathcal{P} = \{C_k : k \in \mathcal{K}\}$ is a partition of $\mathcal{D}$ if

(i) $\forall k \in \mathcal{K}, C_k \neq \emptyset$,

(ii) $\mathcal{D} = \cup_{k\in\mathcal{K}} C_k$,

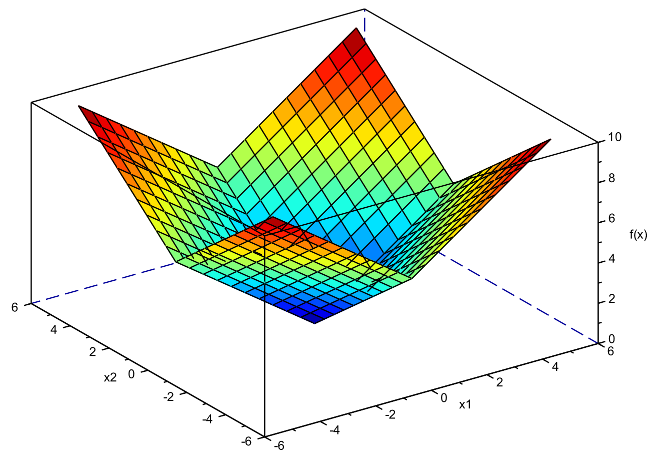


Figure 2.3: Example of a bivariate PLQ function - the l_1 norm function.

(iii) $C_k \cap C_l = \emptyset$ for $k \neq l$.

Fact 2.18. *(Partition of a convex PLQ function [Roc70, Theorem 18.2]) Let $\mathcal{C}$ be a non-empty convex set and $\mathcal{F}(\mathcal{C})$ be the collection of all non-empty faces of $\mathcal{C}$ and $\mathcal{RI}(\mathcal{C}) = \{\text{ri } F : F \in \mathcal{F}(\mathcal{C})\}$ be the collection of all relative interiors of non empty faces of $\mathcal{C}$. Then $\mathcal{RI}(\mathcal{C})$ is a partition of $\mathcal{C}$.*

Definition 2.19. [Entity] Assume f is a PLQ function and $\cup_k C_k =$ dom f is a polyhedral subdivision that induce a partition of dom f. An entity is an element of the partition of the domain. For $f : \mathbb{R}^2 \to \mathbb{R} \cup \{+\infty\}$, an entity is either a vertex, an edge or a face [GL13].

Definition 2.20. [Vertex] If the dimension of an entity is 0 then it is called a vertex. Figure 2.4 is the example of vertex .

Definition 2.21. [Edge] If the dimension of an entity is 1 then it is called an edge. An edge for any $x_1 \neq x_2$ is defined as

$$\mathcal{E} = \{x \in \mathbb{R}^d : x = \lambda_1 x_1 + \lambda_2 x_2, \lambda_1 + \lambda_2 = 1, \lambda = (\lambda_1, \lambda_2) \in \Lambda\}. \tag{2.2}$$

In $\mathbb{R}^2$, an edge is classified as follows

(i) Line: when $\Lambda = \mathbb{R}^2$.

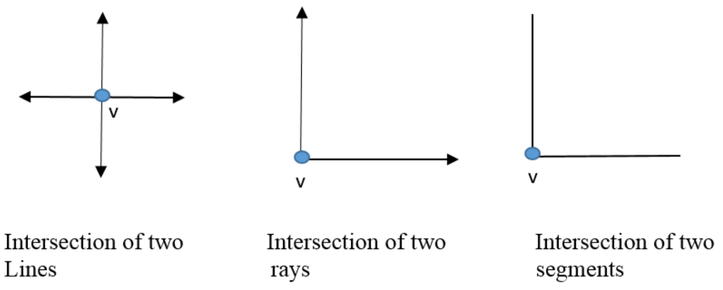


Figure 2.4: Vertex in $\mathbb{R}^2$.

(ii) Ray: when $\Lambda = \mathbb{R}_+ \times \mathbb{R}$ where $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$.

(iii) Segment: when $\Lambda = \mathbb{R}_+ \times \mathbb{R}_+$.

Definition 2.22. [Face] If an entity has a nonempty interior then it is called a face [Kha13]. For a PLQ function, a face Q_k is categorized as

(i) a QQ face if the associated function is quadratic with Q_k a symmetric positive definite matrix.

(ii) a QL face if the associated function is quadratic with Q_k a symmetric positive semi-definite matrix with exactly one positive and one zero eigenvalues.

(iii) a LL face if the associated function is linear.

Definition 2.23. [Extreme point, [Fan63]] An extreme point of a convex set C, is a point $x \in C$, which has a property that if $x = \theta x_1 + (1-\theta)x_2$ with $x_1, x_2 \in C$ and $\theta \in [0,1]$, then $x_1 = x$ and/or $x_2 = x$.

An extreme point of C is not in the relative interior of any line segment lying entirely in C. An example of extreme point is showed in Figure 2.5.

2.3 Subdifferential of a function

Definition 2.24. [Subgradient] The subgradient of a function f at a point $\bar{x} \in \operatorname{dom} f$, is a vector s such that

$$f(x) \geq f(\bar{x}) + \langle s, x - \bar{x} \rangle,$$

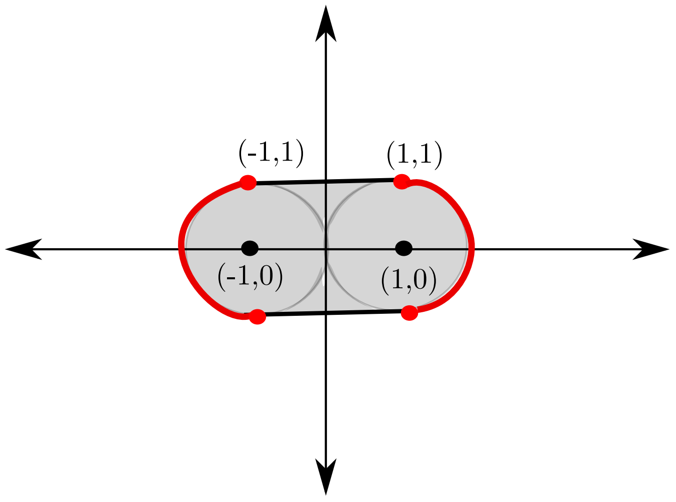


Figure 2.5: Red points are the extreme points for the set $S = \text{co } B(-1,0) \cup B(1,0)$ where $B(1,0)$ is the closed ball of radius 1 centered at $(1,0)$.

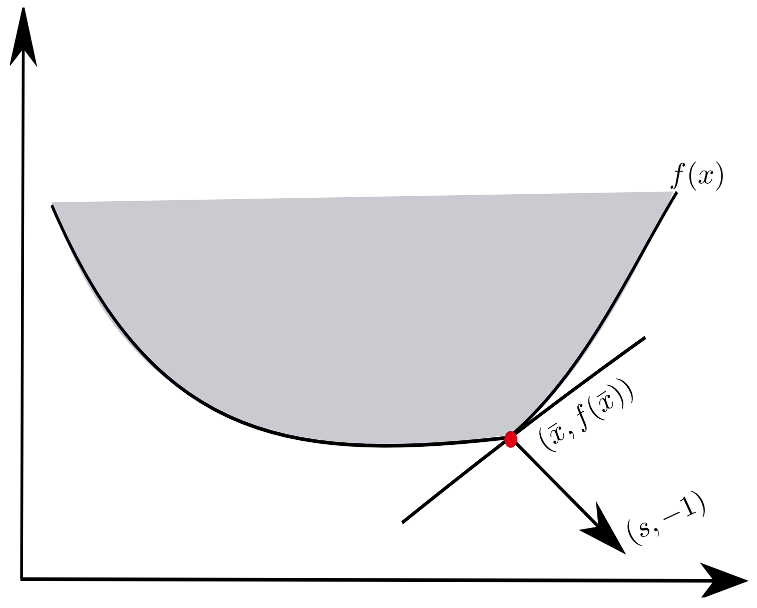


Figure 2.6: The subgradient of a function.

for all $x \in \mathbb{R}^d$. The vector s is said to be a subgradient of f at $\bar{x}$. It creates a supporting hyperplane which has normal $(s, -1)$ and goes through the point $(\bar{x}, f(\bar{x}))$ of the epigraph of f [RW09, Theorem 8.9].

If the function f is convex and differentiable then f has a unique subgradient at a point $\bar{x}$ and it is the gradient . If f is convex but nondifferentiable at $\bar{x} \in$ int dom f then f has multiple subgradients at $\bar{x}$. This is illustrated in Figure 2.7. A subgradient of the PLQ function f, defined by Equation (2.1), at a point x is

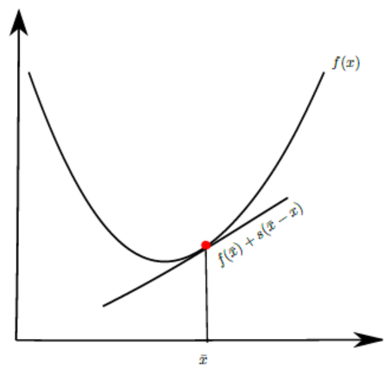
$$s = \nabla f_k(x) = Q_k x + q_k. \tag{2.3}$$

Definition 2.25. [Subdifferential[Roc70, RW09, MN15]] The subdifferential of a function f at a point $\bar{x}$ is a closed convex set which is the collection of all subgradients of f at $\bar{x}$. It is represented by the notation $\partial f(x)$ and defined as

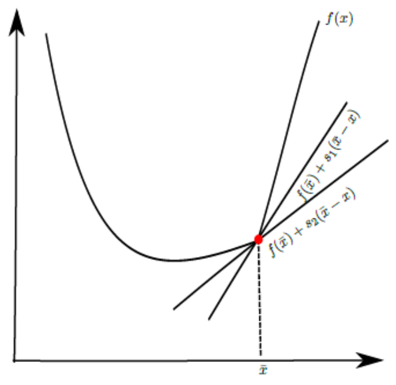
$$\partial f(x) = \{s \in \mathbb{R}^d : f(x) \geq f(\bar{x}) + \langle s, x - \bar{x} \rangle, \forall x \in \mathbb{R}^d\},$$

when $x \in \operatorname{dom} f$. If $x \notin \operatorname{dom} f$ by convention we set $\partial f(x) = \emptyset$ [Bal10].

If the function is convex and differentiable at $x \in$ int dom$f$, then $\partial f(x) = \{\nabla f(x)\}$. If a PLQ function f is not differentiable at x then its subdiffer-



(a) The function $f(x)$ is differentiable at a point x and has a subgradient s.



(b) The function is not differentiable at x and has many subgradients at x. Two subgradients e.g. s_1 and s_2 are shown.

Figure 2.7: The subgradient of a convex function.

ential is

$$\partial f(x) = \text{co } \{\nabla f_k(x) : C_k \ni x\}.$$

For example, in Figure 2.7(a), the subdifferential is single valued i.e. $\partial f(x) = \{s\}$ but in Figure 2.7(b), it is multi-valued i.e. $\partial f(x) = [s_1, s_2]$.

Consider the example of the absolute value function

$$\begin{aligned} f(x) =& |x| \\ =& \begin{cases} -x, & \text{if } x \leq 0; \\ x, & \text{if } x > 0; \end{cases} \end{aligned}$$

The absolute value function and its subdifferential is presented in Figure 2.8

The subdifferential at any $x > 0$ is $\partial f(x) = \{\nabla f(x)\} = \{1\}$ and the subdifferential at any $x < 0$ is, $\partial f(x) = \{\nabla f(x)\} = \{-1\}$. The function is not differentiable at $x = 0$. The subdifferential at $x = 0$ is $\partial f(0) = \text{co } \{-1, 1\} = [-1, 1]$.

