Convex Hulls and Graph-Matrix Calculus: Algorithms in Computational Convex Analysis

by

Bryan Nicholas Carl Gardiner

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

BACHELOR OF SCIENCE HONOURS

in

The Irving K. Barber School of Arts and Sciences

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2010

Abstract

At the core of Convex Analysis and its applications are a collection of frequently used operators for transforming convex functions, along with the convex hull operation for convexifying functions. While numerical algorithms are usually applied to general functions with little known structure, we focus on the class of univariate piecewise linear-quadratic (PLQ) functions, for which exact algorithms have been developed.

This thesis presents two main results. In the first part, we investigate two convex hull algorithms for univariate PLQ functions. The first algorithm is an extension of the linear-time planar Beneath-Beyond algorithm, and performs a plane sweep that converts a function into its convex hull. The second uses convex duality theory to deconstruct a nonconvex function and build its convex hull using convex operators, in worst-case quadratic time. The trade-off is complexity: the second algorithm can be significantly simpler to implement.

The second part is concerned with the computation of convex transforms, such as the Legendre-Fenchel transform and the Moreau Envelope. We introduce a new family of algorithms that stores and manipulates models of the subdifferential instead of the original function.

We performed numerical experiments comparing the two convex hull algorithms, and comparing the new subdifferential algorithms to existing PLQ algorithms. These experiments confirm the time complexity for the convex hull algorithms, and show that the subdifferential algorithms have the same complexity as the PLQ algorithms, while performing an order of magnitude faster.

Table of Contents

Abstract . . . ii

Table of Contents . . . iii

Acknowledgements . . . v

1 Introduction . . . 1

2 Preliminaries . . . 2
- 2.1 Convex Analysis . . . 2
  - 2.1.1 Convexity . . . 2
  - 2.1.2 Affine and Convex Hulls . . . 3
  - 2.1.3 Set Interiors . . . 4
  - 2.1.4 Subdifferential . . . 4
  - 2.1.5 Convex Conjugate . . . 4
  - 2.1.6 Epi-addition and Epi-multiplication . . . 5
  - 2.1.7 Moreau Envelope . . . 5
  - 2.1.8 Proximal Average . . . 6
  - 2.1.9 Self-Dual Smoothing Operators . . . 7
- 2.2 Piecewise Linear-Quadratic Functions . . . 7

3 Convex Hull Algorithms . . . 9
- 3.1 Direct Approach . . . 9
- 3.2 Convex Duality Approach . . . 18
- 3.3 Performance Comparison . . . 19

4 Graph-Matrix Calculus . . . 22
- 4.1 Representation of PLQ Functions . . . 22
- 4.2 Goebel's Graph-Matrix Calculus . . . 25
- 4.3 Transforms of PLQ Functions . . . 28
- 4.4 Comparison of Algorithms . . . 31

5 Conclusion 35

Bibliography 37

Acknowledgements

I wish to thank Dr. Yves Lucet most sincerely for providing me with the opportunity to do an Honours thesis under him, and the opportunity to pursue research in the area of Convex Analysis. His patient and caring mentorship has been essential, and he has given me much inspiration and confidence.

Many other people have provided support, encouragement, and knowledge throughout my thesis work. I would like to acknowledge Dr. Heinz Bauschke and Dr. Warren Hare for their teaching, and for their everlasting enthusiasm for Mathematics.

I would also like to thank Mike Trienis for his initial implementation of the direct PLQ convex hull algorithm.

Chapter 1

Introduction

Convex Analysis is a rich field with applications in diverse areas. For example, it provides algorithms to compute distance transforms [8] and reverse degradation [5] in image and signal processing, and to improve models in machine learning [4, 23].

There are numerous operators in Convex Analysis that transform one convex function into another. Common examples of these would be the ubiquitous Legendre-Fenchel transform, the Moreau-Yoshida approximation, and the recently-introduced Proximal Average [3]. It is natural to want fast algorithms to perform these operations. Since generic numerical algorithms do not generate an exact model of the transformed function, research has focused on special classes of functions [9, 16, 17]. Restricting oneself to a well-behaved class of functions can allow exact algorithms to be developed. Piecewise linear-quadratic (PLQ) functions are such a class. They are closed under many convex transforms, including those listed above.

Existing PLQ algorithms directly manipulate a model of a function [2]. Because the subdifferential of a PLQ function is a piecewise linear multivalued mapping, the subdifferentials of many transforms are related linearly to the original subdifferentials. Goebel [12] published the exact relationship for a number of operators. We extend his results to additional operators, and provide a method to recover the function values.

First, we will look at algorithms for computing the convex hulls of PLQ functions. We will recall the extension by Trienis [22] of the Beneath-Beyond algorithm for points in a plane, to PLQ functions with unbounded domains. We will then introduce a new algorithm for computing the convex hull of a PLQ function, that uses only the convex conjugate and the pointwise maximum operators.

Chapter 2 sets notations and recalls the necessary theory. Chapter 3 discusses the two convex hull algorithms for PLQ functions. After bringing ourselves into the domain of convex functions, in Chapter 4 we apply Goebel's calculus rules to the computation of a number of convex operators, and compare these techniques to existing PLQ algorithms. Chapter 5 concludes the thesis with a results summary, and directions for future research.

Chapter 2

Preliminaries

In this chapter, we will outline notations and recall results we will use in the rest of the thesis.

We will denote the standard inner product on $\mathbb{R}^d$ with

$$\langle x, y \rangle = x^T y = \sum_{i=1}^{d} x_i y_i,$$

and the quadratic energy function

$$\mathfrak{q} : \mathbb{R}^d \to \mathbb{R} : x \mapsto \frac{1}{2}\|x\|^2.$$

We will write the open ball centered at x with radius ε as

$$B(x; \varepsilon) = \{x \in \mathbb{R}^d : \|x\| < \varepsilon\}.$$

2.1 Convex Analysis

2.1.1 Convexity

We use standard definitions for the convexity of sets and functions.

Definition 2.1.1. *A set C is* convex *if, and only if, (iff)*

$$\forall x_1, x_2 \in C,\ \forall \lambda \in [0, 1],\ (1 - \lambda)x_1 + \lambda x_2 \in C.$$

Another useful notion is that of the epigraph of a function.

Definition 2.1.2. *Given a function $f : \mathbb{R}^d \to \mathbb{R}$, its epigraph* epi$(f)$ *is the set of points in $\mathbb{R}^{d+1}$ that are on or above the graph of f:*

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

A function is defined to be convex iff its epigraph is convex. (Likewise, a function f is concave iff $-f$ is convex, i.e. $\operatorname{epi}(-f)$ is a convex set.)

We will write $\bar{\mathbb{R}}$ to mean the set $\mathbb{R} \cup \{+\infty\}$ of *extended reals*. When working with a function f with bounded domain $\operatorname{dom}(f) \subset \mathbb{R}^d$, it will sometimes be more convenient to instead work with the extended-value function $\bar{f}$, where

$$\bar{f}(x) = \begin{cases} f(x) & \text{, if } x \in \operatorname{dom}(f); \\ +\infty & \text{, if } x \notin \operatorname{dom}(f). \end{cases}$$

We denote the indicator function of a set S by $\iota_S(X)$. This function is defined to be zero for all elements in S, and $+\infty$ for elements outside of S,

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

2.1.2 Affine and Convex Hulls

We recall the definitions of affine and convex hulls for sets in $\mathbb{R}^d$, and for convex hulls of functions.

Fact 2.1.3. *The* affine hull *of a set S is the set of all affine combinations of finite numbers of points in S; that is,*

$$\operatorname{aff}(S) = \left\{ \sum_{i=1}^{n} \lambda_i x_i : x_i \in S \textit{ for } i \in \{1, \ldots, n\},\ \sum_{i=1}^{n} \lambda_i = 1,\ n \geq 1 \right\}.$$

A set is affine *if it is equal to its affine hull.*

Fact 2.1.4. *The* convex hull *of a set S is the smallest convex set that contains S. It is also the intersection of all convex sets containing S. It can be written as*

$$\operatorname{conv}(S) = \left\{ \sum_{i=1}^{n} \lambda_i x_i : x_i \in S,\ \lambda_i \geq 0 \textit{ for } i \in \{1, \ldots, n\},\ \sum_{i=1}^{n} \lambda_i = 1,\ n \geq 1 \right\}.$$

The convex hull is, by definition, a convex set.

The closed convex hull of a function f is defined to be the largest lower semi-continuous, convex function that underestimates f. It is written as $\operatorname{co}(f)$. It can be defined via the epigraph with

$$(\operatorname{co} f)(x) = \inf\{t : (x, t) \in \operatorname{conv} \operatorname{epi} f\}.$$

When we refer to the convex hull of a function, we are referring to its closed convex hull.

2.1.3 Set Interiors

Definition 2.1.5. *The* interior *of a set S, $\operatorname{int}(S)$, is defined as*

$$\operatorname{int}(S) = \{x \in S : B(x;\varepsilon) \subseteq S \textit{ for some } \varepsilon > 0\}.$$

Definition 2.1.6. *The* relative interior *of S, $\operatorname{ri}(S)$, is the interior of S when considered as a subset of its affine hull, that is,*

$$\operatorname{ri}(S) = \{x \in S : B(x;\varepsilon) \cap \operatorname{aff}(S) \subseteq S \textit{ for some } \varepsilon > 0\}.$$

The relative interior will be useful when discussing function domains.

2.1.4 Subdifferential

The subdifferential is a generalization of derivatives to convex but nondifferentiable functions.

Definition 2.1.7. *We define the* subdifferential *of a convex function $f : \mathbb{R}^d \to \bar{\mathbb{R}}$ at a point x to be the set of all* subgradients *s of f at x,*

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

The subdifferential is a multivalued map from $\mathbb{R}^d$ to $\mathbb{R}^d$. We also define the graph of the subdifferential, $\operatorname{gph} \partial f$, as a subset of $\mathbb{R}^d \times \mathbb{R}^d$ by

$$(x, s) \in \operatorname{gph} \partial f \Leftrightarrow s \in \partial f(x).$$

2.1.5 Convex Conjugate

The convex conjugate, also known as the Legendre-Fenchel transform, or just the conjugate, is a fundamental transform in convex analysis. It can be used to construct many other transforms, such as the Moreau envelope and the proximal average.

