Computing the Subdifferential of Convex Piecewise Linear-Cubic Functions

by

Aaron Matthew Mahnic

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

B.SC. COMPUTER SCIENCE HONOURS

in

THE COLLEGE OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2021

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

Computing the Subdifferential of Convex Piecewise Linear-Cubic Functions

submitted by Aaron Matthew Mahnic in partial fulfilment of the requirements of the degree of B.Sc. Computer Science Honours

Dr. Yves Lucet, I. K. Barber School of Arts & Sciences
Supervisor

Abstract

Computational Convex Analysis (CCA) studies the computation of convex operators commonly used in convex analysis. CCA allows researchers to visualize non-trivial convex transforms and build an intuition from these examples. Using the convexity and structure of piecewise linear cubic (PLC) functions as a model, the foundation for a new CCA toolbox has been implemented. The CCA toolbox is built in MATLAB and includes extensive test coverage and examples. In addition, it is free and open source.

The CCA toolbox provides several plotting methods to visualize PLC functions and their properties. Plotting the domain of a PLC function is a useful tool for both research and visualizing the correctness of computed output. Within a PLC function, any polyhedral set can be unbounded and plotting the domain of such a function would lead to an unbounded plot which cannot be displayed. We introduce a method that manipulates the PLCVC data structure in the CCA toolbox to plot an unbounded domain within a specified window.

A convex analysis operation of considerable interest is the Legendre-Fenchel transform, also known as the Fenchel conjugate. The computation of the Fenchel conjugate can be done in linear time. As a first step in implementing a linear time conjugate computation for the CCA toolbox, we introduce a new class (PLCVP) that implements a method for computing the subdifferential at any point of a PLC function.

Table of Contents

List of Tables

List of Figures

Acknowledgements

Thank you to Dr. Yves Lucet for the opportunity to work and learn under his guidance. This has been an invaluable experience in more ways than I can put into words.

Dedication

To my fiancé Nicole and my daughter Sophia.

Chapter 1

Introduction

Computational Convex Analysis (CCA) has proven to be a relevant topic and has found applications in areas such as computer vision, medical imaging and signal/image recovery [Luc10]. There are a number of fundamental transforms that arise in convex analysis that are integral to these applications. Robot navigation uses distance transform algorithms that are a special case of computing the Moreau envelope (also called the Moreau-Yosida approximate, Yosida Approximate or Moreau-Yosida regularization) [Luc10]. Medical imaging applies the Legendre-Fenchel conjugate transform to detect singularities in CT scans [Luc10, Qu11]. Lastly, signal and image recovery makes use of a combination of the former two transforms, called the proximal average, to recover the original signal or image from a corrupted version [CW05].

Computer-aided convex analysis focuses on visualizing fundamental transforms of convex analysis when applied to functions of typically one or two variables [Haq17]. A CCA toolbox was developed to perform these transforms on both univariate and bivariate functions. The latest version of the CCA toolbox is built in MATLAB [MAT21]. The CCA toolbox includes extensive test coverage and examples and is free and open source.

The new CCA toolbox represents bivariate piecewise linear-cubic (PLC) functions in multiple data types. The PLCVC class represents these PLC functions using a vertex and coefficient representation and is based on the structure PLC functions described in [SL21]. The PLCVP class uses a pointwise representation of PLC functions to use graph-matrix calculus (GMC) algorithms. Using GMC algorithms we can compute the Legendre-Fenchel conjugate and other convex analysis transforms in linear time [Haq17, GL11].

In this thesis, we characterize a data structure using vertex and coefficient representation of the PLCVC class and use this data structure to plot the domains of unbounded PLC functions in Chapter 3. In Chapter 4, we summarize a pointwise representation of PLC functions and show how the pointwise representation can be used to compute the Fenchel conjugate. We show subgradient calculus rules for PLC functions and describe how these rules are applied in a new algorithm to compute the subdifferential at any point of a PLC function. Finally, in Chapter 5 we show several examples of subdifferential computations for PLC functions.

Chapter 2

Preliminaries

This chapter reviews several key definitions that will be used throughout this paper.

Definition 2.1. [Bec17] (Convex Set) A set $C \in \mathbb{R}^n$ is convex if $\forall x, y \in C$ and $\lambda \in [0,1]$ it holds that:

$$\lambda x + (1-\lambda)y \in C.$$

Definition 2.2. [RW98] (Convex Hull) For a set $C \subset \mathbb{R}^n$, the convex hull of C consists of all the convex combinations of elements of C:

$$coC = \{\sum_{i=1}^{p} \lambda_i x_i \mid x_i \in C, \lambda_i \geq 0, \sum_{i=1}^{p} \lambda_i = 1, p \geq 0\}.$$

Fact 2.3. *The convex hull of a non-empty set C is the smallest convex set that contains C.*

Definition 2.4. [Bec17] (Normal Cone) For a set $C \subset \mathbb{R}^n$ and a point $x \in C$, the normal cone of C at x is defined as

$$N_C(x) = \{s \in \mathbb{R}^n \mid \langle s, z-x \rangle \leq 0, \forall z \in C\}.$$

Definition 2.5. (Proper Function) A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty, +\infty\}$, is said to be a proper function if $\text{dom}(f) = \{x \in \mathbb{R}^n : f(x) < +\infty\} \neq \emptyset$ and $f(x) > -\infty$ for all $x \in \mathbb{R}^n$.

Definition 2.6. (Convex function) A proper function $f : \mathbb{R}^n \to \mathbb{R}^n \cup \{+\infty\}$ is convex if its domain is a convex set and for any two points, x_1,$x_2 \in \text{dom}f$ and $\theta \in [0,1]$

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

Definition 2.7. (Subgradient) For a proper function $f : \mathbb{R}^n \to (-\infty, \infty]$ with $x \in \text{dom}(f)$, a vector $g \in \mathbb{R}^n$ is called a subgradient of f at x if

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

Definition 2.8. (Subdifferential) The set of all subgradients at f at x is called the subdifferential of f at x and denoted by

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

Definition 2.9. [RW98, Theorem 9.61] (Clarke Subdifferential) Let $f : \mathbb{R}^n \to (-\infty, \infty]$ be a proper convex function. The Clarke subdifferential at $x \in \mathbb{R}^n$ is given by

$$\bar{\partial} f(x) = co\{\lim_{i\to\infty} \nabla f(x_i) : x_i \to x \text{ and } \nabla f(x_i) \text{ exists}\}.$$

In addition, when f is convex then $\bar{\partial} f(x) = \partial f(x)$.

Fact 2.10. *[Bec17, Theorem 3.33] (The Subdifferential at Points of Differentiability) Let $f : \mathbb{R}^n \to (\infty, \infty]$ be a proper convex function, and let $x \in \operatorname{int}(\operatorname{dom}(f))$. If f is differentiable at x, then $\partial f(x) = \{\nabla f(x)\}$.*

Fact 2.11. *[Bec17, Theorem 3.36] (Subdifferential of the Sum) Let $f_1, f_2 : \mathbb{R}^n \to (\infty, \infty]$ be proper convex functions, and let $x \in \operatorname{dom}(f_1) \cap \operatorname{dom}(f_2)$.*
(a) The following inclusion holds,

$$\partial f_1(x) + \partial f_2(x) \subseteq \partial(f_1 + f_2)(x).$$

(b) if $x \in \operatorname{int}(\operatorname{dom}(f_1)) \cap \operatorname{int}(\operatorname{dom}(f_2))$, then

$$\partial(f_1(x) + f_2(x)) = \partial f_1(x) + \partial f_2(x).$$

Definition 2.12. (Indicator Function) For a subset $C \subset \mathbb{R}^n$, the indicator function of C is given by

$$\delta_C(x) = \begin{cases} 0 & x \in C, \\ \infty & x \notin C. \end{cases}$$

Fact 2.13. *(Subdifferential of the Indicator Function) Suppose that $C \subseteq \mathbb{R}^n$ with the indicator function δ_C. The subdifferential of δ_c is given by*

$$\partial \delta_C(x) = N_C(x), \forall x \in C.$$

That is, the subdifferential of the indicator function at a point x is the normal cone $N_C(x)$ of C.