The graph of the subdifferential of f is,

$$\text{gph } \partial f = \{(x, s) : x \in \text{dom } f, s \in \partial f(x)\}.$$

Definition 2.26. [Conjugate function] Consider a proper convex lower semi-continuous function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$. The conjugate of f, denoted by

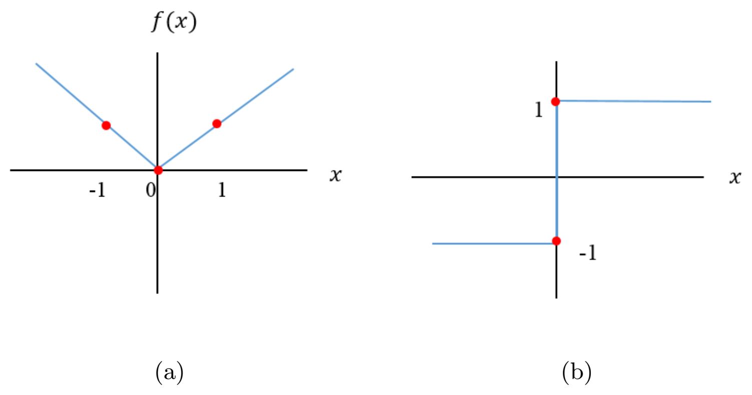


Figure 2.8: (a)The absolute value function and (b) the subdifferential $\partial f(x)$ of the absolute value function.

f^*, is defined as

$$f^*(s) = \sup_x(\langle s, x\rangle - f(x)). \tag{2.4}$$

Consider the geometric interpretation of the conjugate function which is illustrated in Figure 2.9. We want to compute a point x such that the slope of the line segment passing through the point $(x, f(x))$ has the maximum intercept on the s axis.

The affine functions that are less than $f(x)$ are represented by the following inequalities,

$$\begin{aligned} &\langle s, x\rangle - \alpha \leq f(x), \\ \Leftrightarrow\ &\langle s, x\rangle - f(x) \leq \alpha. \end{aligned}$$

The smallest value of α is found at $f^*(s)$ which is the conjugate of $f(x)$.

2.4 Planar graph

Definition 2.27. [Planar graph] A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be a planar graph if it can be drawn in a plane with no two edges crossing each other except at a vertex to which they are incident.

Examples of a planar and a non planar graph are presented in Figure 2.10.

Definition 2.28. [Degree] In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the degree of a node $v \in \mathcal{V}$ is the number of edges $e \in \mathcal{E}$ that are connected to v.

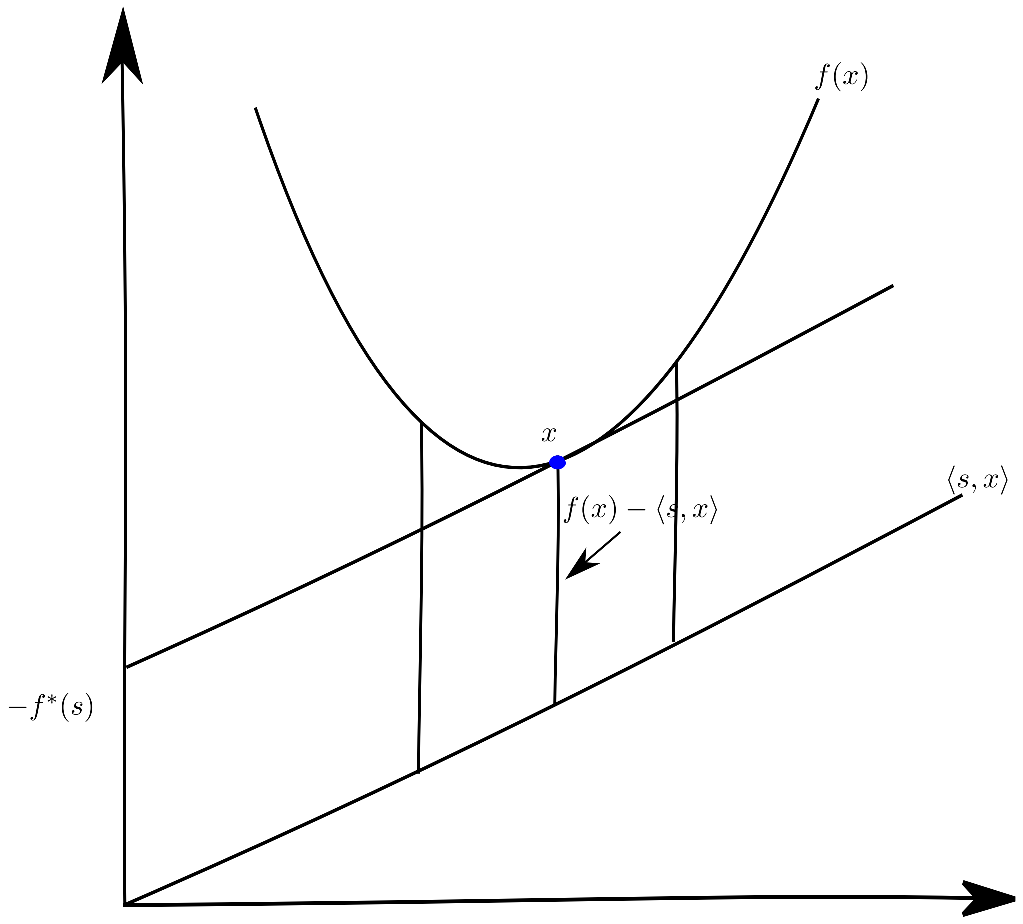
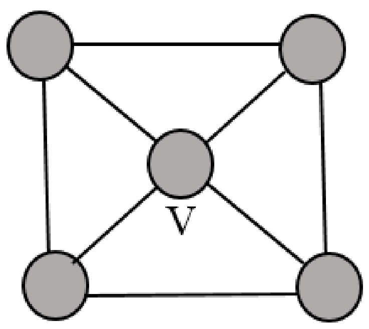
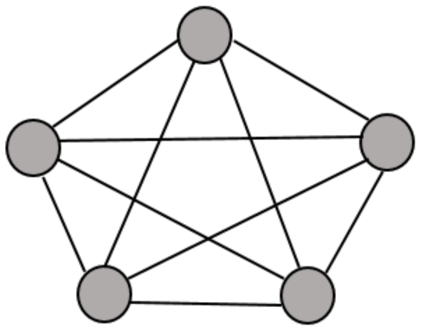


Figure 2.9: Geometric interpretation of a conjugate function



(a) A planar graph.



(b) A non planar graph.

Figure 2.10: Example of planar and non-planar graph.

For example, in Figure 2.10(a) the degree of the node V is 4.

Fact 2.29. *(Euler's Formula) For a connected planar graph $\mathcal{G}$ with n_v vertices, n_e edges and n_f faces, the following must hold:*

$$n_v - n_e + n_f = 2.$$

Chapter 3

Algorithm

In this chapter, we explain our algorithm for computing the conjugate of a proper convex l.s.c. bivariate PLQ function in linear-time using graph-matrix calculus.

3.1 Data structure

The input for our algorithm is a graph, called the *entity graph*, and specific information on each entity. The entity graph is built from a proper convex l.s.c PLQ function. Each node of the entity graph represents an entity of the PLQ function. From now we always assume the PLQ function is a polyhedral subdivision.

Example 3.1. Consider the example of the l_1 norm, $f(x_1, x_2) = |x_1| + |x_2|$. The partition of its domain is illustrated in Figure 3.1. This function has nine entities: four faces, four rays and one vertex.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the entity graph where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node of the set $\mathcal{V}$ represents an entity. When two entities are adjacent then we connect them using an edge. Figure 3.2 is the entity graph corresponding to the l_1 norm function.

In addition, associated with each node of $\mathcal{G}$ we store the following information about an entity: the GPH matrix, the adjacent entities and the entity type.

3.1.1 GPH matrix

We use GPH matrices to represent a PLQ function. For each entity, we need to store enough information so that we can completely recover it from the GPH matrix. We select some points from dom f, compute the full subdifferential and the function value at those points. Points in the GPH matrix representation are stored in order. Suppose we pick n points to completely represent an entity. We represent the GPH matrix as

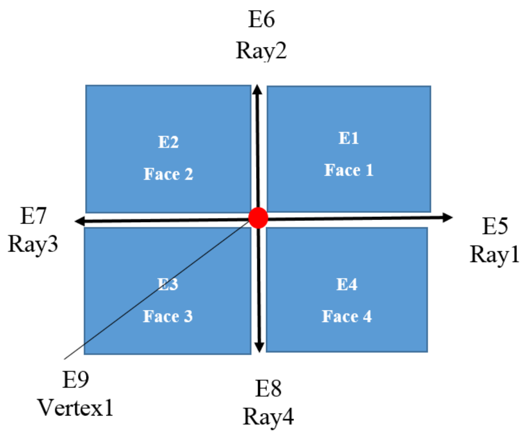


Figure 3.1: The domain of the l_1 norm function

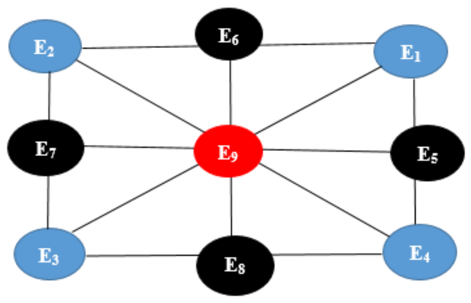


Figure 3.2: Entity graph for the l_1 norm function

$$\text{GPH matrix} = \begin{bmatrix} x_1 & x_2 \dots & x_n \\ s_1 & s_2 \dots & s_n \\ y_1 & y_2 \dots & y_n \\ b_1 & b_2 \dots & b_n \\ b_1^* & b_2^* \dots & b_n^* \end{bmatrix}$$

where x_i is the coordinate of a point, s_i is a subgradient of f at the point x_i and y_i is the value of f at x_i. The points x_i represents a polyhedral set in the plane and are given in clockwise order. Note that we need to compute the full subdifferential of f at x_i. The binary flag b_i is used to identify whether x_i is an extreme point. We classify a point using the flag from Table 3.1.

Table 3.1: Flag for x and s.

Flag	Type
0	extreme point
1	non-extreme point

Similarly, using another binary flag b_i^* we indicate whether s_i is an extreme point. Table 3.1 contains the flags used for b_i^*.

In $\mathbb{R}^d$, the dimension of the GPH matrix is $(2d+3)n$, where n is the number of points. For example, in $\mathbb{R}^2$, the dimension of a GPH matrix is $7n$.

Example 3.2. Consider the following example

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

whose conjugate is

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

The domain of the PLQ function f is shown in Figure 3.3(a). Suppose we want to compute the GPH matrix for the entity $E_1 = \{(x_1, 0) : x_1 \geq 0\}$. The entity E_1 is a ray which is adjacent to $F_1 = \{x : x_1, x_2 \geq 0\}$ associated with function $f_1(x_1, x_2) = \frac{1}{2}{x_1}^2 + \frac{1}{2}{x_2}^2$ and entity $F_2 = \{x : x_1 \geq 0, x_2 \leq 0\}$

associated with function $f_2(x_1,x_2)=\infty$. The domain of $f_2(x_1,x_2)$ is empty and the subdifferential is $\partial f_2(x_1,x_2)=\emptyset$.