Definition 2.1.8. *For a proper function $f : \mathbb{R}^d \to \mathbb{R}$, its conjugate is the function $f^\star : \mathbb{R}^d \to \bar{\mathbb{R}}$ defined by*

$$f^\star(y) = \sup_{x \in \operatorname{dom}(f)} \{\langle y, x\rangle - f(x)\}$$

Using extended-value functions, we can write the conjugate without constraints as $f^\star(y) = \sup_{x\in\mathbb{R}^d}\{\langle y, x\rangle - \bar{f}(x)\}$. As $f^\star$ is the supremum of convex functions, it is convex.

Fact 2.1.9. *[20, Theorem 12.2] If f is convex, then the conjugate $f^\star$ is always convex and lower semi-continuous. Furthermore, $f^{\star\star}$ is the closed convex hull of f, and $f^\star$ is proper iff f is proper.*

We highlight several cases where the conjugate is easy to compute.

Fact 2.1.10. *[20, p.106] The only function on $\mathbb{R}^d$ that is self-conjugate, i.e. $f^\star = f$, is $\mathfrak{q}(x) = \frac{1}{2}\|x\|^2$.*

Example 2.1.11. *If $f(x) = ax^2 + bx + c$ for $a > 0,\ b, c \in \mathbb{R}$, then $f^\star(y) = \frac{1}{4a}(y-b)^2 - c$, and $f^{\star\star} = f$.*

Example 2.1.12. *If $f(x) = bx + c$ for $b, c \in \mathbb{R}$, then $f^\star(y) = \iota_{\{b\}}(y) - c$, and $f^{\star\star} = f$.*

Example 2.1.13. *If $f(x) = \frac{1}{p}\|x\|^p$ with $x \in \mathbb{R}^d$, $1 < p < +\infty$, then $f^\star(y) = \frac{1}{q}\|y\|^q$, where $\frac{1}{p} + \frac{1}{q} = 1$.*

2.1.6 Epi-addition and Epi-multiplication

We make use of two further operators, known as *epi-addition* (or *inf-convolution*), and *epi-multiplication*. We denote these, respectively, by:

$$(f \square g)(x) = \inf_{y \in \mathbb{R}^d} \{f(y) + g(x-y)\}$$

$$(\alpha \bigstar f)(x) = \begin{cases} \alpha \cdot f(x/\alpha) & \text{, if } \alpha > 0; \\ \iota_{\{0\}} & \text{, if } \alpha = 0. \end{cases}$$

2.1.7 Moreau Envelope

The *Moreau envelope*, also known as the *Moreau-Yoshida approximate*, is the inf-convolution of a function with $\frac{1}{2\lambda}\|\cdot\|^2$. That is,

$$e_\lambda f(x) = (f \square \lambda^{-1}\mathfrak{q})(x) = \inf_{y \in \mathbb{R}^d} \{f(y) + \frac{1}{2\lambda}\|y - x\|^2\},$$

for $\lambda > 0$. The Moreau envelope is an underestimator of f, which converges pointwise to f as $\lambda \searrow 0$ [21, Theorem 1.25]. By expanding the norm-squared, we can write the Moreau envelope in terms of the conjugate as

$$e_\lambda f(x) = \frac{\|x\|^2}{2\lambda} - \frac{1}{\lambda} g_\lambda^\star(x),$$

where $g_\lambda(x) = (\lambda f + \frac{1}{2}\|\cdot\|^2)(x)$. When f is convex, the Moreau envelope can be written using the conjugate,

$$e_\lambda f = (f^\star + \lambda^{-1}\mathfrak{q})^\star.$$

The *proximal mapping* is the set of minimizers of the Moreau envelope. It is defined as

$$\operatorname{Prox}_\lambda f(x) = \operatorname*{argmin}_{y\in\mathbb{R}^d}\{f(y) + \frac{1}{2\lambda}\|y-x\|^2\}.$$

2.1.8 Proximal Average

The proximal average is an operator for continuously transforming one function into another, and unlike the arithmetic average, it produces a proper function even if the domains of two proper input functions f and g are disjoint. The proximal average is the function whose proximal mapping is the convex combination of the proximal mappings of the input functions [3, Remark 6.2].

Definition 2.1.14. *Given two proper functions f and g, and a parameter $\lambda \in [0,1]$, the* proximal average $P_\lambda(f,g) : \mathbb{R}^d \to \bar{\mathbb{R}}$ *is defined as*

$$P_\lambda(f,g) = \big((1-\lambda)(f+\mu^{-1}\mathfrak{q})^\star + \lambda(g+\mu^{-1}\mathfrak{q})^\star\big)^\star - \mu^{-1}\mathfrak{q}$$

for a smoothing parameter $\mu > 0$.

The proximal average is self-dual, that is,

$$(P_\lambda(f,g))^\star = P_\lambda(f^\star, g^\star).$$

If f and g are convex, then $P_\lambda(f,g)$ is also convex. Convexity with respect to other parameters have been studied in [15]. Recent extensions to the proximal average include the generalized *kernel average* of n functions in [19], as well as the nonconvex proximal average in [13].

The proximal average can be written as a minimization problem using [1, Formula 20],

$$P_\lambda(f,g)(x) = \inf_{x=(1-\lambda)x_1+\lambda x_2}\left((1-\lambda)f(x_1) + \lambda g(x_2) + \frac{(1-\lambda)\lambda}{2\mu}\|x_1-x_2\|^2\right).$$

We call the proximal average *exact* at x if the infimum is attained for $x_1, x_2 \in \mathbb{R}^d$. The proximal average can also be written using Moreau envelopes as in [1, Theorem 8.3],

$$P_\lambda(f,g)(x) = -e_\mu(-(1-\lambda)e_\mu f - \lambda e_\mu g).$$

2.1.9 Self-Dual Smoothing Operators

We will present results on two recently published operators that are self-dual, i.e. they satisfy

$$T(f^\star) = (Tf)^\star.$$

The first is a smoothing operator defined by Goebel in [12] as

$$s_\lambda f = (1-\lambda^2)e_\lambda f + \lambda \mathfrak{q}.$$

The second operator was published in [19] and is defined as

$$T_\lambda f = P_\lambda(f, \mathfrak{q}).$$

In both cases, $\lambda \in (0,1)$.

2.2 Piecewise Linear-Quadratic Functions

Piecewise linear-quadratic (PLQ) functions are piecewise functions $f : \mathbb{R}^d \to \bar{\mathbb{R}}$, where $\operatorname{dom}(f)$ is convex and partitioned into polyhedral pieces, and each piece is assigned a linear or quadratic function. We will additionally assume throughout this thesis that f is continuous on the interior of its domain, and differentiable on each of its pieces, though f may fail to be differentiable at the boundary of each piece.

The class of PLQ functions is well studied [2, 21] because it is closed under many of the standard operations in convex analysis, such as addition, scalar multiplication, conjugation, and the Moreau envelope. When $d = 1$, the class is also closed under the pointwise minimum and maximum operations.

In this thesis, we are specifically concerned with univariate PLQ functions $f : \mathbb{R} \to \bar{\mathbb{R}}$. We divide the real line into n pieces, with the i^{th} piece being defined by $f_i(x) = a_i x + b_i x + c_i$ over the interval $[x_{i-1}, x_i]$, where $-\infty = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = +\infty$. Given such a function, we represent it as the $n \times 4$ matrix

$$\begin{bmatrix} x_1 & a_1 & b_1 & c_1 \\ x_2 & a_2 & b_2 & c_2 \\ \vdots & & & \vdots \\ \infty & a_n & b_n & c_n \end{bmatrix}.$$

Example 2.2.1. *The univariate quadratic energy function, $\mathfrak{q}(x) = \frac{1}{2}x^2$ is represented by the matrix*

$$\begin{bmatrix} \infty & \frac{1}{2} & 0 & 0 \end{bmatrix}.$$

Example 2.2.2. *The function $f(x) = ||x-1|-1|$ is represented by the matrix*

$$\begin{bmatrix} 0 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 2 \\ \infty & 0 & 1 & -2 \end{bmatrix}.$$

As a special case, we support representation of the indicator function for a single point, $f(x) = \iota_{\{\bar{x}\}}(x) + c$, where $\bar{x} \in \mathbb{R}$, with the matrix:

$$\begin{bmatrix} \bar{x} & 0 & 0 & c \end{bmatrix}$$

Any references to PLQ functions from this point forward will be referring to univariate PLQ functions unless stated otherwise.

Chapter 3

Convex Hull Algorithms

Convex hulls form a bridge between the fields of Computational Geometry and Convex Analysis. The former typically focuses on computing the convex hull of a set of points, lines, curves, or other objects in Euclidean space; the latter is concerned with the convex hulls of functions. Fortunately, there is a direct link: representing a function's curve can enable us to use an applicable algorithm from Computational Geometry.

The Beneath-Beyond algorithm is an incremental algorithm for computing the convex hull of a set of points in Euclidean space [7]. In the plane, this algorithm requires $O(n)$ time, given sorted input (this is not an issue, as the first column of the matrix of a PLQ function is always sorted). The convex hull of a piecewise linear function could be calculated by passing to the Beneath-Beyond algorithm the set of points where the function is non-differentiable. Quadratic pieces require special attention, but do not require asymptotically more time. The initial implementation of this algorithm was provided in [22], and it is studied further in [10].

After describing this algorithm, we present a new method for computing convex hulls [10] by decomposing a nonconvex PLQ function into individual convex pieces, conjugating, then applying the maximum operation to those pieces to build the convex hull. As this algorithm stores a model that may grow at each iteration, it takes $O(n^2)$ time in the worst case.

3.1 Direct Approach

The planar Beneath-Beyond algorithm performs a plane sweep and incrementally adds points to, and removes points from, the convex hull. We apply the same technique to a PLQ function f, performing a plane sweep along the x-axis to find the convex hull of the epigraph. At any point in the algorithm, the function to the left of our current position is convex.