Definition 2.14. [RW98] (Fenchel Conjugate) For any function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, the conjugate function $f^* : \mathbb{R}^n \to \mathbb{R} \cup [-\infty, +\infty]$ is given by

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

2.1 Piecewise Linear-Cubic Functions

Definition 2.15. [RW98] (Polyhedral Set) A set $C \subset \mathbb{R}^n$ is said to be a polyhedral set if it can be expressed as the intersection of a finite number of closed half-spaces or hyperplanes. The polyhedral set can equivalently be expressed as a finite number of linear constraints.

Definition 2.16. (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 $\mathcal{D} \subset \mathbb{R}^n$ 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) $\mathrm{ri}C_{k1} \cap \mathrm{ri}C_{k2} = \emptyset$, where $k_1, k_2 \in \mathcal{K}, k_1 \neq k_2$.

Definition 2.17. (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 subset of both.

Definition 2.18. (Piecewise Linear-Cubic Function) A Piecewise Linear Cubic function is a special case of piecewise polynomial functions with a polyhedral domain such that on each polyhedral set the function is defined by either a linear, quadratic or cubic function.

Similarly, a Piecewise Linear Quadratic (PLQ) function would be defined as having either a linear or quadratic function on each polyhedral set.

Definition 2.19. [Jak13, GJL14] (Edge) For any $x_1, x_2 \in \mathbb{R}^n$ and $x_1 \neq x_2$, we define an edge by the set

$$\mathcal{E} = \{x \in \mathbb{R}^n : x = \lambda_1 x_1 + \lambda_2 x_2, \lambda_1 + \lambda_2 = 1\}.$$

We classify edges as:

(i) Segment: When $\lambda_1, \lambda_2 \geq 0$.

(ii) Ray: When $\lambda_1 \geq 0$.

(iii) Line: When $\lambda_1, \lambda_2 \in \mathbb{R}$.

Definition 2.20. (Vertex) A vertex is either a starting point of a ray, one of the end points of a segment, or it is an isolated point.

Definition 2.21. (Face) A set F is a face of a polyhedral P if $\dim(F) = \dim(P) - 1$ and there exists a linear subspace H such that $F = H \cap P$.

Definition 2.22. [BJS11] (Direction, Extreme Direction) Given a convex set C, a nonzero vector d is a direction of C if for each $x \in C$ the ray $\{x + \lambda d : \lambda \geq 0\}$ also belongs to the set.

An extreme direction of a convex set is a direction of the set that cannot be represented as a positive combination of two distinct directions of the set.

Definition 2.23. [Roc70] (Extreme Point) Given a convex set C, a point $x \in C$ is an extreme point of C if x cannot be written as $\lambda y + (1 - y)z$ for $y, z \in C$,$z \neq x$, and $0 < \lambda < 1$.

Fact 2.24. *[Haq17] (Representation of a Polyhedral Set) Let C be a polyhedral set. There exists extreme points x_i and extreme directions d_j such that any $x \in C$ can be written,*

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

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

In addition, if C is bounded then the set of extreme directions is empty. If C is not bounded then the set of extreme directions is nonempty and has a finite number of elements.

Definition 2.25. (Planar Graph) A graph $G = (V, E)$ 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.

Fact 2.26. *[HL18, Proposition 2.13] Let $G = (V,E)$ be a connected planar graph with n_e edges and n_v vertices. Assume $n_v \geq 3$. Then $n_e \leq 3n_v$.*

Chapter 3

Plotting Unbounded Piecewise Polynomials

3.1 PLCVC

The PLCVC class within the CCA toolbox implements a data type for bivariate PLC functions. The class is able to represent both bounded and unbounded PLC functions.

3.1.1 Data Structure

To store a PLC function, each polyhedral set is divided into vertices, edges, and faces. Vertices indicate either the intersection of edges or a single point. Edges are either a line segment between two vertices or a ray which extends infinitely in one direction. If the interior of the polyhedral set is non-empty it is called a face.

As input, PLCVC takes in four specific matrices: $\mathcal{V}$, $\mathcal{E}$, f and $\mathcal{F}$. The matrix $\mathcal{V}$ contains the coordinates $(x, y) \in \mathcal{V}$ for each vertex. Edges are stored in the edge list $\mathcal{E}$ where each $e_i \in \mathcal{E}$ is a pair of indices (v_1, v_2) that index the two endpoints of the edge in the vertex list $\mathcal{V}$. If an edge is a ray, then a binary flag for each edge $e_i \in \mathcal{E}$ is set to specify that the edge extends infinitely past the vertex $v_2 \in e_i$. For a PLC function with n polyhedral sets, the coefficients for the polynomial function associated with each face are provided in an $n \times 10$ matrix f. The matrix F indexes the function indices from f for the faces to the left and right of each edge $e_i \in \mathcal{E}$.

In matrix form, the input is given by

$$\mathcal{V} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ . & . \\ . & . \\ x_n & y_n \end{bmatrix}, \mathcal{E} = \begin{bmatrix} e_{1,1} & e_{1,2} & b_1 \\ e_{2,1} & e_{2,2} & b_2 \\ . & . & . \\ . & . & . \\ e_{n,1} & e_{n,2} & b_n \end{bmatrix}, \mathcal{F} = \begin{bmatrix} i_{1,1} & i_{1,2} \\ i_{2,1} & i_{2,2} \\ . & . \\ . & . \\ i_{n,1} & i_{n,2} \end{bmatrix},$$

$$f = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & q_{1,1} & q_{1,2} & q_{1,3} & l_{1,1} & l_{1,2} & l_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & q_{2,1} & q_{2,2} & q_{2,3} & l_{2,1} & l_{2,2} & l_{2,3} \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ c_{n,1} & c_{n,2} & c_{n,3} & c_{n,4} & q_{n,1} & q_{n,2} & q_{n,3} & l_{n,1} & l_{n,2} & l_{n,3} \end{bmatrix}.$$

Note that not every input matrix needs to be defined with values, as long as the domain of the polyhedral partition the input represents is connected. For example, a cubic function with an infinite domain that has no vertices and no edges can be represented using an empty list for both $\mathcal{V}$ and $\mathcal{E}$.

Example 3.1. Consider the following example of a PLC function

$$f(x) = \begin{cases} -x^3 - y^3 + x^2 + y^2 & x \leq 0, y \leq 0; \\ -x^3 + 2y^3x^2 + 2y^2 & x \leq 0, y \geq 0; \\ 3x^3 + 2y^3 + 2x^2 + 2y^2 & x \geq 0, y \geq 0; \\ 3x^3 - y^3 + 2x^2 + y^2 & x \geq 0, y \leq 0. \end{cases}$$

The function f is continuous with a full domain. It has four distinct polyhedral faces. They can be represented using the $\mathcal{V}$, $\mathcal{E}$, f, $\mathcal{F}$ data structure as:

$$\mathcal{V} = \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \mathcal{E} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \\ 1 & 4 & 0 \\ 1 & 5 & 0 \end{bmatrix},$$

$$f = \begin{bmatrix} -1 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 2 & 1 & 0 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 & 2 & 0 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & -1 & 2 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \mathcal{F} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}.$$

If we take the first row from f as f_1

$$f_1 = \begin{bmatrix} -1 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

then we can see that,

$$f_1 * \begin{bmatrix} x^3 \\ x^2y \\ xy^2 \\ y^3 \\ x^2 \\ xy \\ y^2 \\ x \\ y \\ constant \end{bmatrix} = -x^3 - y^3 + x^2 + y^2,$$

which represents the first component in the PLC function f.