The point $x=(0,0)$ is an extreme point of E_1 and we represent it using the flag $b=0$. The subdifferential of the function $\tilde{f}_1$ defined on $\mathbb{R}_+\times\mathbb{R}_+$ and equal to f_1 is

$$\partial\tilde{f}_1(x_1,x_2)=\{0\}\times(-\infty,0].$$

The point $(0,0)$ is an extreme point of $\tilde{f}_1$ and we store it twice in the GPH matrix. For $x=(0,0)$ a subgradient of $\tilde{f}_1$ is $s=(0,0)$ which is an extreme point of $\partial\tilde{f}_1(0,0)$ and we indicate it by using $b^*=0$. Any other subgradient at $x=(0,0)$ e.g. $s=(0,-1)$ is a non-extreme point and has the flag $b^*=1$.

Consider another point $x'=(1,0)$ (or any other point) from E_1. We use the flag $b=1$ to indicate that x' is a non-extreme point in the primal. The subdifferential of the function f_1, whose domain is $\mathbb{R}_+\times\mathbb{R}_+$, at x' is $\{(1,0)\}$ which is an extreme point in the dual. The GPH matrix for E_1 is

$$\mathrm{GPH}_1=\begin{bmatrix}1&0&0\\0&0&0\\1&0&0\\0&0&-1\\1&0&\infty\\1&0&0\\0&0&1\end{bmatrix}.$$

In $\mathbb{R}^2$, we can recover the polyhedral set by computing the convex hull of x_i where $i=1,\cdots,n$ (s_i in the dual). Note that we need at least one extreme point to recover the polyhedral set and all non-extreme points are connected with their adjacent extreme points.

Fact 3.3. *(Representation of a polyhedral set [Roc70, Theorem 19.1],[BJS11, Theorem 2.1]) Let $C=\{x\in\mathbb{R}^2 : a_i^Tx\le b_i, x\ge 0, i=1,\cdots,m\}$ be a polyhedral set. Assume the set of extreme points is nonempty; then it contains a finite number of elements e.g. $x_1,x_2,\cdots,x_k$. If C is bounded then the set of extreme direction is empty. If C is not bounded then the set of extreme directions is nonempty and has a finite number of elements e.g. $d_1,d_2,\cdots,d_l$. Moreover, $\bar{x}\in C$ if and only if it is represented as a convex combination of the extreme points and the nonnegative linear combination of the extreme directions, that is,*

$$\bar{x}=\sum_{i=1}^{k}\lambda_i x_i+\sum_{i=1}^{l}\mu_i d_i,$$

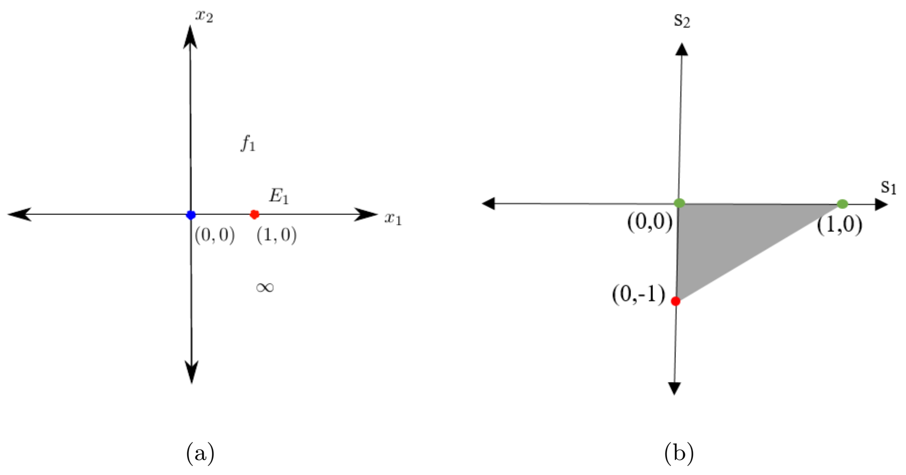


Figure 3.3: (a)The domain of a PLQ function f, we use blue dot to indicate an extreme point in the primal (b)The subdifferential of f, green dots are used to represent the extreme points in the dual.

where $\sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0, i = 1, \cdots, k, \mu_i \geq 0, i = 1, \cdots, l.$

If we store enough points then it is possible to represent any polyhedral set using a GPH matrix. We store the polyhedral set having different number of inequalities as follows,

(i) No inequality: If no inequality is found then the domain is the full space. To represent the full space we create an artificial polyhedral decomposition. Note that the polyhedral decomposition is not unique. However the decomposition should be a polyhedral subdivision. Figure 3.4 shows one possible representation of the full space.

(ii) Polyhedral sets with one inequality: If the inequality represents a half space then we use the same representation as the full space i.e. we make an artificial polyhedral decomposition and then store the extreme and the non-extreme points. If the inequality represents a ray, which already contains an extreme point, we pick a non-extreme point to represent it. We divide a line into two rays and store it with one extreme and two non-extreme points. Examples are presented in Figure 3.5.

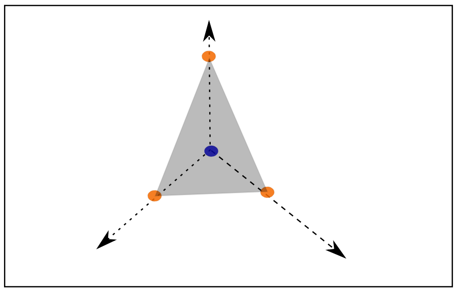


Figure 3.4: Example of no inequality. We use dotted lines to indicate the artificial polyhedral decomposition. We represent extreme points by using blue and non-extreme points by using orange color.

(iii) Polyhedral sets with two or more inequalities: Consider the following two cases,

  (a) If the hyperplanes associated with inequalities intersect, we pick the extreme points of the intersection. If the extreme points are not enough to represent the polyhedral set then we add some non-extreme points.

  (b) If the hyperplanes do not intersect e.g. if they are parallel then we represent each of them individually using extreme and non-extreme points and recover the polyhedral set by computing the convex hull.

  The representation of the polyhedral set with two or more inequalities is illustrated in Figure 3.6 and 3.7.

Consider the domain of the l_1 norm function which is illustrated in Figure 3.1. Suppose we want to compute the GPH matrix of the face E_1. The entity E_1 contains all the points with nonnegative coordinates. There is no unique representation of the GPH matrix. For efficiency we should pick minimal number of points to completely represent an entity.

Example 3.4. The entity E_1 can be represented using different GPH ma-

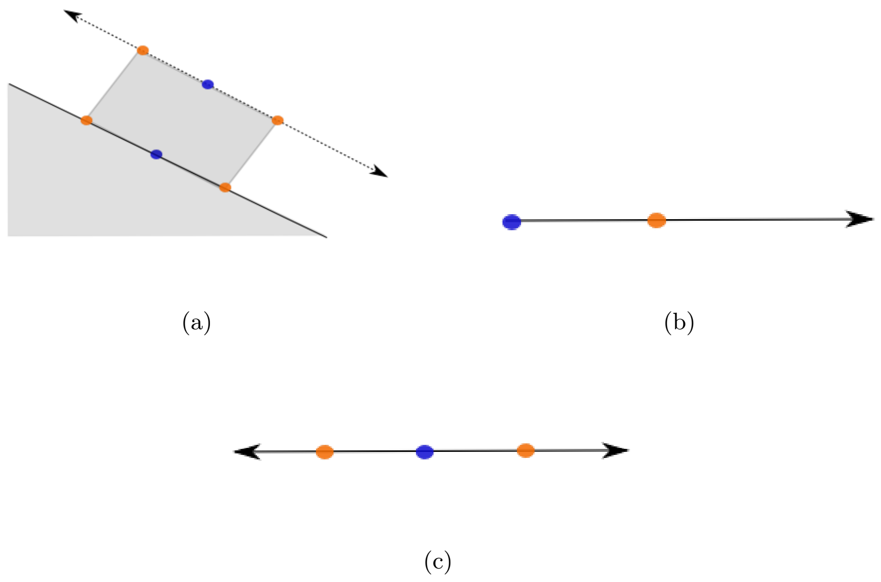


Figure 3.5: Example of polyhedral sets with one inequality. (a) A half space. To represent it we assume a ray in the half space and then select the extreme and the non-extreme points. (b) A ray which contains an extreme point. To represent it completely we pick a non-extreme point. (c) A line. We represent a line using two rays.

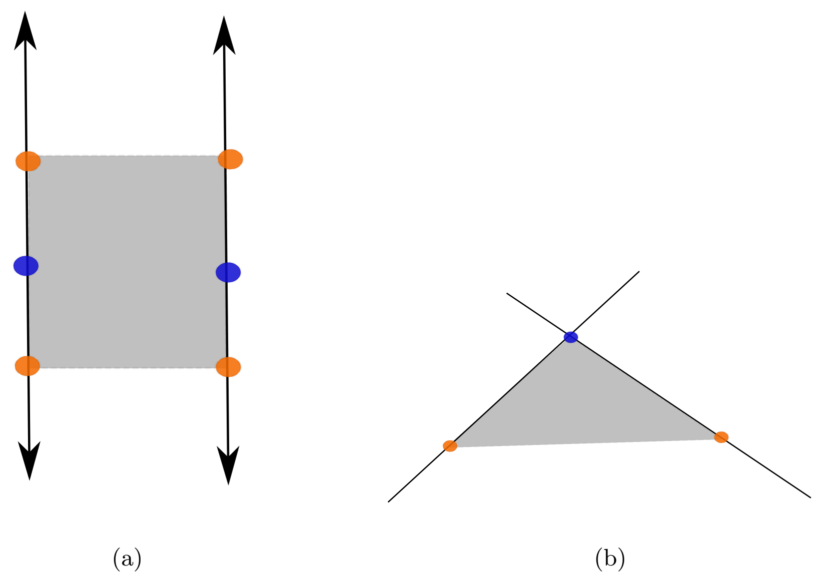


Figure 3.6: Example of the polyhedral sets with two hyperplanes. (a) The polyhedral set if the hyperplanes are parallel (b) The polyhedral set if two hyperplanes intersect.

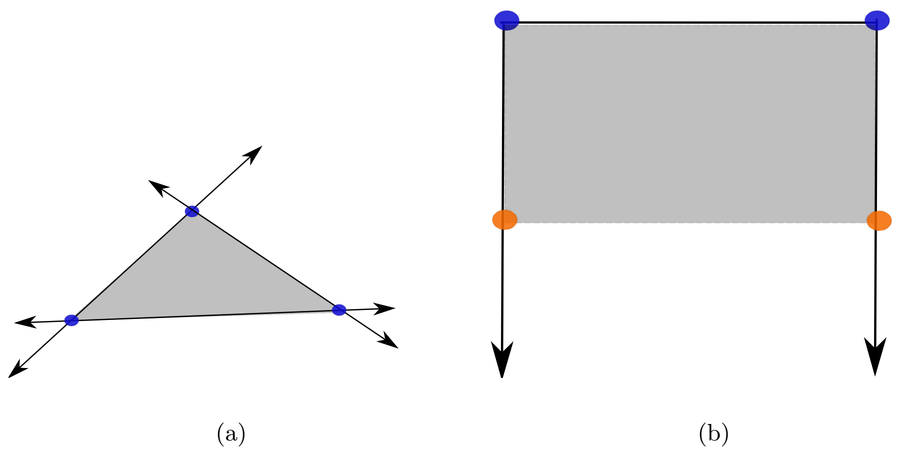


Figure 3.7: Example of polyhedral sets with three inequalities.

trices. For example either of the following would work.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 10 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 12.5 & 1 & 0 & 1 & 50 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now we show how to recover the function from the GPH matrix. The procedure is general so we explain it for the PLQ function $f : \mathbb{R}^d \to \mathbb{R}^d \cup \{+\infty\}$. Recall that, on each piece of the domain the function f is quadratic and represented by Equation (2.1). The subgradient of f at a point x is represented by Equation (2.3). For each x we get d equations from (2.3)(gradient information) and 1 equation from knowing the function value. So for each x we have a total of $d + 1$ equations.