Initially, there are a number of cases we must consider, which would cause $\operatorname{co} f$ to be $-\infty$ everywhere.

Proposition 3.1.1. *[10, Proposition 4.2] Let f be a PLQ function. Then*

$\operatorname{co} f \equiv -\infty$ *iff any of the following conditions hold.*

(i) There is an x with $f(x) = -\infty$,

(ii) $a_1 < 0$,

(iii) $a_n < 0$,

(iv) $a_1 = a_n = 0$, and $b_1 > b_n$.

The direct algorithm begins with the leftmost finite piece of f, and progresses rightward. At any point in the algorithm, the function to the left of the current index is convex. In each iteration, two adjacent pieces of f are considered. If those two pieces, treated as a single function, are convex, then we move right, else we convexify the two pieces, backtracking if necessary. A sample run of the algorithm is shown in figure 3.1. Pseudocode is presented in Algorithm 1, which we call plq_coDirect.

Notation 3.1.2. *We write (f_i, f_{i+1}) to denote the i^{th} and $(i+1)^{th}$ pieces of f considered together, i.e. f restricted to $[x_{i-1}, x_{i+1}]$.*

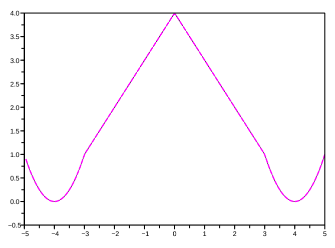
Before we begin the plane sweep, we will simplify f by replacing any nonconvex pieces of f with their convex hulls. This simplifies the cases we must consider to convexify each pair of pieces. A single piece f_i is only nonconvex on its domain if $a_i < 0$, in which case we can replace the piece with the line segment between its two endpoints.

We also need a test to know when two adjacent pieces are convex together. The following proposition treats this case.

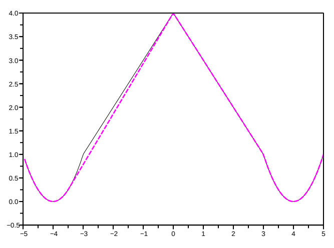
Proposition 3.1.3. *[22, Theorem 3.34] Let f be a continuous PLQ function such that $\operatorname{co} f \not\equiv -\infty$. Then f is nonconvex iff there are two adjacent pieces f_i (on $[x_{i-1}, x_i]$) and f_{i+1} (on $[x_i, x_{i+1}]$) such that $f_i'(x_i) > f_{i+1}'(x_i)$.*

In order to construct the convex hull of two adjacent PLQ functions on line 21 of Algorithm 1, three cases must be considered. If both pieces f_i, f_{i+1} are quadratic (that is, $a_i, a_{i+1} > 0$), then the following system must be solved for x_α, x_β to find a tangent line joining the pieces.

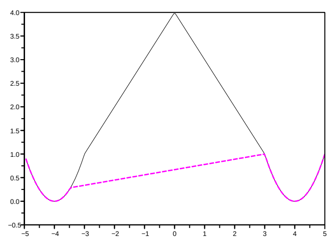
$$f_i'(x_\alpha) = f_{i+1}'(x_\beta) \tag{3.1}$$
$$f_i'(x_\alpha)(x - x_\alpha) + f_i(x_\alpha) = f_{i+1}'(x_\beta)(x - x_\beta) + f_{i+1}(x_\beta) \tag{3.2}$$
$$x_\alpha \in [x_{i-1}, x_i] \tag{3.3}$$
$$x_\beta \in [x_i, x_{i+1}] \tag{3.4}$$



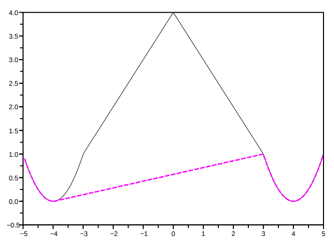
(a) Initial, nonconvex PLQ function. The first two pieces are nonconvex; convexify and move forward.



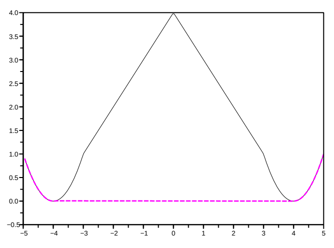
(b) Two linear pieces are nonconvex. Collapse to one piece and backtrack.



(c) Quadratic-linear nonconvex case. Convexify; the slope at -5 is unchanged, so advance.



(d) Linear-quadratic case. We need to backtrack again.



(e) Several more iterations lead to the convex hull.

Figure 3.1: Demonstration of the steps of the plq_coDirect algorithm.

Algorithm 1: plq_coDirect

```
Input: $f$ (a PLQ function)
Output: $\operatorname{co} f$ (a convex PLQ function)
begin
    if any conditions in Proposition 3.1.1 hold then
        $\operatorname{co} f \leftarrow -\infty$
        return
    end
    for $i \leftarrow 1$ to $n$ do
        if $a_i < 0$ then
            Replace the $i^{\text{th}}$ piece of $f$ with the linear function between $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$;
        end
    end
    // Plane sweep.
    $i_0 \leftarrow i \leftarrow$ the index of the first finite piece of $f$;
    $i_f \leftarrow$ the index of the last finite piece of $f$;
    // Let $N$ denote the number of finite pieces of $f$.
    while $(i \leq i_f - 1)$ and $(N \geq 2)$ do
        if $f_i'(x_i) \leq f_{i+1}'(x_i)$ then
            // $(f_i, f_{i+1})$ are convex.
            $i \leftarrow i + 1$; // Add $f_i$ to the convex hull and advance.
        else
            // Convexify $f_i$ and $f_{i+1}$.
            Replace $(f_i, f_{i+1})$ with $\operatorname{co}((f_i, f_{i+1}))$ on $[x_{i-1}, x_{i+1}]$ in $f$;
            Update $i_f$ according to inserted/removed pieces;
            if $f_{i-1}'(x_{i-1}) > f_i'(x_{i-1})$ then
                // $(f_{i-1}, f_i)$ has lost convexity.
                $i \leftarrow \max(i - 1, 1)$;
            else
                $i \leftarrow$ the index of the rightmost piece in $[x_{i-1}, x_{i+1}]$;
                // $f$ is now convex on $[x_{i_0}, x_i]$.
            end
        end
    end
    $\operatorname{co} f \leftarrow f$;
end
```

In this case, an additional linear piece will be inserted into f on $[x_\alpha, x_\beta]$, possibly replacing one or both quadratic pieces when $x_\alpha = x_{i-1}$ or $x_\beta = x_{i+1}$. A quadratic-quadratic convex hull is demonstrated in Figure 3.2.

If f_i is quadratic and f_{i+1} is linear, then we find a tangent line connecting f_i at x_α to the right endpoint of f_{i+1}. We set f to be this tangent line on $[x_\alpha, x_{i+1}]$. Note that x_α may be x_{i-1}, in which case the quadratic piece will be removed. Symmetry handles the case where f_i is linear and f_{i+1} is quadratic; see Figure 3.3. The system we now solve is

$$\begin{aligned} f_i'(x_\alpha)(x_{i+1} - x_\alpha) + f_i(x_\alpha) &= f_{i+1}(x_{i+1}) \\ x_\alpha &\in [x_{i-1}, x_i] \end{aligned} \tag{3.5}$$

Care must be given to the linear-quadratic case. After solving a linear-quadratic system and convexifying (f_i, f_{i+1}), if we have lost convexity at x_{i-1}, we first check to see if f_{i-1} is quadratic. If it is, then we may enter a cycle of alternately convexifying (f_{i-1}, f_i) and (f_i, f_{i+1}), converging to the convex hull of f on $[x_{i-1}, x_{i+1}]$. As the linear function f_i is repeatedly shifted downward, we treat this as a special case and remove f_i from the model, solving the quadratic-quadratic system (Equations 3.1–3.4) to find the tangent line for f_{i-1} and f_{i+1}.

The simplest case to handle is when both f_i and f_{i+1} are linear functions. We are convexifying because (f_i, f_{i+1}) are nonconvex, implying that $b_i > b_{i+1}$. Here we replace both linear pieces with a single linear function connecting the points $(x_{i-1}, f_i(x_{i-1}))$ and $(x_{i+1}, f_{i+1}(x_{i+1}))$. Figure 3.4 provides an example of this case.

Special attention may be necessary in implementing the last two cases, when a linear piece extends to $\pm\infty$ (e.g. $x_{i+1} = \infty$ in (3.5)). These cases are handled by finding a line whose slope is that of the unbounded piece. Examples of these cases are shown in Figures 3.3(b), 3.4(b), and 3.4(c).

After inserting $\text{co}((f_i, f_{i+1}))$ into f, we would like to increment the index i and consider the next two pieces in the function. By Proposition 3.1.3, f is convex iff its slope increases at each of the breaks in its domain. The function (f_i, f_{i+1}) is now convex, however, if the slope of f approaching x_i from the right has decreased, then (f_{i-1}, f_i) may fail to be convex. Hence the algorithm must backtrack and reconsider (f_{i-1}, f_i).

Lemma 3.1.4. *With consecutive backtracks, every backtrack after the first reduces the two pieces currently being considered to a single linear piece.*

Proof. After convexifying (f_i, f_{i+1}), Algorithm 1 only backtracks when necessary, i.e. when $f_i'(x_{i-1})$ decreases sufficiently. The only time $f_i'(x_{i-1})$

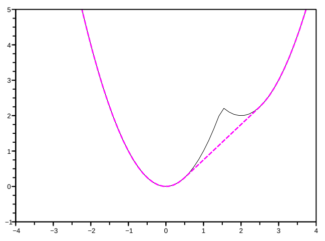
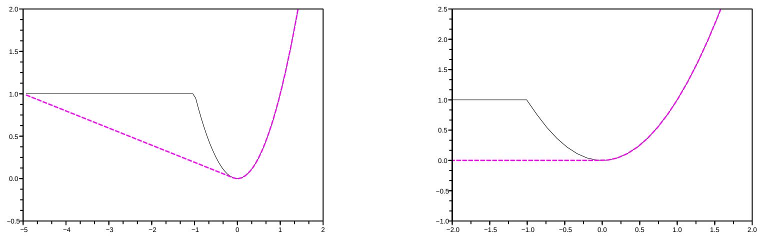
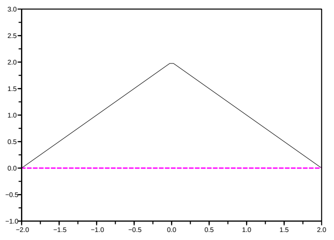


Figure 3.2: Quadratic-quadratic convex hull case.