3.1.2 Computing Boundedness

The edge list $\mathcal{E}$ includes a binary flag for each $e_i \in \mathcal{E}$ that indicates whether the edge is a ray or a line segment as shown in Table 3.1. Upon instantiation this flag is used to determine if a PLC function is unbounded or not.

Flag	Edge type
1	line segment
0	ray

Table 3.1: Flag for edge

Proposition 3.2. *For a convex piecewise polynomial function f with a list of edges $\mathcal{E}$, if there exists at least one ray in $\mathcal{E}$, then* $\operatorname{dom}(f)$ *is unbounded.*

Proof. Let C be the closed convex set $\operatorname{dom}(f)$. If there is at least one ray in the edge list of f, then the set of extreme directions of C is nonempty. By Fact 2.24, if the set of extreme directions is nonempty, then C is unbounded. Thus, $\operatorname{dom}(f)$ is unbounded. $\square$

The converse of Proposition 3.2 is not necessarily true. There are two cases of PLC functions that will have no edges in the edge list $\mathcal{E}$ to consider,

(i) A function with a full domain.

(ii) A needle function (a function whose domain is a single point).

Distinguishing between these two cases is done by looking at the vertex matrix V. The function from (i) will have no vertex, while the needle function (ii) will have a single vertex.

To determine if a PLC function is bounded or not, PLCVC checks for the presence of rays by scanning the binary flags in $\mathcal{E}$ as defined in Table 3.1. If there are no edges in $\mathcal{E}$, then the total number of vertices in $\mathcal{V}$ is used to determine boundedness. As both the edge list $\mathcal{E}$ and vertex list $\mathcal{V}$ are provided to PLCVC at input as a planar graph. The algorithm runs through N edges and M vertices in $\mathcal{O}(N + M)$. For a planar graph, by Fact 2.26, we have that $M \leq 3N$ and therefore

$$\mathcal{O}(N + M) = \mathcal{O}(N + 3N) = \mathcal{O}(N).$$

Checking boundedness is performed in linear time where N is the number of vertices in a PLC function f.

3.2 Plotting

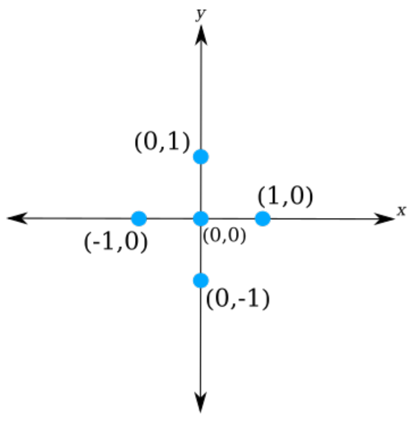
PLCVC includes a function to plot bounded domain PLC functions. We extended this function to plot functions with unbounded domains. When the domain is unbounded we replace the function f with $f + \delta_D$ where δ_D is the indicator function of a box D.

For any point in $\operatorname{dom}(f) \cap D$, the function will be equal to $f(x)$, while for any other point, it will be infinity. This gives us $\operatorname{dom}(f) \cap D$ which is the domain of f restricted to the box D. We specify D as a bounding box of the region of interest on $\operatorname{dom}(f)$ that we want to plot.

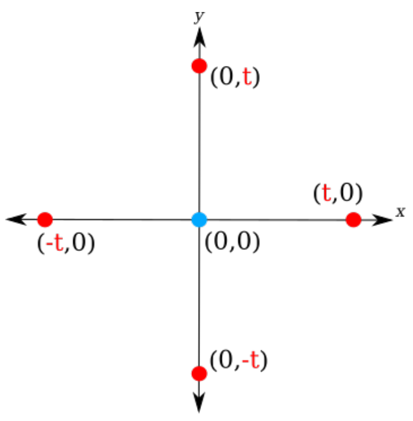
Example 3.3. Consider a PLC function with a vertex representation stored as

$$\mathcal{V} = \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \mathcal{E} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \\ 1 & 4 & 0 \\ 1 & 5 & 0 \end{bmatrix},$$

$$f = \begin{bmatrix} -1 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 2 & 1 & 0 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 & 2 & 0 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & -1 & 2 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \mathcal{F} = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}.$$



(a) Graph of vertices for a PLC function f.



(b) Vertices after non-extreme vertices have been extended by a factor of t.

Figure 3.1: Graph of vertices for a PLC function f before and after extending non-extreme vertices

In this example, every edge is a ray so the only extreme point is the vertex (0,0). After extending the non-extreme vertices by a magnitude $t = 1000$, the resulting matrix $\mathcal{V}_e$ will be

$$\mathcal{V}_e = \begin{bmatrix} 0 & 0 \\ -1000 & 0 \\ 0 & 1000 \\ 1000 & 0 \\ 0 & -1000 \end{bmatrix}.$$

A visual representation of the non-extreme vertex extension is illustrated in Figure 3.1(b).

The final step requires computing an intersection for each face. Let $\mathcal{B}$ be a matrix of points (x_i, y_i) that represent the size of the window to plot on such that

$$\mathcal{B} = \begin{bmatrix} x_{min} & y_{min} \\ x_{max} & y_{max} \end{bmatrix}.$$

We compute a bounding box as a polygon using the minimum and maximum x, y coordinates in $\mathcal{B}$. The bounding box polygon represents the window to plot the output to. Using the extended vertex list $\mathcal{V}_e$, a polygon p_i is constructed for each face. The intersection of each face polygon p_i with the

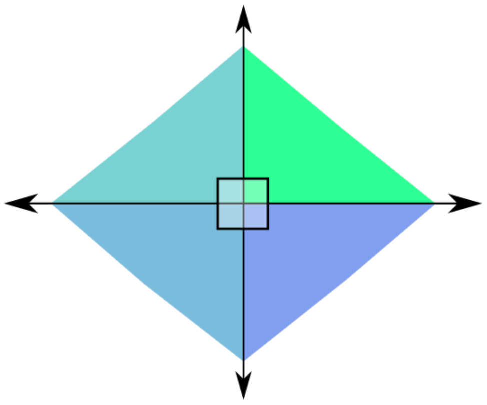


Figure 3.2: This figure shows how the plot unbounded domain algorithm works on a PLC function with four unbounded faces. Each coloured polygon on the plane represents a face. The black square represents the bounding box which is displayed as output.

bounding box computed from $\mathcal{B}$ represents the domain of each polyhedral set in a PLC function. A visualization of the bounding box overlaying the domain of a function that has been restricted to D is shown in Figure 3.2. Plotting each p_i together shows the entire domain of a PLC function within the specified bounding box $\mathcal{B}$.