The number of unknowns for Q_k is $d + d - 1 + \cdots + 1 = \frac{d(d+1)}{2}$ since Q_k is a symmetric matrix. The number of unknowns for q_k is d and for α_k is 1. The total number of unknowns are

$$\frac{d(d+1)}{2} + d + 1 =(d + 1)(\frac{d}{2} + 1).$$

So the minimum number of points required to determine the function f_k is $(\lceil d/2 \rceil + 1)$.

In $\mathbb{R}^2$, if we want to recover a function from a GPH matrix, we need to compute the value of 6 unknowns. For each point we get 3 equations. So we need to store at least two points to represent an entity. However, if two points are not enough to completely represent the polyhedral set then we need to store more points.

For every x we denote the function as follows

$$f_k(x) =y = \frac{1}{2} \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \alpha. \tag{3.1}$$

So the gradient can be computed as

$$s_1 = ax_1 + bx_2 + q_1, \tag{3.2}$$

$$s_2 = bx_1 + cx_2 + q_2. \tag{3.3}$$

Consider the entity E_1 of Example 3.1, which is a face. The GPH matrix for E_1 is,

$$\mathrm{GPH}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Here we pick three points from dom f to represent E_1.

The 6th row of the GPH of E_1 is the flag for these points and the last row is the flag for the subdifferentials. We have an extreme point $(0,0)$ and two non-extreme points ($(1,0)$ and $(0,1)$). We use a 0 as a flag to indicate that the point $(0,0)$ is an extreme point. Similarly, we use 1 to indicate that the points $(1,0)$ and $(0,1)$ are non-extreme points.

To get the function that defines the entity E_1 we need to compute the value of 6 unknowns $(a, b, c, q_1, q_2, \alpha)$. From Equation (3.2) we get the following linear system,

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ q_1 \end{bmatrix}.$$

The solution of the system is

$$\begin{bmatrix} a \\ b \\ q_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Similarly using the Equation (3.3) we compute

$$\begin{bmatrix} b \\ c \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

If we substitute the value of a, b, c, q_1 and q_2 in Equation (3.1) we get $\alpha = 0$. We deduce

$$f_1(x_1, x_2) = x_1 + x_2. \tag{3.4}$$

In the GPH matrix representation, we can store a point x multiple times if the subdifferential is multi-valued i.e. if we have multiple subgradient at x [GL10, GL11]. For example, consider the entity E_5 of Example 3.1.

The entity E_5 is a ray which is adjacent to two faces: E_1 and E_4. Consider $x = (0,0)$. The face E_1 is associated with $f_1(x) = x_1 + x_2$ and the only subgradient of f_1 at $x = (0,0)$ is $(1,1)$. Similarly, the face E_4 is associated with

$$f_4(x) = x_1 - x_2,$$

and at $x = (0,0)$, the subgradient is $(1,-1)$. At any point on E_5 (except $(0,0)$) the subdifferential is

$$\begin{aligned}\partial f(x_1,x_2) =&\{1\} \times [1,-1],\\ &= \operatorname{co}\{(1,1),(1,-1)\}.\end{aligned}$$

and has two extreme points $(1,1)$ and $(1,-1)$. We represents the GPH matrix associated with E_5 as

$$\mathrm{GPH}_5 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Similarly the vertex E_9 is adjacent to four faces. The subdifferential is

$$\begin{aligned}\partial f(0,0) =&[-1,1] \times [-1,1],\\ &= \operatorname{co}\{(1,1),(1,-1),(-1,1),(-1,-1)\}.\end{aligned}$$

This set is representable using 4 points and a GPH matrix associated with E_9 is

$$\mathrm{GPH}_9 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Note that the subdifferential of every x inside of a face is single valued. However, the subdifferential of a vertex or at any point of an edge (line/ray/segment) is usually multi-valued.

In our implementation, we use two hypermatrices H_p and H_d to store the GPH matrices of all entities in the domain of f and the domain of

f^* respectively. A hypermatrix is a generalization of a matrix to a multi-dimensional array. In $\mathbb{R}^d$, the dimension of the hypermatrix is $N \times M \times n_{max}$ where N is the total number of entities, $M = (2d+3)$ is the number of rows of the GPH matrix and $n_{max} = \max_{i=1,\dots,N} n_i$ where n_i is the number of columns in the GPH matrix of entity i. For example, the function l_1 norm has nine entities and the maximum value of n_i is 4 which is found from the GPH matrix of the vertex E_9. So the dimension of the hypermatrix H used to store the entities of the l_1 norm is $9 \times 7 \times 4$.

Recall that with our data structure, for any point x, we can recover not just one subgradient of f at x but the full subdifferential of f at x.

3.1.2 Adjacent entities

Each node of $\mathcal{G}$ contains information of its adjacent entities. We store the adjacency information of all entities using a matrix. We call this matrix a *Neighbour Matrix* and denote it by NM. The dimension of NM is $N \times m$ where N is the number of entities and m is the maximum degree of all vertices in the entity graph.

Consider Example 3.1 which has nine entities. The vertex E_9 has eight adjacent entities which is the maximum. So we use a Neighbour Matrix of dimension 9×8. The entity E_1 is adjacent to E_5, E_6 and E_9. In the entity graph $\mathcal{G}$, the node $\mathcal{V}_1$ which contains E_1, stores the indices of the adjacent entities i.e. 5, 6, 9. We represent the Neighbour Matrix of Example 3.1 as

$$NM = \begin{bmatrix} 5 & 6 & 9 & 0 & 0 & 0 & 0 & 0 \\ 6 & 7 & 9 & 0 & 0 & 0 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\ 5 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 9 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 9 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 9 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 9 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}.$$

Each row of this matrix contains the adjacency information of the corresponding entity. For example, the index of all adjacent entities of E_5 is found in the 5th row of NM.

However, we never read the full NM matrix. We extract the non-zero elements from each row which gives the full adjacency information of an entity. As a result the time complexity is not increased.

3.1.3 Entity Type

The type of each entity is defined by using a flag from Table 3.2. We use an array, called T, of dimension $1 \times N$ to store the type of all entities.

Table 3.2: Entity type.

Entity Type	Flag
Vertex	1
Face	2
Line	3
Ray	4
Segment	5

For example, the array which stores the entity type of the l_1 norm function is

$$T = \begin{bmatrix} 2 & 2 & 2 & 2 & 4 & 4 & 4 & 4 & 1 \end{bmatrix}$$

and means that entities E_1 to E_4 are faces, entities E_5 to E_8 are rays and E_9 is a vertex.

We visualize the data structure using Figure 3.8. Consider a PLQ function with N entities and each entity has a GPH matrix, adjacency information and entity type. We store all GPH matrices in a hypermatrix, all adjacency information in neighbour matrix NM and all entity types in an array T.

3.2 Algorithm

Consider the conjugate computation problem as a graph traversal problem. In Algorithm 1, we use breadth-first search to traverse the entity graph $\mathcal{G}$.

Consider $f : \mathbb{R}^2 \to \mathbb{R} \cup \{+\infty\}$ where dom f has N entities. In this algorithm, when we traverse an entity i, we store the index of all adjacent entities in an array denoted D. The dimension of D is $1 \times N$. We traverse $\mathcal{G}$ according to the index stored in D. Note that we will not store any index in D which is already stored in D. To check that the index of an entity is already stored in D or not we use a binary array I of dimension $1 \times N$. In the binary array, if the value of an element is 1 then the index of the corresponding entity is already stored in D and if the value is 0 then it is not stored yet.

Algorithm 1: Compute_PLQ_Conjugate

```
input  : $H_p$ (a hypermatrix which contains the GPH matrices of all
         entities), $NM_p$ (contains the adjacency information of all
         entities), $T_p$ (contains the type of all entities).
output: $H_d$ (a hypermatrix which contains the GPH matrices of all
         dual entities), $NM_d$ (contains the adjacency information of
         all dual entities), $T_d$ (contains the type of all dual entities).

Initialize $I$ with zero and set $I(1) = 1$ ;
Initialize $D$ with zero and set $D(1) = 1$ and $\bar{N} = 1$;
for $i \leftarrow 1$ to $N$ do
    $j \leftarrow D$(i) ;
    $G_p \leftarrow H_p(j,:,:)$ ;
    $[G_d, t] \leftarrow$ Conjugate_GPH($G_p$) ;
    $H_d(j,:,:) \leftarrow G_d$ ;
    $T_d(j) \leftarrow$ t ;
    if ($\bar{N} < N$) then
        $E_{adj} = NM_p(j,:)$ ;
        Compute_Index($E_{adj}, D, I, \bar{N}$) ;
    end
end
$NM_d \leftarrow NM_p$ ;
```

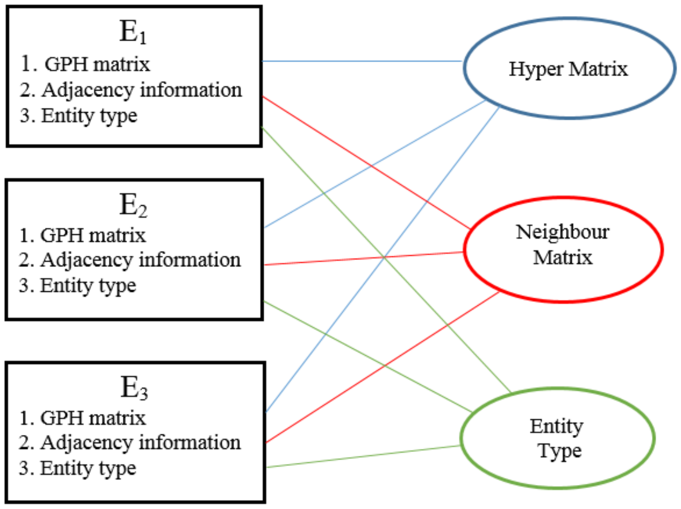


Figure 3.8: Visualization of the data structure

Suppose we want to traverse the entity graph $\mathcal{G}$ of Figure 3.2 using Algorithm 1. We start from E_1 which is a face. We have the following informations about E_1,

$$G_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\text{Indices of adjacent entities} = \begin{bmatrix} 5 & 6 & 9 \end{bmatrix},$$
$$\text{Entity type} = 2.$$

We start to traverse the entity graph $\mathcal{G}$ from the entity E_1 and initialize D as

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

the index array I as

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and $\bar{N} = 1$. We apply Algorithm 2 to compute the conjugate of E_1. The matrix G_d is the GPH matrix which contains the conjugate of E_1, we have

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

The next step is to identify the type of entity of G_d, i.e. if it is a vertex, an edge, or a face. We compute the total number of unique subdifferentials of G_d, which is 1 and deduce that the conjugate of the face E_1 is a vertex.

Next we check the adjacent entities of E_1 using Algorithm 3. We find that the entity E_1 is adjacent to the entities $E_5,\ E_6,\ E_9$. Then we check the corresponding index of I to see whether these entities are already included in D or not. In the index array I, the 5^{th}, 6^{th} and 9^{th} elements are zero i.e. these elements are not included in D. So we update D, I and $\bar{N}$ as

$$\begin{aligned} I =& \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \\ D =& \begin{bmatrix} 1 & 5 & 6 & 9 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \\ \bar{N} =& 4. \end{aligned}$$

We check the index of the next entity from D which is 5. Now we move to E_5 and do the same computation. Some steps of traversing the entity graph($\mathcal{G}$) of the l_1 norm function are illustrated in Figure 3.9.

We traverse all nodes of $\mathcal{G}$ and update the array D and I accordingly. The changes of D and I after each iteration of Algorithm 1 are presented in Table 3.3.

In Table 3.3, we use red color for a currently visiting vertex and blue color for already visited vertex.