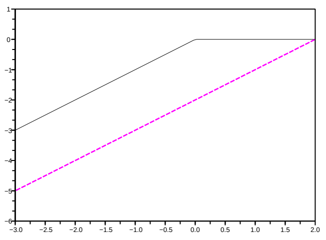

(a) Bounded linear piece.

(b) Unbounded linear piece.

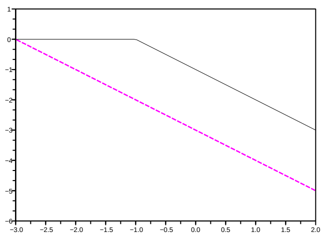
Figure 3.3: Linear-quadratic convex hull case.



(a) Bounded convex hull.



(b) Convex hull unbounded on the left.



(c) Convex hull unbounded on the right.

Figure 3.4: Linear-linear convex hull case.

decreases is when a linear piece is inserted on some interval $[x_{i-1}, x_\beta]$, for $x_{i-1} < x_\beta \leq x_i$.

Hence, after backtracking, the right piece will be linear. After the first backtrack, we will be in either the quadratic-linear case or the linear-linear case. The only way backtracking can happen in these cases is if the current two pieces are replaced by a single linear piece. By induction, this holds for all consecutive backtracks beyond the first. □

Proposition 3.1.5. *Algorithm 1 runs in optimal linear time and space.*

Proof. Assume $\operatorname{co} f \not\equiv -\infty$. Convexifying pieces in lines 6–10 requires $O(1)$ time for each piece. To show that the while loop on lines 15–31 requires linear time, note again that each iteration requires constant time. The while loop terminates either when we reach the rightmost finite piece of f, or when f contains a single finite piece. In every iteration, we make progress to at least one of these conditions.

Moving right will lead to the first condition being satisfied. If we don't move right, then we have lost convexity at x_{i-1}. There are four subcases to consider, based on the types of the original f_i, f_{i+1} being convexified.

(i) *Linear-linear:* Taking the convex hull of two linear pieces always produces a single linear piece on $[x_{i-1}, x_{i+1}]$, so progress is made toward N reaching 1.

(ii) *Quadratic-linear:* The only way for Algorithm 1 to decrease $f_i'(x_{i-1})$ is to insert a linear piece extending rightward from x_{i-1}. As f_{i+1} is linear, this means again that (f_i, f_{i+1}) has collapsed to a single linear piece.

(iii) *Quadratic-quadratic:* As in case (ii), a linear piece has been inserted whose left endpoint is x_{i-1}. In the quadratic-quadratic case we look for the linear function tangent to both quadratic pieces, so consider the right endpoint of the linear function, x_β. If $x_\beta = x_{i+1}$, then again $\operatorname{co}((f_i, f_{i+1}))$ is now a single linear piece. If $x_\beta < x_{i+1}$, then the convex hull is a linear-quadratic function, which is handled in the next case (after convexification).

(iv) *Linear-quadratic:* If $x_\beta = x_{i+1}$, then (f_i, f_{i+1}) collapses to a single piece. Otherwise, $\operatorname{co}((f_i, f_{i+1}))$ still contains two pieces. As we have lost convexity at x_{i-1}, consider the type of f_{i-1}.

If f_{i-1} is quadratic, then backtracking as usual will lead to a cycle of repeatedly convexifying (f_{i-1}, f_i) and (f_i, f_{i+1}), so remove the linear piece and solve the quadratic system instead.

If f_{i-1} is linear, then it is necessary to backtrack without decrementing the number of pieces. We claim that this can happen at most once before adding f_{i+1} to the convex hull. Fix $\bar{x}$ as the current value of x_{i-1}, and let f_j denote, across all iterations, the piece of f whose right endpoint is the value of x_i before any backtracking. (Currently, $f_j = f_i$.) By Lemma 3.1.4, each consecutive backtrack will now decrease N, and the right piece in all these backtracks will be f_j.

When consecutive backtracking is no longer necessary, we will again consider having convexified (f_j, f_{j+1}) in the linear-quadratic case, and assume f is still nonconvex at x_{j-1}. We enumerate the reasons why backtracking stopped.

(a) If $j = 1$, then we have convexified (f_1, f_2), and consequently move forward (we cannot backtrack any further).

(b) If f_{j-1} is quadratic, then we the use quadratic-quadratic system as above, and either move forward, or when the two quadratic pieces collapse to one, backtrack.

(c) If f_{j-1} is linear, it may be necessary to backtrack again without collapsing (f_j, f_{j+1}). Note that, as there was at least one consecutive backtrack to get to this point, x_{j-1} is now strictly less than $\bar{x}$, and also that x_{j-1} is one of the partition points that was in f when we assigned $\bar{x}$ (backtracking cannot introduce new partition points). Hence there is a finite number of partition points that x_{j-1} can take on in this case, and once x_{j-1} moves left beyond a partition point, it is deleted from the model and cannot be considered by a future linear-quadratic case.

Thus, though a single linear-quadratic piece may require $O(n)$ time, partition points removed from the model will not be considered by future linear-quadratic cases, so $O(n)$ work is done by all linear-quadratic cases.

The other three cases always progress toward either $i = i_f$ or $N = 1$, so overall, Algorithm 1 runs in linear time.

The function f begins with $O(n)$ pieces, and each iteration adds at most one piece (a tangent line between two quadratics), or deletes at most one piece, hence Algorithm 1 requires linear space as well.

□

3.2 Convex Duality Approach

An alternate method that we present to compute the convex hull is based on dividing a PLQ function into individual pieces, each of which is convex. Consider a PLQ function f with n pieces. Assuming $\operatorname{co} f \not\equiv -\infty$, this method will also assume that each individual piece is convex (convexifying individual pieces may initially be necessary, as with the direct algorithm). We name this algorithm plq_coSplit.

Define the functions $(g_i)_{i=1}^n$ by

$$g_i(x) = \begin{cases} f_i(x) & \text{, if } x \in [x_{i-1}, x_i] \\ +\infty & \text{, otherwise} \end{cases}$$

Each g_i is the extended-value function of f_i restricted to $[x_{i-1}, x_i]$. We can write $f = \min g_i$. Also, as each g_i is convex, $\operatorname{co} g_i = g_i$. Hence we can write $\operatorname{co} f$ as

$$\operatorname{co} f = \operatorname{co} \min g_i = \operatorname{co} \min \operatorname{co} g_i.$$

From here, we apply the following theorem.

Theorem 3.2.1. *[14, Theorem X.2.4.4] Let $g_1, \ldots, g_n$ be proper, lower semi-continuous functions. Then*

$$(\operatorname{co} \max(g_1, \ldots, g_n))^\star = \operatorname{co} \min(g_1^\star, \ldots, g_n^\star).$$

This holds for an arbitrary finite number of functions.

By applying Theorem 3.2.1 with [14, Proposition X.2.6.1], we rewrite the convex hull of f as

$$\operatorname{co} f = \operatorname{co} \min(\operatorname{co} g_1, \ldots, \operatorname{co} g_n) = (\max(g_1^\star, \ldots, g_n^\star))^\star. \tag{3.6}$$

This formulation provides a basis for computing the convex hull of f. Doing this requires the ability to compute the conjugate of already-convex functions, and to construct the pointwise maximum of n convex functions. The conjugate of each g_i is easily found via an explicit formula in [2, Proposition 5.2]. The complete method is presented in Algorithm 2.

As the difference in line counts suggests, Algorithm 2 requires less work to implement, provided that algorithms are already available for computing the maximum and conjugate of convex PLQ functions.

Algorithm 2: plq_coSplit

```
   Input: $f$ (a PLQ function)
   Output: $\operatorname{co} f$ (a convex PLQ function)
1  begin
2      Check if $\operatorname{co} f \equiv -\infty$, and return if necessary;
3      Convexify individual nonconvex pieces of $f$;
4      $g \leftarrow -\infty$;
5      for $i \leftarrow 1$ to $n$ do
6          Construct $g_i$ as above;
7          $g \leftarrow \max(g, g_i^\star)$;
8      end
9      $\operatorname{co} f \leftarrow g^\star$;
10 end
```

Proposition 3.2.2. *Algorithm 2 has linear space complexity, worst-case quadratic time complexity, and best-case linear time complexity.*

Proof. Each g_i contains a single piece, hence $g_i^\star$ will contain $O(1)$ pieces. Every iteration, g will have $O(1)$ partition points added to it, in the case where the partition points in $g_i^\star$ are not partition points of g. After n iterations, g will contain $O(n)$ pieces. As every call to max is a linear operation, across all calls to max we take $O(n^2)$ time.

If every g_i after the first introduces no new pieces into g, then each call to max will take constant time, making the algorithm run in linear time. This is the case, e.g., when computing the convex hull of a piecewise affine interpolation of the function $f(x) = -\frac{1}{2}|x|^2$. ☐

3.3 Performance Comparison

To compare the plq_coDirect and plq_coSplit algorithms, we compute the convex hull of a piecewise affine function with an increasing number of pieces. The results may be seen in Figures 3.5 and 3.6.

Figure 3.5 confirms that the worst-case complexity is attained for both algorithms when computing $\operatorname{co}(\frac{1}{2}\|\cdot\|^2)$, with plq_coDirect running in linear time and plq_coSplit showing quadratic performance. As the model of the convex hull does grow to contain n pieces, quadratic growth is expected. In contrast, when computing $\operatorname{co}(-\frac{1}{2}\|\cdot\|^2)$ in Figure 3.6, plq_coSplit's model does not grow, and ultimately a model with a single piece is produced (a linear function between the bounds of the domain).