Algorithm 1: Plot Unbounded Domain

```
input  : $P$ (PLCVC polyhedral set), xAxis (1x2 matrix specifying
         the boundary along the x axis to plot), yAxis (1x2 matrix
         specifying the boundary along the y axis to plot)
output: 2d plot of the resulting domain for $P$
begin
    if Domain is unbounded then
        $eVs \leftarrow$ extendNonExtremeVertices($P$);
        $facePolygons \leftarrow \emptyset$;
        boundaryView = createPolygon(xAxis,yAxis);
        foreach $face_i \in P$ do
            Compute the indices of each vertex $v \in P.P_i$ into $iX, iY$;
            $[X, Y] \leftarrow$ indices of each point $(x, y) \forall iX, iY \in eVs$;
            Compute the boundary $B$ of points $(x, y) \in [X, Y]$;
            boundaryFace = createPolygon($B$);
            facePolygons $\leftarrow boundaryView \cap boundaryFace$;
        end
        plot(each polygon $f \in facePolygons$);
    else
        plotBoundedDomain();
    end
end
```

Proposition 3.4. *Consider a PLC function with N vertices and M faces, the plotBoundedDomain function runs in linear time $\mathcal{O}(N)$*

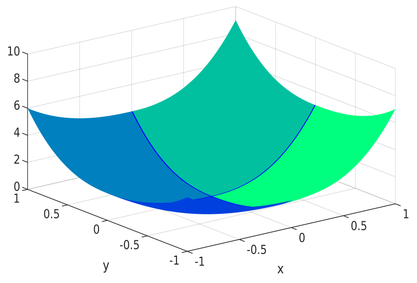
Proof. The computation to restrict the function f with $f_{|D}$ is done in $\mathcal{O}(N)$ by computing a new set of vertices V_D from the vertex matrix $\mathcal{V}$. Computing the intersection is done by building a polygon from the vertices in V_d and taking the intersection of this polygon with the bounding box which requires us to loop through the N vertices in V_D. Total runtime is dominated by the number of vertices and thus $\mathcal{O}(N)$.

Therefore, the algorithm runs in linear time. □

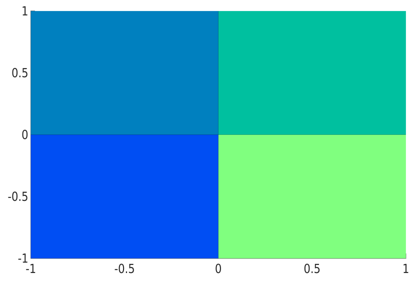
3.2.1 Example Plots

Cubic Function

Recall the cubic function from Example 3.1. It is comprised of four unbounded faces. Figure 3.3(a) shows a plot of the function in $\mathbb{R}^3$, while Figure 3.3(b) shows the plot of the domain for this function in $\mathbb{R}^2$. The four colours of each plot each represent a face. We see the output of the *plotUnboundedDomain* method correlates with the function plot.



(a) Plot of a cubic PLC function



(b) Plotted domain of the same function

Figure 3.3: PLCVC plot of the function and domain of a cubic convex PLC function. In both plots, the different colours represent different faces.

Quadratic Function

Consider the function,

$$f(x,y) = \begin{cases} x^2 + y^2 + 2xy & \text{x} \geq 0, y \geq 0; \\ \infty & \text{otherwise.} \end{cases}$$

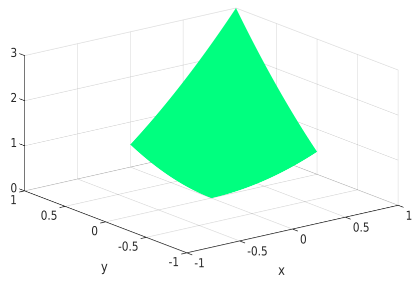
A plot of this function is shown in Figure 3.4(b). The domain of f is restricted to the first quadrant of the Cartesian plane. Using a bounding box $\mathcal{B} = [1,\ 1] \times [1,\ 1]$, we expect to see the domain plotted only where $x, y \geq 0$. The plot of $\text{dom}(f)$ is shown in Figure 3.4(b). In the figure, we see $\text{dom}(f)$ is correctly plotted on the first quadrant, while any $(x, y) \notin \text{dom}(f)$ are not plotted.

Changing Bounding Box

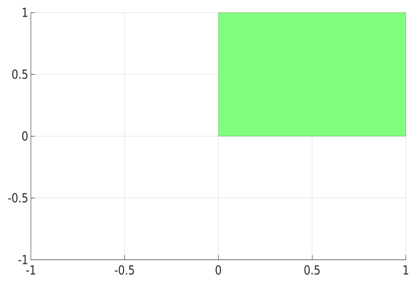
The bounding box that determines the region of interest to plot can be passed by the user to the *plotUnboundedDomain* method. In this example we show how changing the bounding box changes the output. Consider the function,

$$f(x,y) = \begin{cases} x^2 + y^2 & x \geq 0, y \geq 0; \\ -x^2 + y^2 & x < 0, y \geq 0; \\ \infty & \text{otherwise.} \end{cases}$$

The domain of this function is restricted to positive y values. The function has two faces separated by an edge along the y-axis. For each plot in 3.5 we denote the face on the left F_1 and the face on the right F_2. In Figure 3.5(a) the plot of dom(f) is shown centered on the origin. In Figure 3.5(a), we specify a new bounding box and shift the window along the x-axis by 0.5 units. The effect this has on the plot is that a larger area of the domain for the face F_1 is shown while the area of the domain for face F_2 is smaller as compared to Figure 3.5(a). In Figure 3.5(c) we shift the bounding box used in Figure 3.5(a) up by one unit along the y-axis. The function is defined everywhere above the x-axis and the plot shows a full domain in the window accordingly.

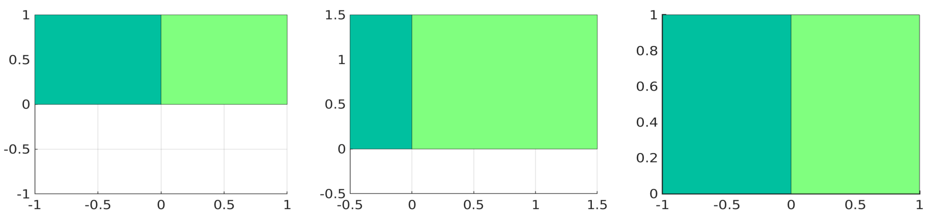


(a) Plot of a quadratic PLC function



(b) Plotted domain showing the function is restricted to the first quadrant

Figure 3.4: Plot of a quadratic function and the plot of the function's domain.



(a) Domain plot centered at the origin

(b) Domain plot shifted along the x-axis

(c) Domain plot above the x-axis

Figure 3.5: All three figures show the domain plot for the same PLC function. The figures illustrate the effect of changing the bounding box of the plot domain method.

Chapter 4

Computing the Subdifferential of PLC Functions

4.1 PLCVP

4.1.1 Motivation

With the CCA library, we want to perform convex transformations as efficiently as possible. The conjugate transformation can be computed in linear time by using the following relationship between the conjugate function and subgradients.

Theorem 4.1. *[Bec17, Theorem 4.20]* (Conjugate subgradient theorem). Let $f : \mathbb{R}^n \to (\infty, \infty]$ be a proper convex function. The following claims are equivalent for any $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$.

(i) $\langle x, y \rangle = f(x) + f^*(y)$.

(ii) $y \in \partial f(x)$.

If in addition f is closed, then (i) and (ii) are equivalent to

(iii) $x \in \partial f^*(y)$.

From Theorem 4.1 we note that if f is closed then,

$$y \in \partial f(x) \iff x \in \partial f^*(y). \tag{4.1}$$

Recall the graph of the subdifferential,

$$(x, y) \in \mathrm{gph}\partial f \iff y \in \partial f(x). \tag{4.2}$$

Applying Equations (4.1) and (4.2) then,

$$\begin{aligned}(x,y) \in \text{gph}\partial f &\iff y \in \partial f(x)\\ &\iff x \in \partial f^*(y)\\ &\iff (y,x) \in \text{gph}\partial f^*.\end{aligned} \tag{4.3}$$

By (4.3) we can see the domain of all subgradients in the subdifferential of a function is the same set as the range of all subgradients in the subdifferential of its conjugate function. This relationship leads to graph-matrix calculus [GL11] where numerous rules are given to transform the graph of a subdifferential of a function to the graph of a transformed function's subdifferential. Here, from Goebel's graph-matrix calculus, the Fenchel conjugate transform is given by

$$\text{gph}\partial f^* = \begin{bmatrix} 0 & id \\ id & 0 \end{bmatrix} \text{gph}\partial f. \tag{4.4}$$

For a proper closed convex function f with $x \in \mathbb{R}^n, y = f(x) \in \mathbb{R}^n$, and $s \in \partial f(x) \subset \mathbb{R}^n$ then $f^*(y)$ can be calculated by Fenchel's equality,

$$f^*(y) = \langle s, x \rangle - y. \tag{4.5}$$

As a consequence, we can compute complex transforms such as the conjugate with simple matrix multiplication which is both fast and easy to manipulate in MATLAB. PLCVP aims to represent PLC functions by storing the graph of the subdifferential to take advantage of graph-matrix calculus.