Figure 3.10 shows the partition of the primal and the dual domain of the l_1 norm function. The mapping of the entities from the primal to the dual domain is presented in Table 3.4. According to Table 3.4, the mapping is a one-to-one entity mapping.

3.3 Complexity

The space and time complexity of our algorithm is computed next.

Algorithm 2: Compute_GPH

```
input  : $G_p$ (the GPH matrix in the primal).
output: $G_d$ (the GPH matrix in the dual which contains the
        conjugate of $G_p$), t (type of the dual entity).
//Compute $G_d$ ;
$x = G_p(1:2,:)$, $s = G_p(3:4,:)$ ;
$y = G_p(5,:)$, $b = G_p(6,:)$ ;
$b* = G_p(7,:)$ ;
The conjugate is $G_d \leftarrow \begin{bmatrix} s \\ x \\ s^T x - y \\ b \\ b^* \end{bmatrix}$;
//Compute $t$ ;
$P_u$ = number of unique vectors in $\{G_d(1,k), G_d C(2,k) : k\}$;
$P_t$ = number of columns of $G_d$ ;
if $P_u$ = 1 and $P_t \geq 1$ then
    // $G_d$ is a vertex;
    t = 1 ;
else if $P_u$ = 2 then
    //$G_d$ is an edge;
    $P_{b_0}$ = number of elements in $\{G_d(6,k) : G_d(6,k) = 0\}$ ;
    $P_{b_1}$ = number of elements in $\{G_d(6,k) : G_d(6,k) = 1\}$ ;
    if $P_{b_1}$ = 2 then
        // $G_d$ is a line;
        t = 3 ;
    else if ( $P_{b_0}$ = 1 and $P_{b_1}$ = 1 ) then
        // $G_d$ is a ray;
        t = 4 ;
    else if $P_{b_0}$ = 2 then
        // $G_d$ is a segment;
        t = 5 ;
    end
else if $P_u \geq 2$ then
    // $G_d$ is a face;
    t = 2 ;
end
```

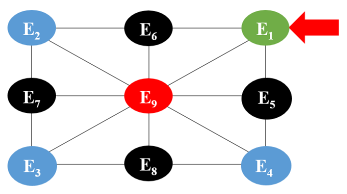
Algorithm 3: Compute_Index

```
input  : $E_{adj}$ (contains the indices of all adjacent entities of an
         entity), $D$ (contains the indices of the entities to traverse), $I$
         (indicate that which entities are already included in $D$), $\bar{N}$
         (number of nonzero elements in $D$).
output: $D$, $I$, $\bar{N}$.

$Adj \leftarrow$ [extract the non zero elements from $E_{adj}$] ;
for $i \leftarrow 1$ to $size\_of(Adj)$ do
    $index \leftarrow Adj(i)$ ;
    if $(I(index) == 0)$ then
        $I(index) = 1$ ;
        $\bar{N} \leftarrow \bar{N} + 1$;
        $D(\bar{N}) = index$ ;
    end
end
```

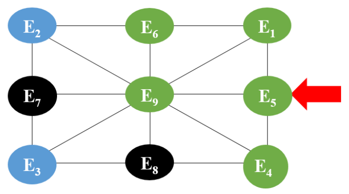
Table 3.3: Iterations of Algorithm 1.

Iterations	*D*									I								
Initialization	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	5	6	9	0	0	0	0	0	1	0	0	0	1	1	0	0	1
2	1	5	6	9	4	0	0	0	0	1	0	0	1	1	1	0	0	1
3	1	5	6	9	4	2	0	0	0	1	1	0	1	1	1	0	0	1
4	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1
5	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1
6	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1
7	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1
8	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1
9	1	5	6	9	4	2	3	7	8	1	1	1	1	1	1	1	1	1



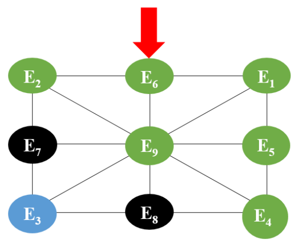
D	I
1	1
5	0
6	0
9	0
0	1
0	1
0	0
0	0
0	1

(a) Traversing E_1.



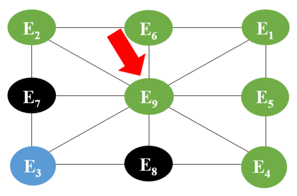
D	I
1	1
5	0
6	0
9	1
4	1
0	1
0	0
0	0
0	1

(b) Traversing E_5.



D	I
1	1
5	1
6	0
9	1
4	1
2	1
0	0
0	0
0	1

(c) Traversing E_6.



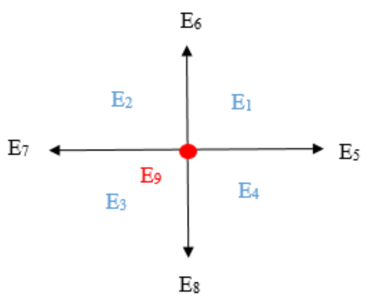
D	I
1	1
5	1
6	1
9	1
4	1
2	1
3	1
7	1
8	1

(d) Traversing E_9.

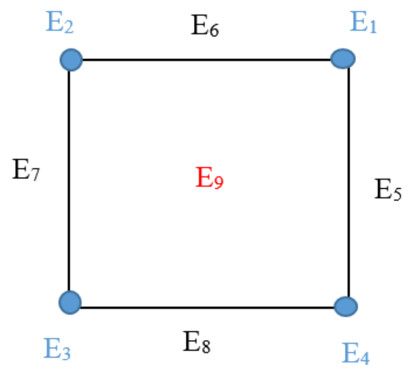
Figure 3.9: Traversing the entity graph of the l_1 norm function using Algorithm 1.

Table 3.4: Mapping of the primal entity to the dual entity for the l_1 norm function.

Primal entity	Type	Dual entity	Type
E_1	face 1	$E_1{}'$	vertex 1
E_2	face 2	$E_2{}'$	vertex 2
E_3	face 3	$E_3{}'$	vertex 3
E_4	face 4	$E_4{}'$	vertex 4
E_5	ray 1	$E_5{}'$	segment 1
E_6	ray 2	$E_6{}'$	segment 2
E_7	ray 3	$E_7{}'$	segment 3
E_8	ray 4	$E_8{}'$	segment 4
E_9	vertex 1	$E_9{}'$	face 1



(a) Partition of dom f.



(b) Partition of dom f^*.

Figure 3.10: The partition of domain of the l_1 norm function and its conjugate.

3.3.1 Space complexity

Proposition 3.5. *Consider a proper convex l.s.c bivariate PLQ function $f : \mathbb{R}^2 \to \mathbb{R}^2 \cup \{+\infty\}$ with N entities. The worst-case space complexity for computing the conjugate of f using Algorithm 1 is $\mathcal{O}(N^2)$.*

Proof. In Algorithm 1, we use two hypermatrices (H_p and H_d) to store the input and output GPH matrices, a Neighbour matrix NM to store the adjacency information, two arrays (T_p and T_d) to store the type of entities, an array D to store the indices to traverse and a binary array I.

The space complexity of our algorithm mainly depends on the size of the hypermatrix and the Neighbour matrix. Recall that the size of the hypermatrix H is $N \times M \times n_{max}$ where N is the total number of entities, $M = (2d + 3)$ is the number of rows of the GPH matrix and n_{max} is the maximum number of columns in the GPH matrix. If we assume d is constant, the worst-case space complexity of H is $\mathcal{O}(N)$.

In addition, we use four arrays: T_p, T_d, D and I. In the worst-case, the space required to store each array is $\mathcal{O}(N)$.

In Algorithm 1, we use a planar graph as the entity graph. Recall that the sum of the degrees of the vertices equals twice the number of edges and the dimension of the Neighbour matrix NM is $N \times m$ where m is the maximum degree of all vertices. For N entities, in the worst-case the maximum degree of the vertices is $(N - 1)$. So the worst-case space complexity of NM is $\mathcal{O}(N^2)$.

$\square$

Remark 3.6. A sparse matrix data structure would lower that space complexity to linear as would an adjacency list data structure i.e. the quadratic space complexity of our implementation is a Scilab limitation.

In addition, the quadratic space complexity does not arise when the maximum degree of vertices is bounded. For example, the worst-case space complexity of a grid domain in $\mathbb{R}^2$ is linear where the maximum degree of the vertices is 8 and the dimension of NM is $N \times 8$. The following result shows that even for nongrid domains, the matrix NM is sparse with at most $O(N)$ nonzero entries.

Corollary 3.7. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected planar graph with n_e edges and n_v vertices. Assume $n_v \geq 3$. Then the total number of edges is $n_e \leq 3n_v - 6$.*

Proof. In a planar graph $\mathcal{G}$, the edges divide the plane into different regions and each region is called a face of the graph $\mathcal{G}$. The total number of edges bordering a face F_i is called the degree of F_i.

Note that, in any planar graph $\mathcal{G}$, since an edge separates 2 faces, the sum of the degree of all faces is equal to twice the number of edges i.e. $\sum_{i=1}^{m} \deg F_i = 2n_e$.

Any face has degree $\deg F_i \geq 3$. So $2n_e = \sum_{i=1}^{m} \deg F_i \geq 3n_f$, hence $n_f \leq \frac{2}{3}n_e$. According to Euler's formula from Fact 2.29, $n_v - n_e + n_f = 2$. So $n_f = 2 + n_e - n_v \leq \frac{2}{3}n_e$, which gives $\frac{1}{3}n_e \leq n_v - 2$, consequently $n_e \leq 3n_v - 6$. □

Unfortunately, we have to use the Neighbour matrix NM to store the adjacency information due to the limitation of suitable data type in Scilab. Linear space complexity can be achieved if this algorithm is implemented in another language like Java or C. Then we can represent each node of the entity graph by using a structure containing the GPH matrix, the adjacent list and the entity type. Although Scilab supports the use of structure, its structure implementation is list-based while other languages use a more efficient array-based implementation.

3.3.2 Time complexity

Proposition 3.8. *The worst-case time complexity for computing the conjugate of a proper convex l.s.c bivariate PLQ function f with N entities using Algorithm 1 is $\mathcal{O}(N)$.*

Proof. The time complexity of our algorithm comes from the following parts

(i) Extracting N GPH matrices from the hypermatrix H_p according to the index stored in D and store it to E,

(ii) Computing the conjugate of E,

(iii) Checking the adjacency information of E and updating the arrays D and I,

(iv) Storing the GPH matrix, containing the conjugate of E, in H_d in the index which is found from the array D.

Task (i) amounts to accessing an index of a hypermatrix H_p which is a multi dimensional array. The advantage of using an array over other data types is that the time for accessing any element of an array is always constant. This is the main reason for using a hypermatrix to store the GPH matrices. So the time complexity for accessing any element of H_p, where the dimension of

H_p is $N \times M \times n_i$, is $\mathcal{O}(1)$. So the total time required to access N elements of H_p is $\mathcal{O}(N)$.

Task (ii) involves computing the conjugate. The time complexity for computing the conjugate of N entities is linear i.e. $\mathcal{O}(N)$.

In addition, task (iii) involves storing the index of adjacent entities in the array D. Recall that the indices stored in D are used to traverse the entity graph. In this part, when storing an index in D we need to check whether the index is already stored in D or not. In the worse-case, searching the index of an entity in an array takes $\mathcal{O}(N)$ time. For N entities, the time for storing all indices in D is $\mathcal{O}(N^2)$. To improve from this quadratic time complexity, we use a binary array I. Every time we want to insert the index of an entity in D we check the corresponding position of I. If we find 0, that means the index is not stored in D and we can insert it in D. If we find 1, it indicates this index is already stored in D. Accessing a position of I takes constant time. So the time complexity for N entities reduces to linear i.e. $\mathcal{O}(N)$.