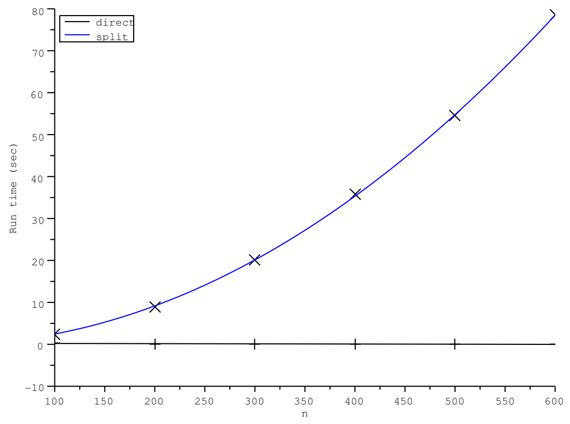


Figure 3.5: Run time (seconds) of the convex hull algorithms computing the convex hull of an n-piece piecewise linear approximation of $f(x) = \frac{1}{2}x^2$ on $[-n/2, n/2]$.

In both graphs, plq_coDirect performs significantly better. Besides quadratic worse-case complexity, plq_coSplit is also making use of entire lower-level algorithms, so overhead invoking these subroutines certainly contributes to plq_coSplit's poorer performance.

All experiments were run on a quad-core 64-bit 2.0GHz HP HDX 18 laptop with 4GB RAM, running Gentoo Linux 10.0. Implementation was done under Scilab 5.1.1, extending the Computational Convex Analysis toolbox freely available from [18].

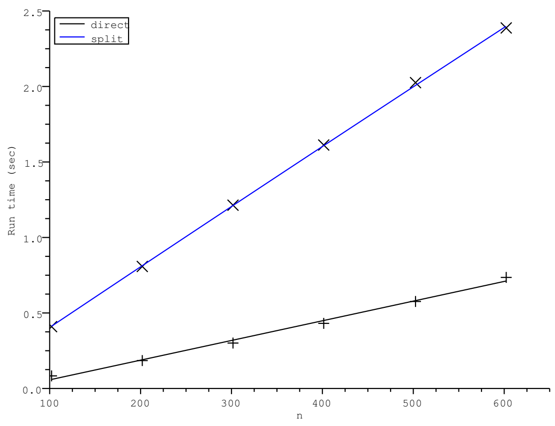


Figure 3.6: Run time (seconds) of the convex hull algorithms computing the convex hull of an n-piece piecewise linear approximation of $f(x) = -\frac{1}{2}x^2$ bounded on $[-n/2, n/2]$.

Chapter 4

Graph-Matrix Calculus

We will now present a new class of algorithms for computing transforms of convex functions, based on unary "calculus rules" [12] for the subdifferential of a convex function. We recall these rules, then provide formulas for additional unary operators, and two binary operators. We then apply these rules to PLQ functions, and describe how to recover an exact model of a function from a transformed subdifferential.

4.1 Representation of PLQ Functions

Ultimately, we wish to be able to perform a transform of a convex PLQ function f,

$$g = Tf,$$

by looking at the graph of the subdifferential of f,

$$\operatorname{gph} \partial f = \{(x, y) : x \in \operatorname{dom} f,\ y \in \partial f(x)\},$$

and computing the subdifferential of the transform via a matrix multiplication

$$\operatorname{gph} \partial(Tf) = A \operatorname{gph} \partial f,$$

for some matrix $A \in \mathbb{R}^{2d \times 2d}$.

For univariate PLQ functions, $A \in \mathbb{R}^{2\times 2}$, and $\operatorname{gph} \partial f$ must be represented in a way suitable for multiplying by A. The data structure used for this graph must also contain a constant of integration, to be able to use integration to recover the PLQ matrix model, and evaluate the function at points in its domain. Conversely, differentiation is able to convert a PLQ matrix model into what we call a GPH matrix,

$$\begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_n & x_{n+1} \\ s_0 & s_1 & s_2 & \dots & s_n & s_{n+1} \\ f_0 & f_1 & f_2 & \dots & f_n & f_{n+1} \end{bmatrix}.$$

The subdifferential of a PLQ function, when single-valued, is a piecewise affine function, and the subdifferential of a convex PLQ function is made up of line segments connected end-to-end that extend to infinity in both directions. We note x_i points in $\operatorname{dom} f$, s_i subgradients of f at x_i, and f_i constants of integration ($f_i = f(x_i)$). The matrix above stores enough points (x_i, s_i) on this graph to be able to recover the entire graph of the subdifferential via piecewise linear interpolation of (x_i, s_i) for all $i \in \{0, \ldots, n+1\}$.

The columns on the GPH matrix are stored in lexicographically increasing order. That is, for $i \leq j$, we have $x_i \leq x_j$, and also $s_i \leq s_j$. If $x_i = x_j$ then $f_i = f_j$. The values $\{x_1, \ldots, x_n\}$ are divisions between the pieces of f, though they are not necessarily distinct: if f is nondifferentiable at a break $\bar{x}$, then there will be two columns $i < j$ with $x_i = x_j = \bar{x}$, and $s_i < s_j$ will be the slopes of f approaching $\bar{x}$ from the left and right, respectively.

The values x_0 and x_{n+1} are in the matrix to store the subdifferential of f on $(-\infty, x_1]$ and $[x_n, +\infty)$, respectively. The line segment from (x_1, s_1) to (x_0, s_0) is extrapolated to infinity, and correspondingly with (x_n, s_n) and (x_{n+1}, s_{n+1}). To represent f being bounded on the left (resp. right), $x_0 = x_1$ (resp. $x_{n+1} = x_n$) indicates a ray extending vertically to $-\infty$ (resp. $+\infty$). In this case, we set f_0 (resp. f_{n+1}) to $+\infty$. Except for f_0 and f_{n+1}, all other f_i and all x_i, s_i must be finite.

Example 4.1.1. *The quadratic energy function may be represented by the GPH matrix*

$$\begin{bmatrix} 0 & 2 \\ 0 & 4 \\ 0 & 4 \end{bmatrix}.$$

Example 4.1.2. *The absolute value function may be encoded as the GPH matrix*

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

Example 4.1.3. *The function where $f(x) = 0$ on $x \in [-1, 1]$ and $f(x) = x^2 - 1$ elsewhere may be stored as*

$$\begin{bmatrix} -2 & -1 & -1 & 1 & 1 & 2 \\ -4 & -2 & 0 & 0 & 2 & 4 \\ 3 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

Example 4.1.4. *The function where $f(x) = x^2$ for $x < 0$, $f(x) = 0$ for $x \in [0, 1]$, and $f(x) = x - 1$ for $x > 1$ may be represented by the matrix*

$$\begin{bmatrix} -1 & 0 & 1 & 1 & 2 \\ -2 & 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

There is no unique GPH matrix representation for a PLQ function. Redundant points may be stored in the middle of the matrix, and the two endpoints (x_0, s_0) and (x_{n+1}, s_{n+1}) can be slid along their respective rays. While normalized GPH rules could be set, this would require normalization after many transformations, as the transformation matrix A can change $\operatorname{gph} \partial f$ drastically. Frequent cleaning could incur significant overhead, so we instead suggest that a separate algorithm be provided to "clean" the GPH matrix when desired, removing duplicate points.

To represent the special case of the point-indicator function $f(x) = \iota_{\{\bar{x}\}}(x) + c$, we use the matrix

$$\begin{bmatrix} \bar{x} & \bar{x} \\ s_0 & s_1 \\ c & c \end{bmatrix} \qquad \text{(with any } s_0 < s_1\text{).}$$

This matrix is the vertical line at $x = \bar{x}$. An affine function $f(x) = ax + b$ is stored as a matrix whose graph is a horizontal line,

$$\begin{bmatrix} x_0 & x_1 \\ a & a \\ ax_0 + b & ax_1 + b \end{bmatrix} \qquad \text{(with any } x_0 < x_1\text{).}$$

Likewise, a quadratic $f(x) = ax^2 + bx + c$ is encoded as

$$\begin{bmatrix} x_0 & x_1 \\ 2ax_0 + b & 2ax_1 + b \\ ax_0^2 + bx_0 + c & ax_1^2 + bx_1 + c \end{bmatrix} \qquad \text{(with any } x_0 < x_1\text{).}$$

Given a matrix

$$\begin{bmatrix} x_0 & x_1 \\ s_0 & s_1 \\ f_0 & f_1 \end{bmatrix},$$

if $x_0 < x_1$ and $s_0 < s_1$, then f is a quadratic function that can be recovered with the following formulas.

$$\begin{aligned} f(x) &= ax^2 + bx + c \\ a &= \frac{s_1 - s_0}{2(x_1 - x_0)} \\ b &= s_0 - 2ax_0 \\ c &= y_0 - ax_0^2 - bx_0 \end{aligned}$$

The case where $x_0 = x_1$ and $s_0 = s_1$ does not form a valid GPH matrix, as two distinct points are required to extrapolate to a maximal graph.

Considering that $\operatorname{gph} \partial f$ is stored in the first two rows of the GPH matrix, we are now able to calculate $A \operatorname{gph} \partial f$ with a simple matrix multiplication.

4.2 Goebel's Graph-Matrix Calculus

Numerous calculus rules were originally published in [12], then slightly extended in [11], showing how the graph of the subdifferential of a convex function is affected under core convex transformations operating on a single function. Specifically, *(i)*, *(ii)*, *(iii)*, *(iv)*, *(vii)*, and *(viii)* were given in [12], and *(v)*, *(vi)*, and *(ix)* were introduced in [11].