4.2 Subdifferentials of PLC Functions

When computing the subdifferential of convex PLC functions, there are three distinct cases to consider given where a point lies on the domain of the function. For a convex PLC function f, either a given point $f(x_1, x_2) = y$ lies outside of dom(f), somewhere on the interior of dom(f) or the point is on the boundary of dom(f). We apply subdifferential calculus based on these cases.

Outside of the Domain

The elements of $\partial f(x)$ are the subgradients of f at $x \in \mathbb{R}^2$ and f is subdifferentiable at x if $\partial f(x) \neq \emptyset$. By convention, when $x \notin$ dom(f), we define $\partial f(x) = \emptyset$. This definition is consistent with the subgradient inequality,

$$f(y) \geq f(x) + \langle s, y - x\rangle, \forall y \in \mathbb{R}^n. \tag{4.6}$$

For a proper function f, the subgradient inequality (4.6) does not hold if $x \notin \mathrm{dom}(f)$ and $y \in \mathrm{dom}(f)$. Thus, if $x \notin \mathrm{dom}(f)$, f is not subdifferentiable at x.

Interior of the Domain

We now consider a point on the interior of the domain. For a PLC function there are several cases.

(i) Interior of a face: The interior of each face is a polyhedral set defined by a polynomial expression which is continuous and differentiable anywhere on the interior. Let $x \in \mathbb{R}^2$, if a function f is convex and differentiable at $x \in \mathrm{dom}(f)$, then $\partial f(x) = \{\nabla f(x)\}$. For any point that lies on the interior of a face, we need only to compute the gradient of f at that point.

(ii) On an edge: If a point $x \in \mathbb{R}^2$ is on the relative interior of an edge, then it is on the intersection of two faces. For a PLC function f with faces F_1 and F_2 that share an edge e_i and have the respective associated functions f_1 and f_2 then $\{x \in \mathrm{ri}(e_i) : f_1(x) = f_2(x)\}$. From (i) we know that for $x \in \mathrm{int}(\mathrm{dom}(f))$ the subdifferential for each face is $\partial f_i(x) = \{\nabla f_i(x)\}$. Since f is convex we can apply the Clarke subdifferential from Definition 2.9 (see [RW98, Theorem 9.61]) and obtain the subdifferential at x by

$$\partial f(x) = \bar{\partial} f(x) = co\{\nabla f_i(x) : f_i(x) = f(x)\}. \tag{4.7}$$

(iii) On a vertex: For the case when a point is on a vertex but within the interior of a PLC function f, the formula (4.7) applies again. If a point on the interior of f that satisfies the constraints for n polyhedral sets, we compute the gradients at the point x for each of the n faces and then return the convex hull.

On the Boundary of the Domain

For each piece of a PLC function f, we associate the function $\tilde{f}_k = f_k + \delta_{C_k}$, where δ_{C_k} is the indicator function. By the subdifferential of the sum rule 2.11,

$$\partial \tilde{f}_k(x) = \partial[f_k(x) + \delta_{C_k}(x)] = \partial f_k(x) + \partial \delta_{C_k}(x). \tag{4.8}$$

The subdifferential of the indicator function ∂_{C_k} is the normal cone N_{C_k} from the Fact 2.13 so,

$$\partial f_k(x) + \partial \delta_{C_k}(x) = \nabla f_k(x) + N_{C_k}(x). \tag{4.9}$$

Applying (4.9) with the definition of the normal cone (2.4), then for any $x \in C_k$ and for all $z \in C_k$, we have the subdifferential at $\tilde{f}_k(x)$ is,

$$\begin{aligned}\partial \tilde{f}_k(x) &= \nabla f_k(x) + N_{C_k}(x)\\ &= \{s + \nabla f_k(x) \in \mathbb{R}^2 : \langle s, z - x\rangle \le 0\}.\end{aligned}$$

The subdifferential on an edge of a polytope P is the normal cone of x at P. The normal cone N_P will be a ray normal to the edge. For a vertex of P, each edge of the normal cone N_P will be a ray normal to the incident edges at v.

Proposition 4.2. *Let f be a proper PLC bivariate function having a finite number of faces $f_1, f_2, ..., f_n$. Assume v is a vertex of an edge on the boundary of* dom(f) *and that v is an extreme point. There is at least another edge having v as an extreme point such that $v = e_1 \cap e_2$ where E_1 and e_2 are edges on the boundary of* dom(f). *The normal cone at v is $N_{dom(f)}f(v) = N_{f_1}(v) \cap N_{f_2}(v)$ where f_i is the face of f with edge e_i.*

Proof. For v to be an extreme point, there must be two incident edges that are on the boundary of $dom(f)$. If there are more than two incident edges to v, only two of these edges can be on the boundary of dom(f) and any other edge is within the interior of dom(f). An edge that is within the interior of dom(f) then v is not an extreme point for that edge.

Assume s is a subgradient of f_1. If s is also a subgradient of f_2 then it is a subgradient of f. Therefore, any subgradient in the intersection of the subdifferentials of f_1 and f_2 is a subgradient of f. Conversely, any subgradient of f is also in the intersection of f_1 and f_2. □

4.2.1 Data Structure

As input, our subdifferential algorithm uses the same data structure that we use for PLCVC as described in Chapter 3. The input takes a vertex matrix $\mathcal{V}$, an edge matrix $\mathcal{E}$, a face adjacency matrix $\mathcal{F}$, and coefficient

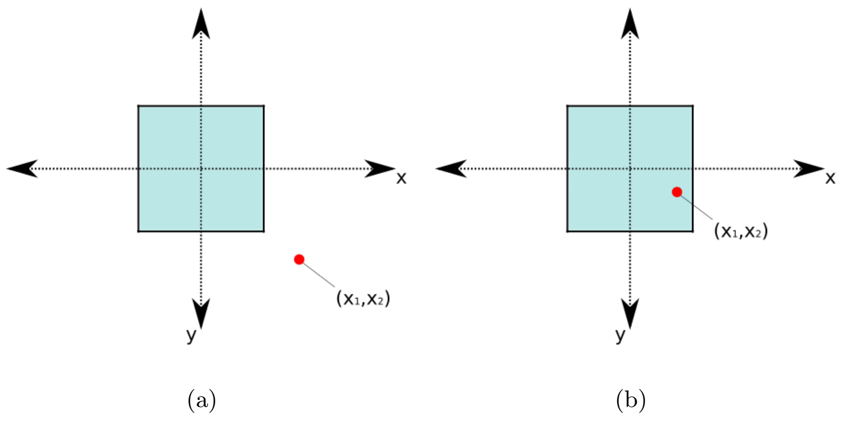


Figure 4.1: Figure (a) shows a point outside of the domain of f and (b) shows a point inside of the domain of f.

matrix f. Alternatively, a PLCVC object can be provided as input to the PLCVP class and these matrices will be passed along to the subdifferential algorithm. The subdifferential is always a closed convex polyhedral set so the algorithm outputs the subdifferential using the same data structure. To differentiate between the input and output, we will assign V_s, E_s, and F_s to be the subdifferential output matrices.