Finally, task (iv) requires accessing a position of a hypermatrix H_d and storing the output GPH matrix in H_d. Similar to the first part, this also takes linear time i.e. the time complexity is $\mathcal{O}(N)$.

In total, the time complexity is

$$\mathcal{O}(N + N + N + N) = \mathcal{O}(N).$$

Therefore, the overall time complexity for our algorithm is linear. $\square$

We proved that it is possible to compute the conjugate of a bivariate PLQ function in linear time while using the same input and output data structure.

Chapter 4

Numerical experiments

In this chapter, we will present some examples and the result of a numerical experiment. Our algorithm is implemented in Scilab [SCI].

4.1 Example 1

The PLQ (the l_1 norm) function explained in Chapter 4 was an example of a polyhedral subdivision. Now we explain an example of a PLQ function that is not a polyhedral subdivision.

Example 4.1. Consider the PLQ function f which is defined as follows,

$$f(x_1, x_2) = \begin{cases} \frac{1}{2}(x_1^2 + x_2^2), \text{ if } x_1 \geq 0, x_2 \geq 0; \\ -2x_1 + \frac{1}{2}x_2^2, \text{ if } x_1 \leq 0, x_2 \geq 0; \\ \infty, \text{ otherwise.} \end{cases} \tag{4.1}$$

The domain of f has three pieces: a strictly convex piece, a convex piece and a piece with infinite function value. The dom f is decomposed into six entities: two finite faces, three rays and one vertex. This is presented in Figure 4.2.

The polyhedral set for the PLQ function f is $\mathcal{C} = \{C_1, C_2, C_3\}$ which is not a polyhedral subdivision because $C_1 \cap C_3$ or $C_2 \cap C_3$ is neither empty nor a common proper face of both. We need to modify the domain to make it a polyhedral subdivision.

Note that there is no unique way to make a polyhedral subdivision. We divide C_3 into two pieces where on each piece the function value is infinite. Now the domain of f becomes a polyhedral subdivision as $C_1 \cap C_3$ and $C_2 \cap C_4$ is a common proper face of both. The modified domain of f is presented in Figure 4.3.

The input for Algorithm 1 consists of a hypermatrix H_p, a Neighbour matrix NM and an array T_p. The hypermatrix H_p has the following six

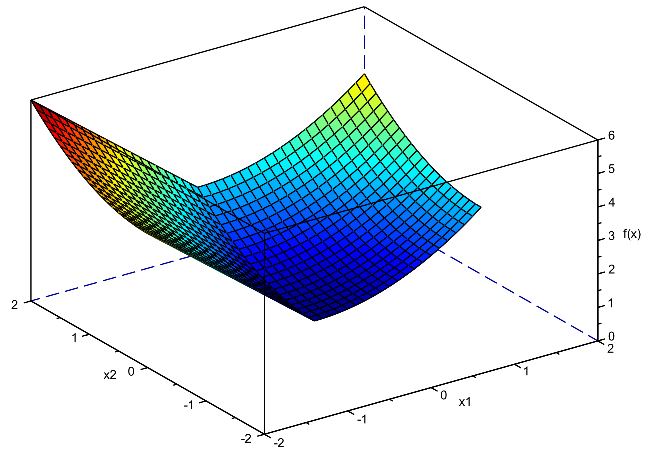
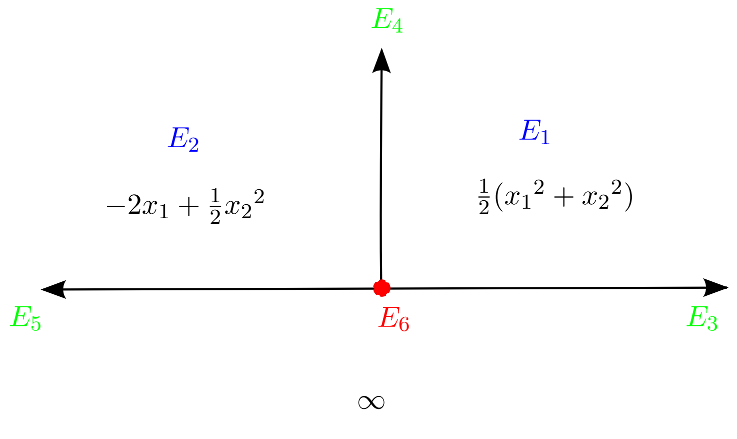
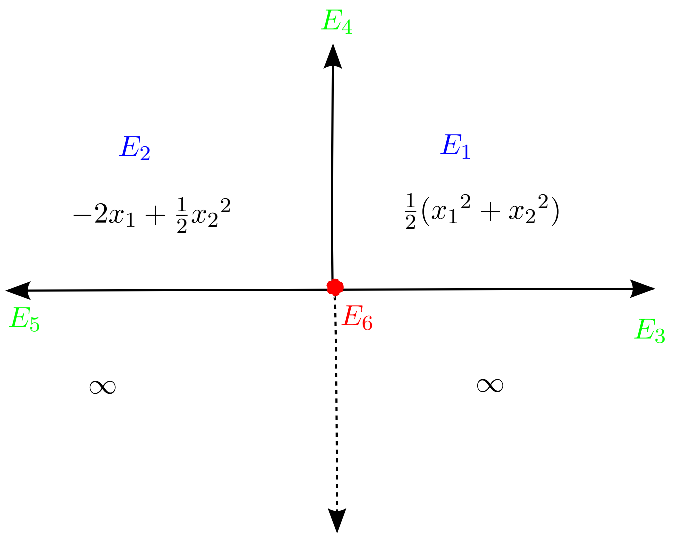


Figure 4.1: The function $f(x_1, x_2)$ from Equation 4.1



E_4
E_2
$-2x_1 + \frac{1}{2}{x_2}^2$
E_1
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
E_5
E_6
E_3
∞

Figure 4.2: Partition of dom f which is not a polyhedral subdivision.



E_4
E_2
E_1
$-2x_1 + \frac{1}{2}{x_2}^2$
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
E_5
E_6
E_3
∞
∞

Figure 4.3: Partition of dom f which is a polyhedral subdivision.

GPH matrices,

$$H_p(1,:,:) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_p(2,:,:) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ -2 & -2 & -2 \\ 0 & 0 & 1 \\ 2 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

The entity E_3 is a ray which is adjacent to two faces: F_1 and F_3. We pick two points to build a GPH matrix for E_3. The function associated with F_1 is

$$f_1(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2),$$

and the function associated with F_3 is

$$f_3(x_1, x_2) =\infty.$$

Since $f_3 = \infty$, dom $f_3 = \emptyset$ and the subdifferential is $\partial f_3 = \emptyset$.

Assume we pick a point $(0, 0)$ which is an extreme point of E_3. The subdifferential of the function $\tilde{f_1}$ defined on $\mathbb{R}_+ \times \mathbb{R}_+$ and equal to f_1 is

$$\partial \tilde{f_1}(0, 0) =\{0\} \times (-\infty, 0].$$

In the GPH matrix we store the point $x = (0, 0)$ twice because it is the extreme point of the function $\tilde{f_1}$. For $s = (0, 0)$, we set $b^* = 0$ as it is the extreme point of $\partial\tilde{f_1}(0, 0)$. When we choose any other subgradient e.g. $s = (0, -1)$ then we set $b^* = 1$ as it is a non-extreme point.

Similarly, if we pick another point $(1, 0)$, the subdifferential of f_1 at $(1, 0)$ is $\{(1, 0)\}$ which is an extreme point. The GPH matrix of E_3 is

$$H_p(3, :, :) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \\ \frac{1}{2} & 0 & \infty \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

A similar computation is required to compute the subdifferential of E_5 and E_6 because in both entities we need to represent the infinity in the domain.

$$H_p(4, :, :) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & -2 & -2 \\ 1 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_p(5, :, :) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ -2 & -2 & -2 \\ 0 & 0 & -2 \\ 2 & 0 & \infty \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$H_p(6,:,:) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -2 & -2 & 0 \\ 0 & 0 & -2 & -1 \\ 0 & 0 & \infty & \infty \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The maximum degree of entities is 5. So the Neighbour matrix NM of f is

$$NM = \begin{bmatrix} 3 & 4 & 6 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 1 & 6 & 0 & 0 & 0 \\ 1 & 2 & 6 & 0 & 0 \\ 2 & 6 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}.$$

The type of all input entities are stored in T_p as

$$T_p = \begin{bmatrix} 2 & 2 & 4 & 4 & 4 & 1 \end{bmatrix}$$

which indicates that the domain of f has two faces, three rays and a vertex.

We apply Algorithm 1 and compute the conjugate of f

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

The conjugate is illustrated in Figure 4.4. The domain of f^* has twelve entities: four faces, five rays, one segment and two vertices. The partition of dom f^* is illustrated in Figure 4.5.

We store all the entities making a partition of dom f^* in a hypermatrix

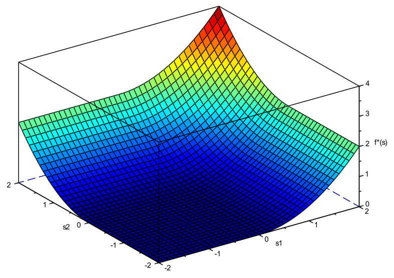


Figure 4.4: The conjugate $f^*(s_1, s_2)$.

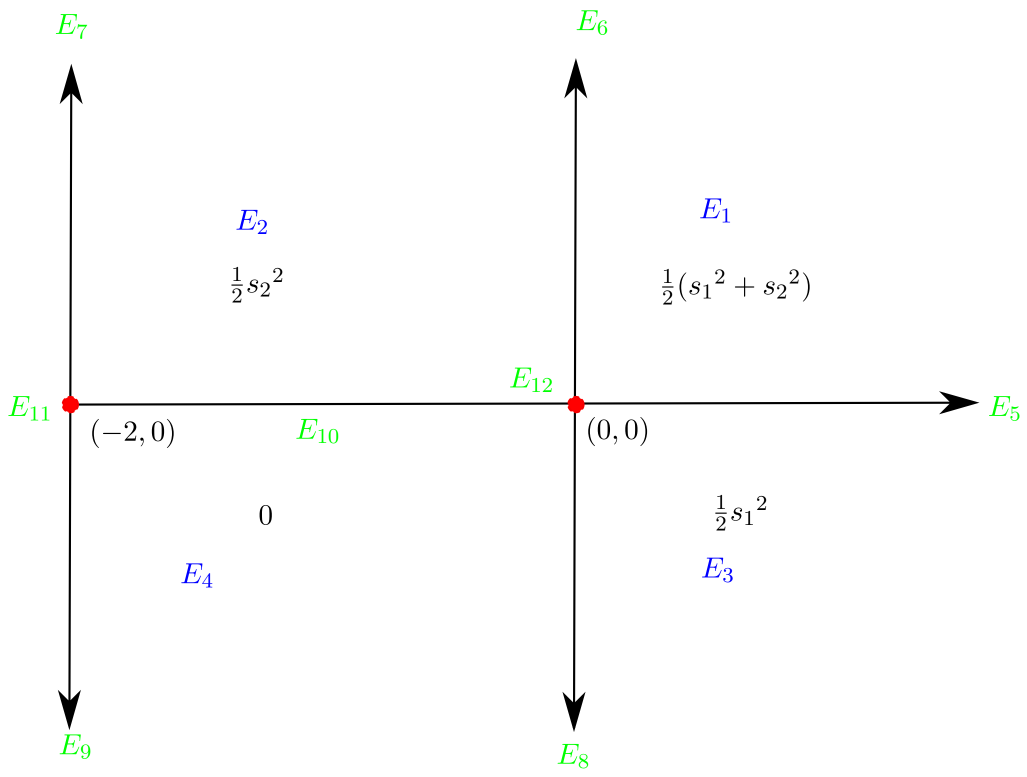


Figure 4.5: Partition of the domain of f^*.