Theorem 4.2.1 (Goebel's Graph-Matrix Calculus for unary operators)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a proper, lower semi-continuous, convex function. Then the following rules hold, for all $\alpha > 0$, $\beta \geq 0$, and $\lambda \in [0,1]$.*

(i) $\operatorname{gph} \partial(f^\star) = \begin{bmatrix} 0 & \mathrm{Id} \\ \mathrm{Id} & 0 \end{bmatrix} \operatorname{gph} \partial f.$

(ii) $\operatorname{gph} \partial(f + \beta\mathfrak{q}) = \begin{bmatrix} \mathrm{Id} & 0 \\ \beta\,\mathrm{Id} & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(iii) $\operatorname{gph} \partial(\alpha f) = \begin{bmatrix} \mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(iv) $\operatorname{gph} \partial(e_\beta f) = \begin{bmatrix} \mathrm{Id} & \beta\,\mathrm{Id} \\ 0 & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(v) $\operatorname{gph} \partial(\alpha \bigstar f) = \begin{bmatrix} \alpha\,\mathrm{Id} & 0 \\ 0 & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(vi) $\operatorname{gph} \partial f(\alpha\cdot) = \begin{bmatrix} \alpha^{-1}\,\mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(vii) $\operatorname{gph} \operatorname{Prox}_\beta(f) = \begin{bmatrix} \mathrm{Id} & \beta\,\mathrm{Id} \\ \mathrm{Id} & 0 \end{bmatrix} \operatorname{gph} \partial f.$

(viii) $\operatorname{gph} \partial(s_\lambda f) = \begin{bmatrix} \mathrm{Id} & \lambda\,\mathrm{Id} \\ \lambda\,\mathrm{Id} & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

(ix) $\operatorname{gph} \partial(T_\lambda f) = \begin{bmatrix} (1 - \frac{\lambda}{2})\,\mathrm{Id} & \frac{\lambda}{2}\,\mathrm{Id} \\ \frac{\lambda}{2}\,\mathrm{Id} & (1 - \frac{\lambda}{2})\,\mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f.$

Proof. Let $J = \begin{bmatrix} 0 & \mathrm{Id} \\ \mathrm{Id} & 0 \end{bmatrix}$. Formula *(i)* is shown via

$$(x,y) \in \operatorname{gph} \partial f \Leftrightarrow y \in \partial f(x) \Leftrightarrow x \in \partial f^\star(y) \Leftrightarrow (y,x) \in \operatorname{gph} \partial f.$$

For *(ii)*, we note that $\partial(f + \beta\mathfrak{q}) = \partial f + \beta\partial\mathfrak{q} = \partial f + \beta \,\mathrm{Id}$, thus $(x,y) \in \operatorname{gph} \partial f \Leftrightarrow (x, y + \beta x) \in \operatorname{gph} \partial(f + \beta\mathfrak{q})$. We get *(iii)* using $\partial(\alpha f) = \alpha(\partial f)$:

$$y \in \partial f(x) \Leftrightarrow \alpha y \in \partial(\alpha f)(x) \Leftrightarrow \alpha y \in \alpha\partial f(x) \Leftrightarrow (x, \alpha y) \in \operatorname{gph} \partial f.$$

Similarly, $\partial(e_\beta f) = \partial(f \Box \beta^{-1}\mathfrak{q}) = \partial(f^\star + \beta\mathfrak{q})^\star$, and using previous rules yields *(iv)*. To obtain *(v)*, we use the identity $\alpha \bigstar f = (\alpha f^\star)^\star$, and applying the two previous rules gives

$$\begin{aligned} \operatorname{gph} \partial(\alpha f^\star)^\star &= J \operatorname{gph} \partial(\alpha f^\star) = J \begin{bmatrix} \mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f^\star = J \begin{bmatrix} \mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} J \operatorname{gph} \partial f \\ &= \begin{bmatrix} \alpha\,\mathrm{Id} & 0 \\ 0 & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f. \end{aligned}$$

Using both multiplication rules *(iii)* and *(v)*, and rewriting $f(\alpha\cdot)$ as $\alpha(\alpha^{-1}\bigstar f)$ leads to

$$\begin{aligned} \operatorname{gph} \partial f(\alpha\cdot) &= \operatorname{gph} \partial(\alpha(\alpha^{-1}\bigstar f)) = \begin{bmatrix} \mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} \begin{bmatrix} \alpha^{-1}\,\mathrm{Id} & 0 \\ 0 & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f \\ &= \begin{bmatrix} \alpha^{-1}\,\mathrm{Id} & 0 \\ 0 & \alpha\,\mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f, \end{aligned}$$

thus showing *(vi)*. From [5, Section 2.2], we have $\operatorname{Prox}_\beta f = (\mathrm{Id} + \beta\partial f)^{-1}$, so

$$\operatorname{gph}(\mathrm{Id} + \beta\partial f) = \begin{bmatrix} \mathrm{Id} & 0 \\ \mathrm{Id} & \beta\,\mathrm{Id} \end{bmatrix},$$

and inverting the graph gives *(vii)*. By writing out the definition of $s_\lambda f$ and expanding based on previous rules,

$$\begin{aligned} \operatorname{gph} \partial(s_\lambda f) &= \operatorname{gph} \partial((1-\lambda^2)e_\lambda f + \lambda\mathfrak{q}) \\ &= \begin{bmatrix} \mathrm{Id} & 0 \\ \lambda\,\mathrm{Id} & \mathrm{Id} \end{bmatrix} \begin{bmatrix} \mathrm{Id} & 0 \\ 0 & (1-\lambda^2)\,\mathrm{Id} \end{bmatrix} \begin{bmatrix} \mathrm{Id} & \lambda\,\mathrm{Id} \\ 0 & \mathrm{Id} \end{bmatrix} \operatorname{gph} \partial f, \end{aligned}$$

and matrix multiplication proves *(viii)*. The proof of *(ix)* is postponed until after the next theorem. □

We also recall rules in [11] for calculation of the transformed subdifferential under the binary operations of addition, infimal convolution, and taking the proximal average.

Theorem 4.2.2 (Graph-Matrix Calculus for binary operators)**.**
Let f_1 and f_2 be two proper, lower semi-continuous, convex functions. Then the following rules hold (for $i \in \{1,2\}$).

(i) **Addition rule:** *If* $\operatorname{ri}\operatorname{dom} f_1 \cap \operatorname{ri}\operatorname{dom} f_2 \neq \emptyset$*, then*

$$(x,s) \in \operatorname{gph}\partial(f_1+f_2) \Leftrightarrow \exists (x_i,s_i) \in \operatorname{gph}\partial f_i \textit{ such that } \begin{cases} x = x_1 = x_2, \\ s = s_1 + s_2. \end{cases}$$

(ii) **Inf-convolution rule:** *If* $\partial f_1(x_1) \cap \partial f_2(x_2) \neq \emptyset$*, then*

$$(x_1+x_2, s) \in \operatorname{gph}\partial(f_1 \Box f_2) \Leftrightarrow (x_i, s) \in \operatorname{gph}\partial f_i.$$

(iii) **Proximal average rule:** *Let $\lambda \in (0,1)$ and $\mu = 1$. Assume that $P_\lambda(f_1,f_2)$ is exact at $x = (1-\lambda)x_1 + \lambda x_2$, where $x_i \in \operatorname{dom} f_i$ for $i \in \{1,2\}$. Then:*

$$(x,s) \in \operatorname{gph}\partial P_\lambda(f_1,f_2) \Leftrightarrow \begin{cases} s_1 \in \partial f_1(x_1) \\ s_2 \in \partial f_2(x_2) \\ s = x_1 + s_1 - x = x_2 + s_2 = x \end{cases}$$

Proof. As f_1, f_2 are convex, we have by [14, Corollary XI.3.1.2]: $\partial f_1(x) + \partial f_2(x) = \partial(f_1+f_2)(x)$. Hence,

$$s \in \partial(f_1+f_2)(x) \Leftrightarrow s = s_1 + s_2 \text{ for } s_i \in \partial f(x_i).$$

Similarly, [14, Corollary XI.3.4.2] gives $\partial f_1(x_1) \cap \partial f_2(x_2) = \partial(f_1 \Box f_2)(x)$, so

$$s \in \partial(f_1 \Box f_2)(x) \Leftrightarrow \exists x_1, x_2,\ s \in \partial f_1(x_1) \text{ and } s \in \partial f_2(x_2).$$

Using $\partial P_\lambda(f_1,f_2)(x) = -x + \bigcap_{i:\lambda_i>0}(\partial f_i(x_i) + x_i)$ from [1, Theorem 7.1],

$$s \in \partial P_\lambda(f_1,f_2)(x) \Leftrightarrow \begin{cases} s = x_1 + s_1 - x & \text{, for some } s_1 \in \partial f_1(x_1); \\ s = x_2 + s_2 - x & \text{, for some } s_2 \in \partial f_2(x_2). \end{cases}$$

□

We now include the proof of Theorem 4.2.1*(ix)*.

Proof. Using the definition of $T_\lambda f = P_\lambda(f, \mathfrak{q})$, and that $s_2 \in \partial\mathfrak{q}(x_2) \Leftrightarrow s_2 = x_2$, we have

$$(x,s) \in \operatorname{gph}\partial(T_\lambda f) \Leftrightarrow \begin{cases} x = & (1-\lambda)s_1 + \lambda s_2 \\ s = & x_1 + s_1 - x \\ s = & x_2 + s_2 - x \end{cases}$$

Solving for x and s yields

$$\begin{aligned} x &= (1-\frac{\lambda}{2})x_1 + \frac{\lambda}{2}s_1 \\ s &= \frac{1}{2}x_1 + (1-\frac{\lambda}{2})s_1 \end{aligned}$$

which are the coefficients in Theorem 4.2.1*(ix)*. □

4.3 Transforms of PLQ Functions

Given a convex PLQ function f in GPH matrix format, Theorem 4.2.1 shows how to compute many of the fundamental unary convex transforms. Having computed $A \operatorname{gph}\partial f$, we still need to recover at least one constant of integration to be able to make use of this new graph. We present ways to transform f_i values from ∂f's graph to new values in $A\partial f$'s graph. Unlike the matrices presented by Theorem 4.2.1, the method to compute new f_i's depends highly on the transform being performed.

We set our notations first. We will assume we are transforming a convex function f given by the GPH matrix $[x_f, s_f, y_f]^T$, and that we have calculated x_g, s_g for the transformed function $g = Tf$ given by $[x_g, s_g, y_g]^T$. The notation $x\,.\!*y$ denotes elementwise multiplication, and $x\,.\hat{}\,n$ denotes elementwise exponentiation.