4.2.2 Algorithm

For a bivariate convex PLC function f, we can determine if a point $x \in \mathbb{R}^2$ lies outside of the dom(f) by evaluating the function at the point $f(x)$. The PLCVC class implements an evaluation method (*eval*) that we use for this purpose. If a point does not satisfy the constraints for any of the polyhedral sets in f then the value is infinity. Any finite value satisfies at least one constraint and therefore $x \in \text{dom}(f)$. In addition to returning the function value at a point, the PLCVC *eval* method returns the index(es) of the face(s) containing the point in a matrix $\mathcal{R}$ for any polytope which satisfies the constraints. If the PLCVC *eval* method returns infinity, then there is no subdifferential at the point and we return an empty set. Figure 4.1 compares a point outside of dom(f) to one on the interior of dom(f).

When we know that a point is on the interior of the domain of f, the next step is determine if the point is on an edge or a vertex so that we can apply the subdifferential calculus rules. For each face F_i in the region

$\mathcal{R}$ array, we check if the given point x lies somewhere on one of the edges $e_n \in F_i$. If we find the point is on e_n then either the point is on one of the two vertices $v_1, v_2 \in e_n$ or somewhere on the relative interior of e_n. If the point lies on any edge, we store an index to e_n in an edge list iE. One limitation to note is that we are not currently handling floating point errors when calculating if a point is on an edge.

After checking if the point lies on any edge of the faces, if the edge list iE is empty and the number of faces in $\mathcal{R}$ is exactly one, then we know the point is not any vertex or edge. We can conclude that such a point must be on the interior of a face. Using the face number i returned by the eval function, we compute the gradient of the point on the face and return the singleton $\{\nabla f_i(x_1, x_2)\}$ as the subdifferential at this point. If the edge list iE is empty but the region array $\mathcal{R}$ had more than one element, we return an error. This could be the case for a floating point error where a point was on an edge or a vertex but the calculation determining if the point was on an edge found it was not.

For any point with at least one element in the edge list iE, we need to know how many constraints in f the point satisfies and if this point is on the boundary of dom(f).

We perform one loop through the edge list iE to build a list of gradients and properties of each edge in iE. We build the grd cell array as,

$$grd = \{e_j, \nabla f_j(x_1, x_2), b_j\},$$

where $e_j \in iE$ is the edge index, $\nabla f_j(x_1, x_2)$ is the gradient for the face that e_j is on, and b_j is a binary flag indicating if the edge e_j is on a boundary. If an edge is between two faces (equivalently not on a boundary of dom(f)) then it is stored in grd twice so that the gradient of both adjacent faces of the edge e_j are stored. Based on the number of elements in grd, we have a number of possible cases as listed in Table 4.1.

Case No.	No. of gradients	b flag count	Case
1	2	0	On an interior edge
2	>2	0	On interior vertex
3	1	1	On a boundary edge
4	1	>1	On boundary vertex (1 face)
5	>1	>1	On boundary vertex (2+ faces)

Table 4.1: Cases for the grd array

Two possible cases are not included in Table 4.1 because they are handled

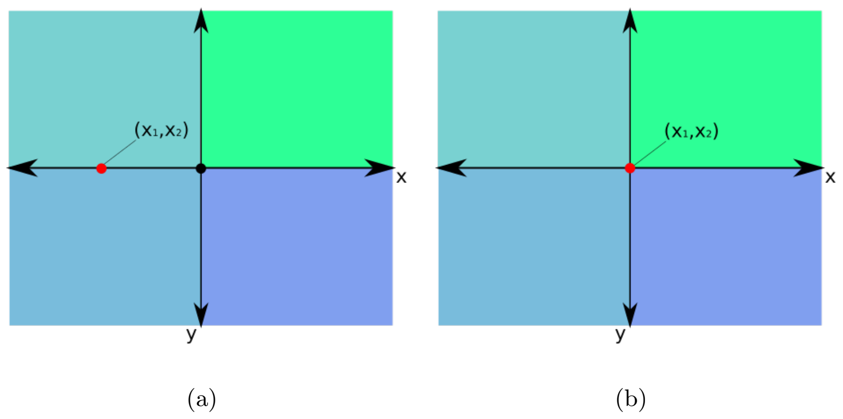


Figure 4.2: Both figures show a PLC function f with four faces and four edges. Figure (a) shows a point on an edge that is in the int(f). Figure (b) shows a point that is on a vertex and in int(f).

prior to the creation of grd. If there are no subgradients then the point x was outside of dom(f). If there was a single gradient that was not on the boundary, then the point was found to be on the interior of a face and the gradient has been returned. In either case, an error is returned if these cases are found after computing grd.

For cases 1 and 2 from Table 4.1, the computation of the subdifferential is similar. We take each gradient in grd and use them as vertices in the vertex list V_s. If there are only two gradients, as per case 1 in Table 4.1 where the point is on an interior edge, then the output will be a line segment between the two vertices. We add a single edge to E_s between the two vertices. We leave the face matrix F_s empty. If there are more than two elements then the face is on an interior vertex. In this case, we build the edge list as a connected graph such that, if V_s has n vertices, there are exactly n edges and each vertex is connected to exactly two edges. Vertices from adjacent faces are connected to each other.

For each of the cases 3,4,5 from Table 4.1 the point is on the boundary. In all of these cases, we compute the normal cone at the point. If the point is on an edge and on the boundary as in case 3 of Table 4.1 and shown in Figure 4.3(a), then we know the normal cone will be a ray that is normal to the edge. In this case, we compute the normal cone which builds a ray by adding two vertices to V_s, a single edge to E_s, and setting $F_s = \emptyset$. This represents the ray normal to the edge e_j.

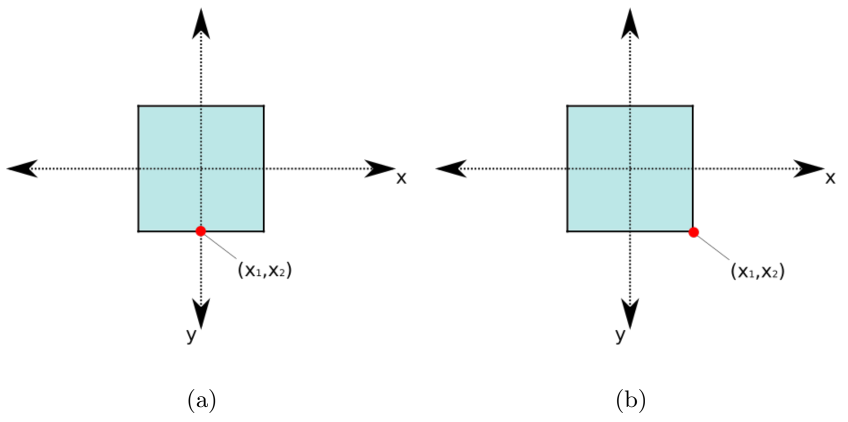


Figure 4.3: Figure (a) shows a point on the boundary of dom(f) on an edge (b) shows a point on the boundary of dom(f) on the vertex.

When the point is found to be on a boundary vertex of only one face, as in case 4 from Table 4.1, we compute the normal cone at the point and return the normal cone representation as V_s,E_s,F_s.

In the final case, we compute the intersection of two or more normal cones. To compute the intersection of two normal cones, we first compute the normal cone for each in the V_s,E_s,F_s representation. With the two normal cones, we note that they both share a vertex at the point we are evaluating. We compute the normalized 360 degree angle each edge makes at this point. We then evaluate where the edges lay in relation to each other to compute the overlap. Once the overlap is determined, a new V_s,E_s,F_s list is built using the edges found that overlapped. If the overlap is determined to only be along a ray, then only a ray is returned.