H_d as follows,

$$H_d(1,:,:) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_d(2,:,:) = \begin{bmatrix} -2 & -2 & -2 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_d(3,:,:) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{2} & 0 & -\infty \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$H_d(4,:,:) = \begin{bmatrix} 0 & 0 & -2 & -2 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_d(5,:,:) = \begin{bmatrix} -2 & -2 & -2 \\ 0 & 0 & -2 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\infty \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$H_d(6,:,:) = \begin{bmatrix} 0 & -2 & -2 & 0 \\ 0 & 0 & -2 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The adjacency information of the entities are not changed in the dual i.e. $NM_p = NM_d$ while

$$T_d = \begin{bmatrix} 2 & 4 & 2 & 2 & 4 & 2 \end{bmatrix}.$$

Table 4.1 is the mapping of the entities from the primal to the dual domain.

Table 4.1: Mapping of the primal entity to the dual entity.

Primal entity	Type	Dual entity	Type
E_1	face 1	$E_1{}'$	face 1
E_2	face 2	$E_7{}'$	ray 3
E_3	ray 1	$E_3{}'$	face 3
E_4	ray 2	$E_2{}'$, $E_6{}'$, $E_7{}'$, $E_{10}{}'$, $E_{11}{}'$, $E_{12}{}'$	face 2, ray 2, ray 3, segment 1, vertex 1, vertex 2
E_5	ray 3	$E_9{}'$	ray 5
E_6	vertex 1	$E_4{}'$,$E_8{}'$	face 4,ray 4

According to Table 4.1, the mapping is a one-to-many entity mapping.

4.2 Example 2

Example 4.2. Consider the example of 2D energy function,

$$f(x_1, x_2) = \frac{1}{2}({x_1}^2 + {x_2}^2)$$

which is presented in Figure 4.6.

This function does not have any inequality constraint. The domain of this function is the full two dimensional space and it has only one entity. To represent the full space we have to make some artificial partitions. Note that the number of partitions is not unique. However the domain should

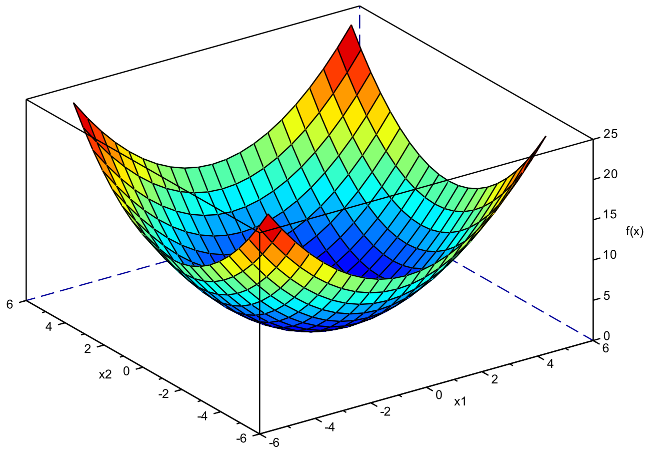


Figure 4.6: The 2D energy function.

be divided in a way that it becomes a polyhedral subdivision. Suppose we divide the domain of the energy function as follows

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

Now the domain of $f(x_1, x_2)$ has four pieces which is illustrated in Figure 4.7. This is a polyhedral subdivision and dom f has nine entities: four faces, four rays and a vertex.

The next step is to compute the GPH matrices for all entities. We build

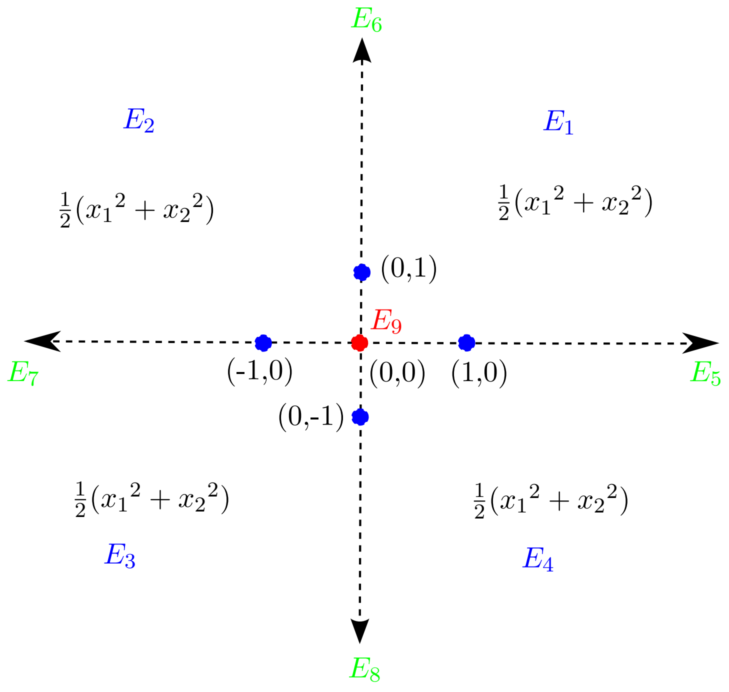

E_6
E_2
E_1
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
(0,1)
E_9
E_7
(-1,0)
(0,0)
(1,0)
E_5
(0,-1)
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
$\frac{1}{2}({x_1}^2 + {x_2}^2)$
E_3
E_4
E_8

Figure 4.7: The domain of 2D energy function. The dotted lines indicate that the partitions are not real partitions.

the GPH matrices H_p as follows,

$$H_p(1,:,:) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_p(2,:,:) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_p(3,:,:) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_p(4,:,:) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$H_p(5,:,:) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_p(6,:,:) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_p(7,:,:) = \begin{bmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_p(8,:,:) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H_p(9,:,:) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The energy function is the only self conjugate function [Roc70]. So the same GPH matrices is found in the dual i.e. $H_p = H_d$. The neighbour matrix NM and the entity type T for the energy function is the same as the l_1 norm function.

All the PLQ functions discussed above have a small number of entities. In the next section we give an example a of PLQ function with a large number of entities.

4.3 Partition of domain using a grid of points

Example 4.3. Consider the following additively separable PLQ function

$$f(x_1, x_2) = x_1^4 + x_2^4.$$

We set $f_1(x_1) = x_1^4$ and $f_2(x_2) = x_2^4$.

We use *plq_build* function from CCA package [CCA] to approximate an univariate PLQ function from f_1. The plq_build function is used for the quadratic approximation of f_1. We use a grid of points to approximate f_1 into an univariate PLQ function. From the univariate PLQ function we approximate a bivariate PLQ function.

For example, if we use the points $0, 1, 2, 3$ and 4 then we get an univariate PLQ function with ten pieces. For preserving the shape plq_build function takes one additional point within every interval and divide the domain into ten quadratic pieces. We get the following PLQ matrix from f_1,

$$\text{PLQ matrix of } f_1(x_1) = \begin{bmatrix} 0.00 & 0.00 & 0.00 & \infty \\ 0.75 & 0.67 & 0.00 & 0.00 \\ 1.00 & 6.00 & -8.00 & 3.00 \\ 1.61 & 9.06 & -14.12 & 6.06 \\ 2.00 & 21.64 & -54.55 & 38.55 \\ 2.57 & 29.16 & -84.65 & 68.65 \\ 3.00 & 49.52 & -189.09 & 202.64 \\ 3.55 & 61.21 & -259.26 & 307.89 \\ 4.00 & 89.47 & -459.70 & 663.40 \\ \infty & 0.00 & 0.00 & 256.00 \end{bmatrix}.$$

Figure 4.8 shows the PLQ function built from $f_1(x_1) = x_1^4$. We consider the pieces with finite function value and approximate the univariate PLQ function into a bivariate PLQ function. The univariate PLQ function f_1 has eight finite pieces. So the bivariate PLQ function contains $8 * 8 = 64$ pieces. Figure 4.9 illustrates the resulting bivariate PLQ function $f(x_1, x_2)$ approximating $x_1^4 + x_2^4$.

Next we compute all the entities and represents them using GPH matrices. We build the hypermatix H_p and compute the neighbour matrix NM. Now we apply our algorithm and compute the conjugate of f which is

$$f^*(s_1, s_2) = f_1^*(s_1) + f_2^*(s_2),$$

where $f_1^*(s_1) = (\frac{3}{4})^{\frac{4}{3}} {s_1}^{\frac{4}{3}}$ and $f_2^* = f_1^*$.

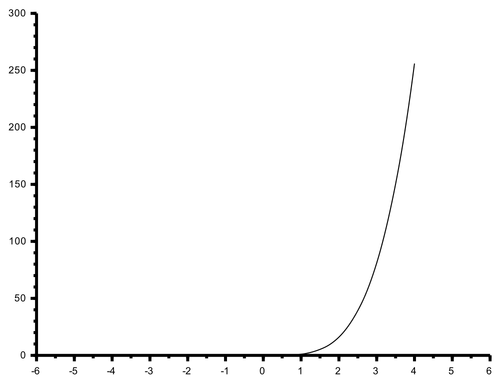


Figure 4.8: The approximation of univariate PLQ function $f_1(x_1)$.

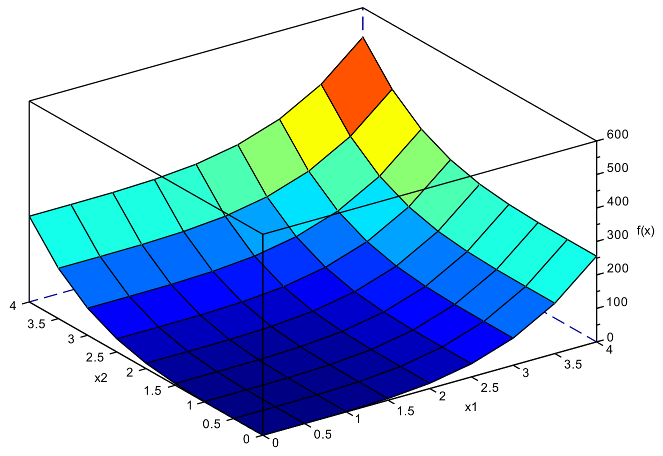


Figure 4.9: The approximation of function $f(x_1, x_2) = x_1^4 + x_2^4$.

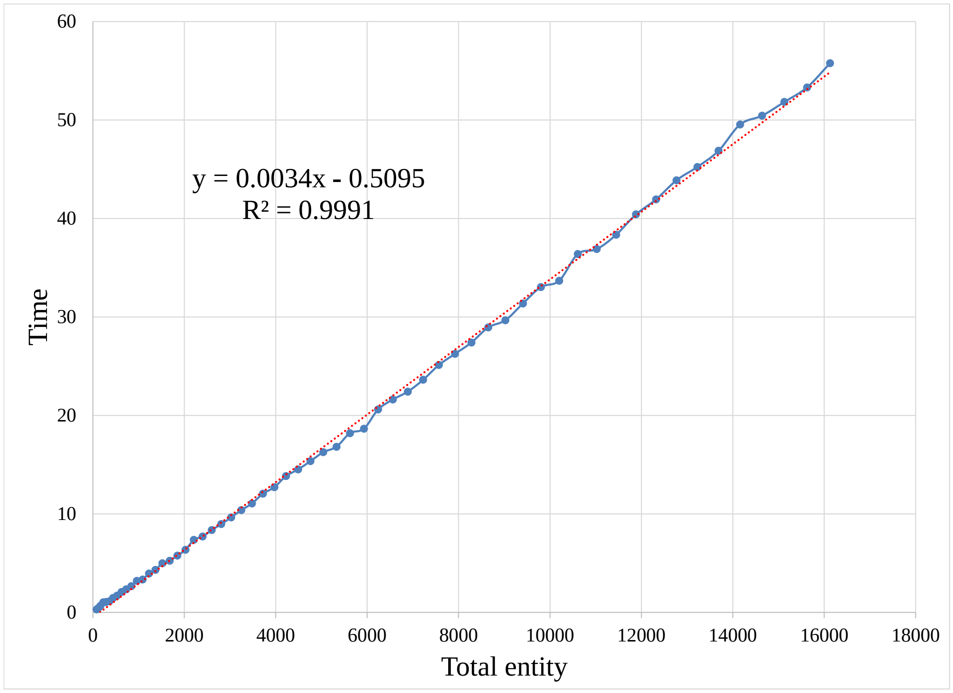


Figure 4.10: The time complexity for computing the conjugate of the function from Example 4.3 where dom f is partitioned into a grid.