Recovery of y_g when $g = \alpha f$ or $g = f + \beta\mathfrak{q}$ is straightforward: $y_g = \alpha y_f$ in the first case, and $y_g = y_f + (\beta/2)x_f\,.\hat{}\,2$ in the second. The following two Propositions show how to compute the transforms of other unary operators.

Proposition 4.3.1. *To compute the Legendre-Fenchel transform, $g = f^\star$, y_g can be calculated via $y_g = s_f\,.\!*x_f - y_f$.*

Proof. As f is convex, looking for a maximizer of $g(s) = f^\star(s) = \max_x(\langle s, x\rangle - f(x))$ means finding x, s such that $s \in \partial f(x)$. For all $i \in \{0, \ldots, n+1\}$, we know that $(s_f)_i \in \partial f((x_f)_i)$, hence $y_g = g(x_g) = g(s_f) = s_f\,.\!*x_f - y_f$. □

Proposition 4.3.2. *Under the Moreau envelope $g = e_\beta f$, we have that $y_g = y_f + (\beta/2)s_f\,.\hat{}2$.*

Proof. The Moreau envelope can be written as $e_\beta f = (f^\star + \beta^{-1}\mathsf{q})^\star$. By the previous Proposition as well as Theorem 4.2.1*(i)*,

$$f^\star := \begin{bmatrix} x_1 \\ s_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} s_f \\ x_f \\ s_f\,.{*}x_f - y_f \end{bmatrix}.$$

Then, if we consider the function $f^\star + \beta^{-1}\mathsf{q}$,

$$f^\star + \beta^{-1}\mathsf{q} := \begin{bmatrix} x_2 \\ s_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ s_1 + (1/\beta)x_1 \\ y_1 + (1/2\beta)x_1\,.\hat{}2 \end{bmatrix}.$$

Conjugating again,

$$g = (f^\star + \beta^{-1}\mathsf{q})^\star := \begin{bmatrix} x_g \\ s_g \\ y_g \end{bmatrix} = \begin{bmatrix} s_2 \\ x_2 \\ s_2\,.{*}x_2 - y_2 \end{bmatrix}.$$

Expanding y_g yields $y_g = (\beta/2)s_f\,.\hat{}2$. □

To compute the sum of two convex PLQ functions f and g finite at more than a single point, the x-axis must be repartitioned to include all of the distinct elements in x_f and x_g. Pseudocode is presented in Algorithm 3. The algorithm requires complexity in time and space linear in the number of pieces of f and g, as Line 3 merges two sorted sequences, after which each partition is only considered once, in $O(1)$ time. Note that additional (linear) work may be required before entering the for loop to determine in which pieces of f and g the entries of x fall.

The proximal average also requires more work to implement than the unary operators we have shown.

Proposition 4.3.3. *Let $f_1 = [x_1, s_1, y_1]$ and $f_2 = [x_2, s_2, y_2]$ be convex PLQ functions in GPH matrix format, and $\lambda \in [0,1]$. Define the operator P by $P = (\mathrm{Id} + \partial f_2)^{-1}(\mathrm{Id} + \partial f_1)$. Then the proximal average $P_\lambda(f_1, f_2) = [x, s, y]$ is given by:*

$$\begin{aligned} x &= (1-\lambda)x_1 + \lambda P x_1 \\ s &= x_1 + s_1 - x = x_2 + s_2 - x \\ y &= (1-\lambda)(y_1 + \frac{1}{2}x_1\,.\hat{}2) + \lambda(f_2(Px_1) + \frac{1}{2}(Px_1)\,.\hat{}2) - \frac{1}{2}x\,.\hat{}2 \end{aligned}$$

Algorithm 3: gph_add

```
Input: $f, g$ (convex PLQ functions, in GPH matrix format)
Output: $h$ (a convex PLQ function, in GPH matrix format)
begin
    $h \leftarrow []$;
    $x \leftarrow$ the ascending distinct partition points in $x_f$ and $x_g$;
    for each interval $[x_i, x_{i+1}]$ produced by the partition $x$ do
        Append to $h$ a column for the right slope at $x_i$ by adding the corresponding right slopes and function values of $f$ and $g$ at $x_i$;
        Append to $h$ another column for the left slopes and function values at $x_{i+1}$;
    end
    if $f(x) = \infty$ or $g(x) = \infty$ for $x < x_1$ then
        Bound $h$ on the left with a column $[x_1, s_1 - 1, \infty]^T$;
    end
    if $f(x) = \infty$ or $g(x) = \infty$ for $x < x_n$ then
        Bound $h$ on the right with a column $[x_n, s_n + 1, \infty]^T$;
    end
    $h \leftarrow$ UniqueColumns($h$);
end
```

Proof. In order to calculate the proximal average at x, we must find the x_1, x_2 that make the proximal average exact. To do this, we parametrize the graph with respect to x_1 and find the corresponding x_2. The formula for s was stated in Theorem 4.2.2. From s, we can see that $x_1 + s_1 = x_2 + s_2$, so $(\operatorname{Id} + \partial f_1)(x_1) = (\operatorname{Id} + \partial f_2)(x_2)$. Thus $x_2 = Px_1$ gives the points that correspond to x_1, i.e.

$$x_2 = (\operatorname{Id} + \partial f_2)^{-1}(\operatorname{Id} + \partial f_1)(x_1).$$

By using a reformulation of the proximal average [1, Proposition 4.3],

$$P_\lambda(f_1, f_2) = \inf_{x=(1-\lambda)x_1+\lambda x_2} \left((1-\lambda)(f_1 + \mathsf{q})(x_1) + \lambda(f_2 + \mathsf{q})(x_2) - \mathsf{q}(x)\right),$$

and substituting x_2, we arrive at the formula for y. □

It is important to note that $(y_1 + \frac{1}{2}x_1\texttt{.\^{}2})$, $(f_2(Px_1) + \frac{1}{2}(Px_1)\texttt{.\^{}2})$, and $(\frac{1}{2}x\texttt{.\^{}2})$ are independent of λ. These values can be precomputed to reduce the time needed to compute the proximal average for various λ and fixed f_1, f_2, x. The proximal average PLQ algorithm does not use such a precomputation scheme. This technique was demonstrated in [11] to compute $P_\lambda(\frac{1}{2}\|\cdot\|^2, \iota_{\{0\}} + 1)$ for many λ (Figure 4.1), and significant performance increases were noticed. The PLQ algorithm required 117 seconds, while the GPH algorithm for multiple λ took only 33 seconds.

4.4 Comparison of Algorithms

To compare the performance of GPH algorithms with existing algorithms, we implemented many of these GPH algorithms alongside their PLQ counterparts in [18]. The results may be seen in Figures 4.2 and 4.3.

We included four algorithms in our experiments. PLQ is the existing algorithm that works with PLQ functions directly. GPH is our new algorithm that works with GPH matrices. In Figure 4.2, LLT is the Linear-time Legendre Transform [16]. Finally, OPT is an algorithm using the n1fc1 nonsmooth bundle method solver built into Scilab 5.1.1 [6] with a warm-start from the previously computed point.

First, we look at computing the conjugate, $(x^4)^\star$, in Figure 4.2. The general OPT algorithm does not perform as well as specialized algorithms (Figure 4.2(a)), and removing OPT from the picture makes it clear that, although both algorithms clearly require linear time, GPH is outperforming PLQ significantly (Figure 4.2(b)). The PLQ conjugate algorithm makes use

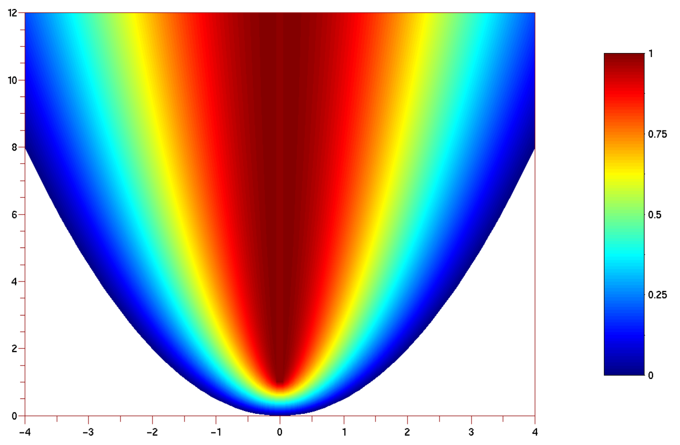
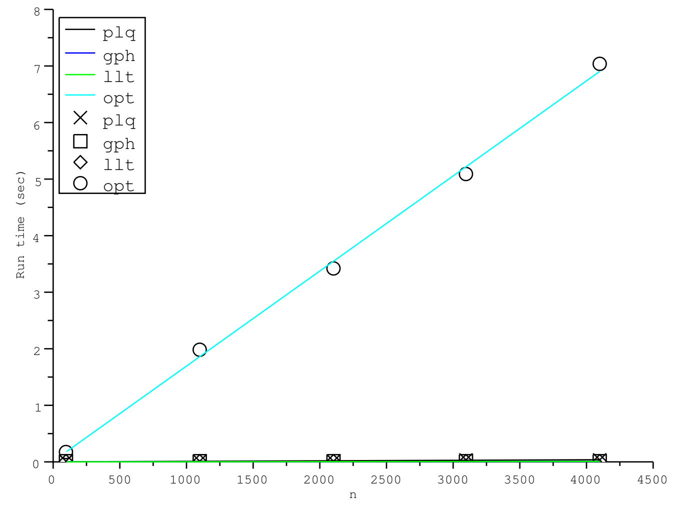


Figure 4.1: The proximal average of $f_1(x) = \frac{1}{2}\|\cdot\|^2$ and $f_2(x) = \iota_{\{0\}}(x) + 1$ as λ ranges from 0 (blue) to 1 (red).