4.2.3 Algorithm

Algorithm 2: Subdiff

```
input : $x$ (1x2 matrix containing point to evaluate at), $P$ (PLCVC
        polyhedral set)
output: $z$ (1x2 matrix of the function evaluated at $x$), $G$
        (polyhedral set of the subdifferential at $x$)
begin
    $z, face \leftarrow PLCVC.eval(x)$;
    if (isInfinity(z)) then
        $G \leftarrow \{\}$; return;
    end
    $iE \leftarrow$ [Edges in $P$ that $x$ lies on];
    if (isEmpty(iE)) then
        $G \leftarrow \{\nabla f_{face}(x)\}$; return;
    end
    grd $\leftarrow [e_j, \nabla f_i(x), b_j]$ if edge $e_j \in iE$ on face $i$;
    if (all $b_j \in grd$ == 0) then
        $V_s, E_s, F_s \leftarrow$ build polyhedral set from $co(\nabla f_g(x) \forall g \in$grd);
    else if (any $b_j \in grd$ == 1) then
        $V_s, E_s, F_s \leftarrow$ computeNormalCone($x$);
    end
    $G \leftarrow \{V_s, E_s, F_s\}$; return;
end
```

Proposition 4.3. *Consider a PLC function with N faces, the Subdiff algorithm runs in $\mathcal{O}(N)$*

Proof. The algorithm runs through a number of cases, and if any of the cases is satisfied along the way, the subdifferential is returned and the algorithm exits. The worst case run time requires the algorithm to pass through each case until the final one. The worst case occurs when a point x is on a vertex on the boundary of dom(f). For every case, the PLCVC *eval* method evaluates a point on a PLC function with M faces in $\mathcal{O}(M)$ time. The evaluation needs to do a linear search of each of the M faces to find the right face to evaluate the function at. Once the face is found, the point is evaluated using the coefficients for that face which is an efficient calculation in $\mathcal{O}(1)$ time. For a point $x \notin$ dom(f) the subdifferential is an empty set and returned in $O(1)$ time.

We loop through the N edges in $\mathcal{E}$ once to build the edge list iE. In addition, we loop through the edges in iE to compute the gradient edge list *grd*. Both of these operations are computed in $O(N)$. If a point is found to not be on any edge then $\nabla f(x)$ is computed and returned as a singleton. Computing the gradient at x is done in $\mathcal{O}(1)$. Evaluating possible cases for *grd* is done in $\mathcal{O}(N)$ as there could be N edges in *grd*. The worst case runtime operation after we have checked the *grd* matrix is when the normal cone must be computed for two faces. The normal cone and subsequent intersection calculations are done in $\mathcal{O}(1)$.

Therefore, the overall time complexity for the algorithm is linear.

Chapter 5

Numerical Experiments

In this chapter we use the subdifferential method in PLCVP to find the subdifferential at various points of a few PLC functions. We use plotting methods from PLCVC to visualize the PLC functions and the subdifferential at these points.

5.0.1 Example 1

Consider the l_1 norm function,

$$f(x_1, x_2) = \begin{cases} x_1 + x_2 & -x_1 \le 0, -x_2 \le 0; \\ -x_1 + x_2 & x_1 \le 0, x_2 \ge 0; \\ -x_1 - x_2 & x_1 \ge 0, x_2 \ge 0; \\ x_1 - x_2 & x_1 \ge 0, x_2 \le 0. \end{cases}$$

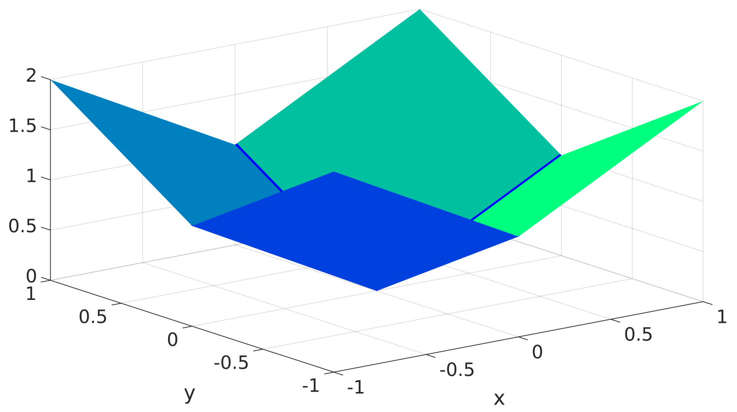
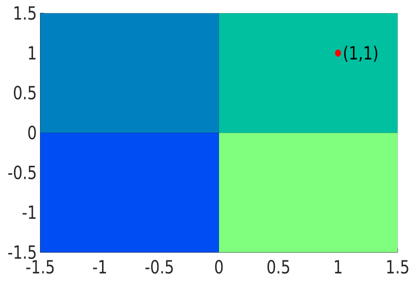


Figure 5.1: Plot of the l_1 norm function.

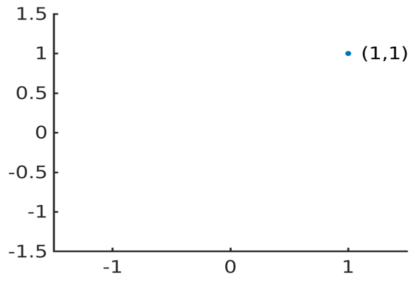
A plot of the l_1 norm function is shown in Figure 5.1. We evaluate the subdifferential at three different points. At $f(1,1)$, the point is on the interior of face three. The subdifferential is the gradient at the point $f_3(1,1)$ which we compute as,

$$\partial f(1,1) = \nabla f_3(1,1) = (\frac{\partial f}{\partial x_1}(1,1), \frac{\partial f}{\partial x_2}(1,1)) = (1,1).$$

Accordingly, the subDiff method returns the same subgradient as the subdifferential for the l_1 norm at (1,1). The output from the subdifferential algorithm at (1,1) on the l_1 norm is shown in Figure 5.2.



(a) Domain plot of the l_1 norm.



(b) ∂f at (1,1).

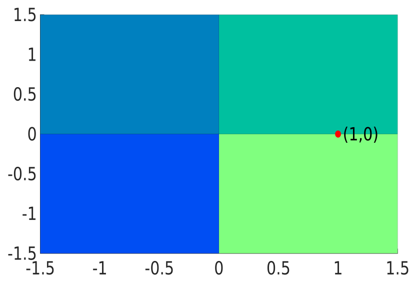
Figure 5.2: Figure (a) shows the domain of the l_1 norm with the point (1,1) highlighted. Figure (b) shows the subdifferential of the l_1 norm at (1,1).

Next, we compute the subdifferential of the l_1 norm at (1,0). This point is on an edge between two faces. The subdifferential is computed as

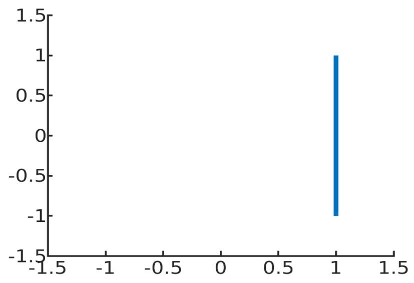
$$\begin{aligned}\partial f(1,0) &= co\{\nabla f_3(1,0), \nabla f_4(1,0)\}\\ &= co\{(1,1),(1,-1)\}\\ &= \{(x_1,x_2) \in \mathbb{R}^2 : x_1 = 1; -1 \leq x_2 \leq 1\}.\end{aligned}$$

The subdifferential algorithm computes the gradients of both faces then adds the gradients as vertices to the V_S matrix. An edge is added between the two vertices to represent the interval found above and the edge is flagged as a line segment. A plot of the output from the subdifferential algorithm for the l_1 norm at (1,0) is seen in Figure 5.3. The matrices returned by the algorithm for the subdifferential of the l_1 norm at this point are,

$$V_S = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, E_S = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}, F_S = \begin{bmatrix} 0 & 0 \end{bmatrix}.$$



(a) Domain plot of the l_1 norm.