We increase the number of pieces by increasing the size of the grid and measure the time for computing the conjugate of f. The time complexity of the function f is presented in Figure 4.10.

4.3.1 Performance Comparison

We compare the performance of our proposed Algorithm with the algorithm developed in [Kha13]. We compute the conjugate of the function $f(x_1, x_2)$ from Example 4.3 and measure the time for the computation using both algorithms. The times for different grid size using both algorithms are shown in Table 3.8.

Table 4.2: Comparison of computation time using the algorithm from [Kha13] and Algorithm Compute_PLQ_Conjugate. All the times are in seconds.

Total entity	[Kha13]	Proposed Algorithm
81	4.1	0.3
169	14.4	0.5
289	20.4	1.2
441	40.2	1.4
625	67.9	2.3
841	110.7	2.9
1,089	173.4	3.7
1,369	263.2	4.8
1,681	375.1	5.4
2,025	604.3	6.5
2,401	921.4	8.4
2,809	1,234.9	9.1
3,249	1,651.4	10.8

The table shows that the computation time of the proposed algorithm is more than 100 times faster than the other algorithm.

Figure 4.11 is the plot of the time complexity for these two algorithms. For the algorithm developed in [Kha13], when we fit a linear trendline the value of R^2 is 0.91. However, if we fit a quadratic trendline then we get $R^2 = 0.99$ which means this algorithm actually runs in quadratic time.

We plot the time complexity of our proposed algorithm. We fit a linear trendline and compute the R^2 value which is 0.99. The value of R^2 proves that the proposed algorithm is a linear time algorithm.

We run all numerical experiments on a Core(TM) i5 processor, 64 bit OS, 8.00 GB RAM, 2.40 GHz HP Pavilion x360 laptop, running Windows 10. The implementation of the algorithm is done using Scilab version 5.5.2. We perform the numerical experiment several times and obtained similar results each time.

The implementation of the algorithm from [Kha13] was a pure Scilab code that did not include the improvement of using the half-edge data structure provided in an external library. At the price of considerable complexity and a loss in portability, the [Kha13] algorithm can be implemented in log-linear time. However, our new algorithm would still be faster (linear-time) and much simpler.

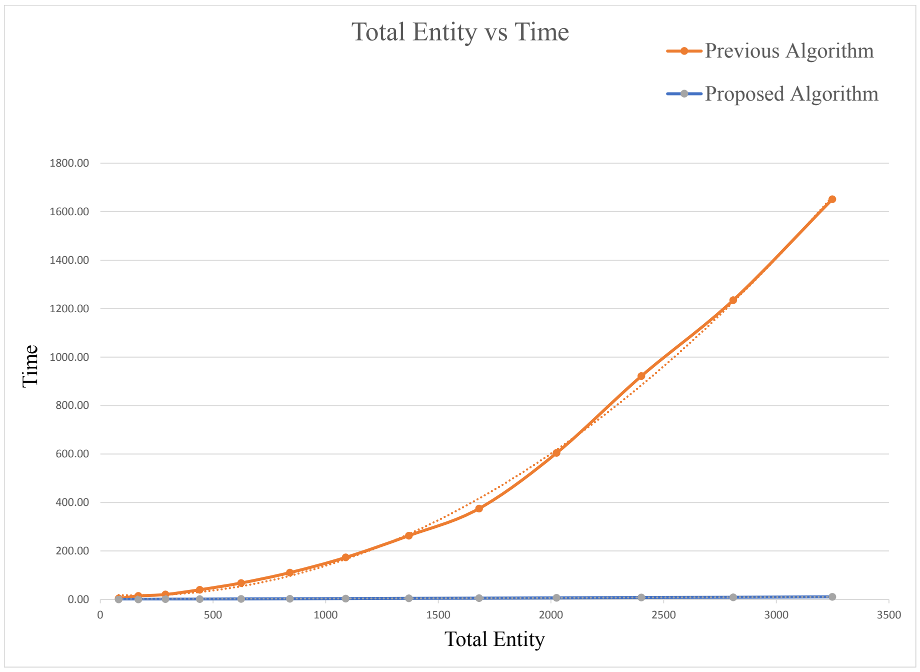


Figure 4.11: The time complexity for computing the conjugate of the function from Example 4.3 using the algorithm from [Kha13] and the proposed algorithm .

Chapter 5

Conclusion

This chapter summarizes the work we have completed so far and our future plans. Our goal was to remove the limitations and improve the worst-case complexity of the existing algorithms developed for computing the conjugate of a bivariate PLQ function. The outcome of this thesis is the first linear-time algorithm to compute the conjugate of a bivariate PLQ function.

In this thesis, we worked with a *proper convex l.s.c* PLQ function. We assume that the domain of the function is a *polyhedral subdivision*. Otherwise we need to create a polyhedral subdivision.

We used GPH matrices to store the entities of a PLQ function and stored all GPH matrices using a hypermatrix. Moreover we stored the full adjacency information of the entities. The adjacency information was not stored in the previous approaches. We traversed the entity graph, accessed each entity and computed the conjugate. We also stored the information about the visited entity. So for each entity we never need to loop through all the entities to check whether it is visited or not. As a result the overall cost for traversing the entity graph becomes linear. This is the improvement of our algorithm over the algorithms developed before (see Table 1.1).

We computed only the convex conjugate operator. The improved complexity of this algorithm encourage us to develope GPH based algorithms for computing the other convex transforms like the proximal average, the Moreau envelope, the addition and the scalar multiplication. An interesting future work will be developing new algorithms using GPH matrix-based data structure for computing all the convex transforms of a bivariate PLQ function.

We implemented our algorithm in $\mathbb{R}^2$. We provided a detail explanation about the implementation in Chapter 3. However there is the potential to implement this algorithm in higher dimension. Directions for future work include computing the conjugate of a convex PLQ function of d variables.

We presented some examples to show the performance of our algorithm for different types of bivariate PLQ functions. The results are included in Chapter 4. We showed that our algorithm can deal with a PLQ function which has thousands of pieces and still computes the conjugate in linear-time. We included a graph in Chapter 4 to show the comparison of performance of this algorithm and a previous algorithm. The graph clearly visualize that our proposed algorithm is significantly faster than the previous algorithms.

Bibliography

[Bal10] E. J. Balder. On subdifferential calculus. *Lecture notes, Universiteit Utrecht*, 2010. → pages 15

[BBM03] F. Borrelli, A. Bemporad, and M. Morari. Geometric algorithm for multiparametric linear programming. *Journal of Optimization Theory and Applications*, 118(3):515–540, 2003. → pages 6

[BJS11] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows.* John Wiley & Sons, 2011. → pages 23

[CCA] The computational convex analysis numerical library. http://atoms.scilab.org/toolboxes/CCA. → pages 55

[CGA] Computational Geometry Algorithms Library. http://www.cgal.org. → pages 5

[Fan63] K. Fan. On the Krein-Milman theorem. *Convexity*, 7:211–220, 1963. → pages 13

[FH12] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 8(19), 2012. → pages 3

[GKL14] B. Gardiner, J. Khan, and Y. Lucet. Computing the partial conjugate of convex piecewise linear-quadratic bivariate functions. *Computational Optimization and Applications*, 58(1):249–272, 2014. → pages 4, 6, 7

[GL10] B. Gardiner and Y. Lucet. Convex hull algorithms for piecewise linear-quadratic functions in computational convex analysis. *Set-Valued and Variational Analysis*, 18(3-4):467–482, 2010. → pages 29

[GL11] B. Gardiner and Y. Lucet. Graph-matrix calculus for computational convex analysis. In *Fixed-Point Algorithms for Inverse*

Problems in Science and Engineering, volume 49 of *Springer Optimization and Its Applications*, pages 243–259. Springer New York, 2011. → pages 4, 29

[GL13] B. Gardiner and Y. Lucet. Computing the conjugate of convex piecewise linear-quadratic bivariate functions. *Mathematical Programming Series B*, 139(1-2):161–184, 2013. → pages 1, 3, 5, 7, 12

[Goe08] R. Goebel. Self-dual smoothing of convex and saddle functions. *Journal of Convex Analysis*, 15(1):179–192, 2008. → pages 4, 5

[Her16] C. Hermosilla. Legendre transform and applications to finite and infinite optimization. *Set-Valued and Variational Analysis*, 24(4):685–705, 2016. → pages 2

[HUL07] J Hiriart-Urruty and Y. Lucet. Parametric computation of the legendre-fenchel conjugate with application to the computation of the moreau envelope. *Journal of Convex Analysis*, 14(3):657, 2007. → pages 2

[Kha13] J. Khan. Computational convex analysis using parametric quadratic programming, 2013. → pages vi, viii, 4, 6, 7, 13, 57, 58, 59

[LBT09] Y. Lucet, H. Bauschke, and M. Trienis. The piecewise linear-quadratic model for computational convex analysis. *Computational Optimization and Applications*, 43(1):95–118, May 2009. → pages 3

[Luc97] Y. Lucet. Faster than the Fast Legendre Transform, the Linear-time Legendre Transform. *Numerical Algorithms*, 16(2):171–185, 1997. → pages 3

[Luc06] Y. Lucet. Fast moreau envelope computation i: numerical algorithms. *Numerical Algorithms*, 43(3):235–249, 2006. → pages 2, 3

[Luc09] Y. Lucet. New sequential exact euclidean distance transform algorithms based on convex analysis. *Image and Vision Computing*, 27(1):37–44, 2009. → pages 1

[Luc10] Y. Lucet. What shape is your conjugate? A survey of computational convex analysis and its applications [reprint of mr2496900]. *SIAM Review*, 52(3):505–542, 2010. → pages 1, 2, 3, 4

[Luc13] Y. Lucet. Techniques and open questions in computational convex analysis. In *Computational and analytical mathematics*, volume 50 of *Springer Proceedings in Mathematical & Statistics (PROMS)*, pages 485–500. Springer, 2013. → pages 1, 3, 4

[MN15] B. S. Mordukhovich and N. M. Nam. Geometric approach to convex subdifferential calculus. *Optimization*, 0(0):1–35, 2015. → pages 15

[Mul88] K. Mulmuley. A fast planar partition algorithm. i. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (1988)*, volume 00, pages 580–589, Los Alamitos, CA, USA, 1988. IEEE Computer Society. → pages 6

[PS11] P. Patrinos and H. Sarimveis. Convex parametric piecewise quadratic optimization: Theory and algorithms. *Automatica*, 47(8):1770 – 1777, 2011. → pages 6

[PW16] C. Planiden and X. Wang. Strongly convex functions, moreau envelopes, and the generic nature of convex functions with strong minimizers. *SIAM Journal on Optimization*, 26(2):1341–1364, 2016. → pages 2

[Roc70] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New york, 1970. → pages 6, 12, 15, 23, 54

[RW09] R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis.* Springer-Verlag, Berlin, 2009. → pages 3, 10, 15

[SCI] Scilab. http://www.scilab.org/. → pages 1, 7, 43