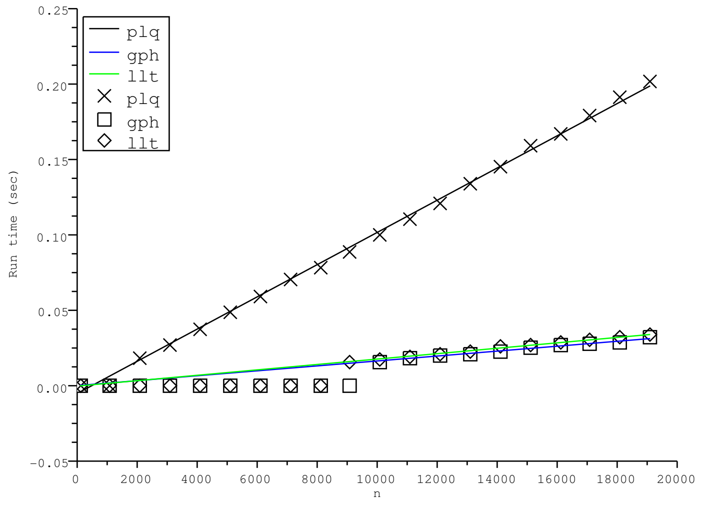
of a “clean” function to normalize its matrix, whereas this is not necessary for the GPH matrix. The GPH algorithm, which produces an exact transformed model, shows comparable performance to the inexact LLT algorithm.

We also used the proximal average algorithm for comparison (Figure 4.3). The results were much the same: OPT was the slowest algorithm; PLQ performed much better; and GPH showed significant improvement over both algorithms.

All experiments were performed on the same computer as in Section 3.3.



(a) All conjugate algorithms.



(b) With OPT removed.

Figure 4.2: Comparison of conjugate algorithms computing $(x^4)^\star$ approximated by n-piece piecewise linear functions on $[-10, 10]$.

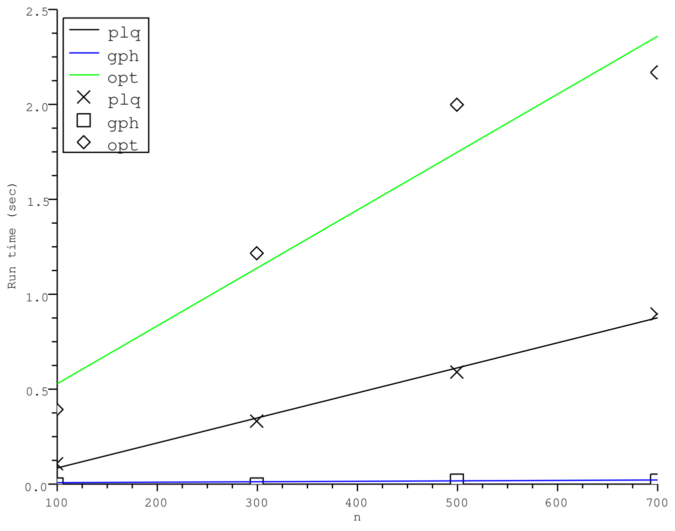


Figure 4.3: Comparison of proximal average algorithms computing $P_{\frac{1}{2}}(x^4, \mathrm{e}^x)$ approximated by n-piece piecewise linear functions on $[-10, 10]$.

Chapter 5

Conclusion

We have provided two means to compute the convex hulls of continuous piecewise linear-quadratic functions. The first algorithm requires treating a number of special cases at each iteration, but runs in optimal linear time. The second requires worst-case quadratic time and runs slower than the first even when it achieves its best-case linear time, but requires less effort to implement.

We also showed that by storing the graph of the subdifferential of a PLQ function in matrix form, it is possible to build a class of algorithms that computes many convex transforms of convex PLQ functions via matrix multiplication, followed by a linear-time calculation of the value of the transformed function. We included complete algorithms for scalar multiplication, addition of the scaled energy function, addition of two functions, the Legendre-Fenchel transform, the Moreau envelope, and the proximal average. Because these calculations are matrix and elementwise vector operations, and because no subalgorithms are necessary, we experienced significant performance gains over algorithms working with PLQ functions directly.

For both sets of algorithms, we experimentally validated our results with implementations in the Computational Convex Analysis toolbox [18].

Directions for future work include extending the univariate PLQ framework to model bivariate PLQ functions. Representation of the domain of a function becomes critical, and it is necessary to find a representation that aids the computation of the conjugate, from which other operators may be built. Computing the convex hull of a bivariate PLQ function is also important, however certain techniques can be ruled out: the duality approach of Section 3.2 cannot be directly applied, as the pointwise maximum of two convex bivariate PLQ functions, while convex, may fail to be PLQ (see Figure 5.1).

Another direction to take would be to extend the Graph-Matrix Calculus algorithms to work with nonconvex functions. Additionally, instead of relying on convex hull algorithms to be able to use convex algorithms on nonconvex functions, individual algorithms may be extended to handle

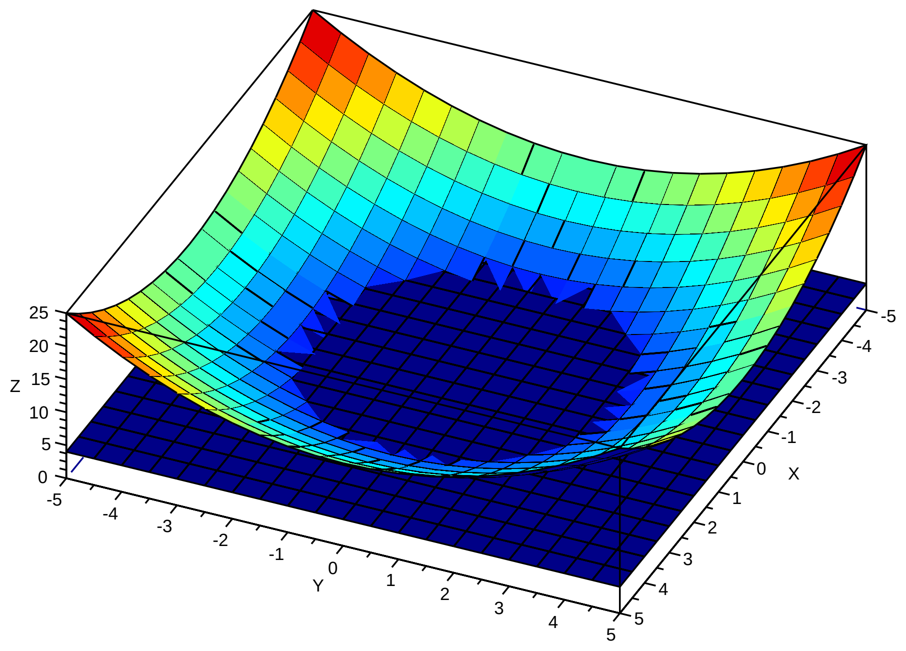


Figure 5.1: $f(x,y) = \frac{1}{2}(x^2 + y^2)$. $g(x,y) = 4$. $\max(f,g)$ is not a PLQ function, as one piece of its domain is a circle about $(0,0)$, which is not polyhedral as the PLQ class requires.

nonconvex cases as well.

Bibliography

[1] H. H. Bauschke, R. Goebel, Y. Lucet, and X. Wang. The proximal average: Basic theory. *SIAM Journal of Optimization*, 19(2):766–785, July 2008.

[2] H. H. Bauschke, Y. Lucet, and M. Trienis. The piecewise linear-quadratic model for computational convex analysis. *Computational Optimization and Applications*, 43(1), May 2009.

[3] H. H. Bauschke, E. Matoukov, and S. Reich. Projection and proximal point methods: convergence results and counterexamples. *Nonlinear Analysis: Theory, methods, and Applications*, 56(5):715–738, 2004.

[4] K. P. Bennett and E. Parrado-Hernández. The interplay of optimization and machine learning research. *J. Mach. Learn. Res.*, 7:1265–1281, 2006.

[5] P. L. Combettes and J.-C. Pesquet. A proximal decomposition method for solving convex variational inverse problems. *Inverse Problems*, 24(6), 2008.

[6] Scilab Consortium. Scilab. `http://www.scilab.org/`, 2010.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[8] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical Report TR2004-1963, Cornell Computing and Information Science, September 2004.

[9] B. Gardiner and Y. Lucet. Numerical computation of Fitzpatrick functions. *Journal of Convex Analysis*, 16(4), January 2009.

[10] B. Gardiner and Y. Lucet. Convex hull algorithms for piecewise linear-quadratic functions in computational convex analysis. Submitted 2010.

[11] B. Gardiner and Y. Lucet. Graph-matrix calculus for computational convex analysis. Submitted 2010.

[12] R. Goebel. Self-dual smoothing of convex and saddle functions. *Journal of Convex Analysis*, 15:179–190, 2008.

[13] W. L. Hare. A proximal average for nonconvex functions: A proximal stability perspective. *SIAM Journal on Optimization*, 20(2):650–666, May 2009.

[14] J.-B. Hiriart-Urruty and C. Lemarchal. *Convex Analysis and Minimization Algorithms*, volume 305–306 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, 1993.

[15] V. Koch, J. Johnstone, and Y. Lucet. Convexity of the proximal average. Submitted June 29, 2009.

[16] Y. Lucet. Faster than the fast Legendre transform, the linear-time Legendre transform. *Numerical Algorithms*, 16(2):171–185, 1997.

[17] Y. Lucet. Fast Moreau envelope computation I: Numerical algorithms. *Numerical Algorithms*, 43(3):235–249, November 2006.

[18] Y. Lucet. Computational Convex Analysis. `http://people.ok.ubc.ca/ylucet/cca`, 2010.

[19] S. M. Moffat. On the kernel average for n functions. MSc thesis, University of British Columbia Okanagan, 2009.

[20] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New York, 1970.

[21] R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis.* Springer-Verlag, Berlin, 1988.

[22] M. Trienis. Computational convex analysis: From continuous deformation to finite convex integration. MSc thesis, University of British Columbia Okanagan, January 2007.

[23] J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-newton approach to nonsmooth convex optimization problems in machine learning. *J. Mach. Learn. Res.*, 11:1145–1200, 2010.