(b) ∂f at (1,0)

Figure 5.3: Figure (a) shows the domain of the l_1 norm with the edge point (1,0) highlighted. Figure (b) shows the subdifferential of the l_1 norm at (1,0).

A final example using the l_1 norm is for the subdifferential at (0,0). This point is a vertex and all four faces are "active" at this point. The computation for the subdifferential requires the gradient of all four faces,

$$\begin{aligned}\partial f(0,0) &= co\{\nabla f_i(0,0) : f_i \in f\}\\ &= co\{[-1,1]\times[-1,1]\}\\ &= \{(x_1,x_2)\in\mathbb{R}^2 : -1\le x_1\le 1; -1\le x_2\le 1\}.\end{aligned}$$

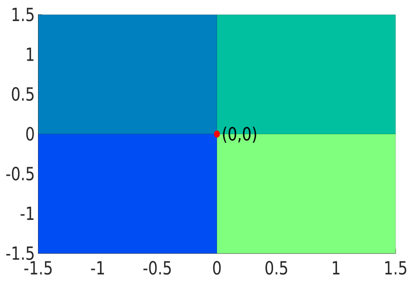
We find that the subdifferential algorithm correctly computes the square around the origin at (0,0). The algorithm finds four gradients and adds them as vertices. A list of edges is built forming a bounded square from the vertices. The matrices returned by the algorithm are,

$$V_S = \begin{bmatrix}-1 & -1\\ -1 & 1\\ 1 & 1\\ 1 & -1\end{bmatrix}, E_S = \begin{bmatrix}1 & 2 & 1\\ 2 & 3 & 1\\ 3 & 4 & 1\\ 4 & 1 & 1\end{bmatrix}, F_S = \begin{bmatrix}0 & 1\\ 0 & 1\\ 0 & 1\\ 0 & 1\end{bmatrix}.$$

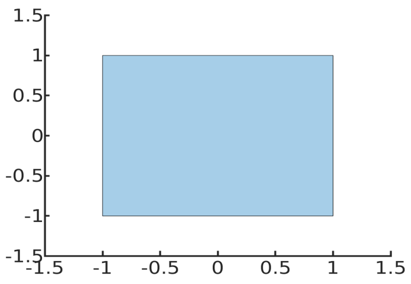
A plot of the subdifferential at (0,0) of the l_1 norm is seen in Figure 5.4

5.0.2 Example 2

To test how the algorithm computes the subdifferential on a boundary consider,



(a) Domain plot of the l_1 norm.



(b) ∂f at (0,0)

Figure 5.4: Figure (a) shows the domain of the l_1 norm with the vertex (0,0) highlighted. Figure (b) shows the subdifferential of the l_1 norm at (0,0).

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

A plot of this function is shown in 5.5.

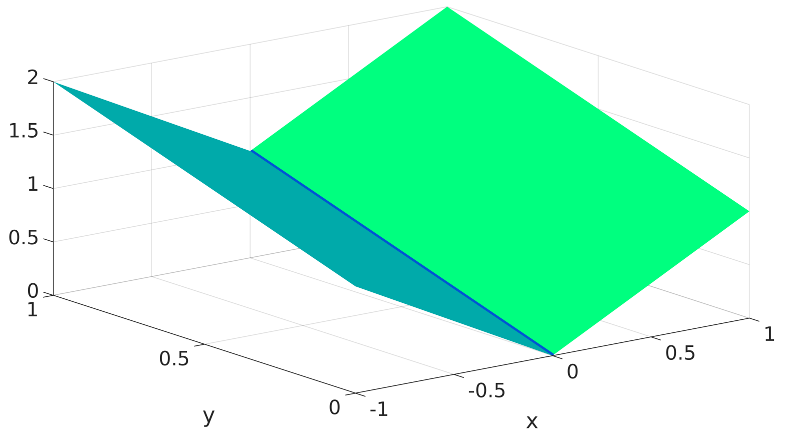
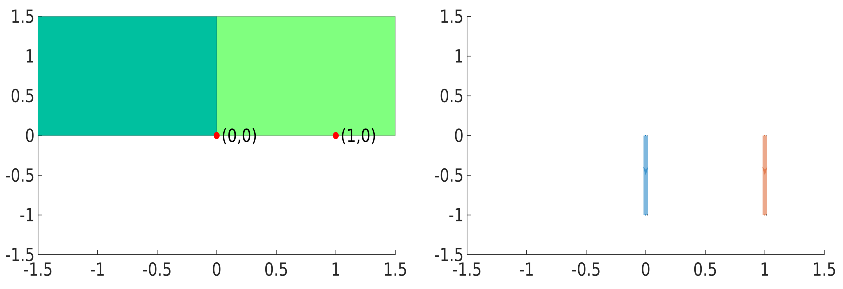


Figure 5.5: Plot of a PLC function with a boundary along the x axis

Since the boundary is a straight line along the x axis, the subdifferential at any point in f will be a ray that is normal to the x axis. We compute the subdifferential on an edge (1,0) and at a vertex (0,0) and find in both cases that a ray is properly returned, as shown in Figure 5.6.



(a) (b)

Figure 5.6: Subdifferentials on an edge

5.0.3 Example 3

Consider the bounded triangle,

$$f(x_1,x_2) = \begin{cases} x_1^2 + x_2^2 + 2x_1x_2 & \text{in triangle with vertices}(0,0),(1,0),(0.5,1); \\ \infty & \text{otherwise.} \end{cases}$$

A plot of this triangle is shown in Figure 5.7. For the subdifferential of the triangle at the vertex (0,0), we expect a normal cone comprised of two rays that are normal to the two edges that meet at the vertex. Indeed this is what we get, as Figure 5.8 shows the boundary of the triangle with the normal cone at (0,0) as returned by the subDiff algorithm. The following vertex representation of the subdifferential at (0,0) of the triangle is returned by the subdifferential algorithm,

$$V_S = \begin{bmatrix} 0 & 0 \\ 0 & -1 \\ -1 & 0.5 \end{bmatrix}, E_S = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \end{bmatrix}, F_S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Both edges are flagged as rays in the edge list E_S as noted by the zeros in column three.

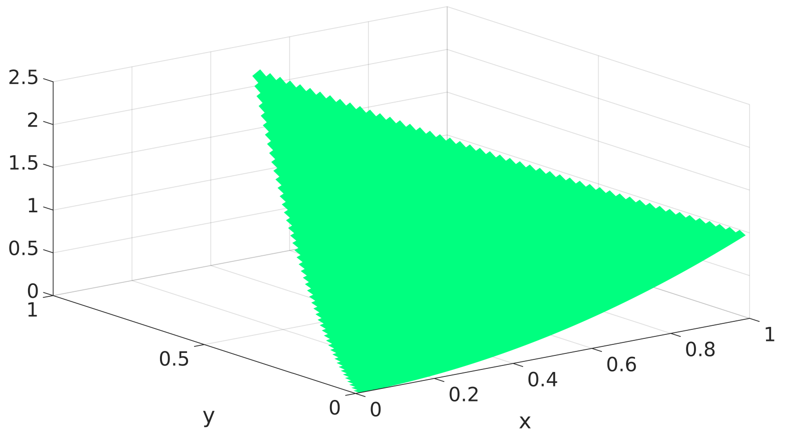


Figure 5.7: A plot of a bounded triangle with only one face.

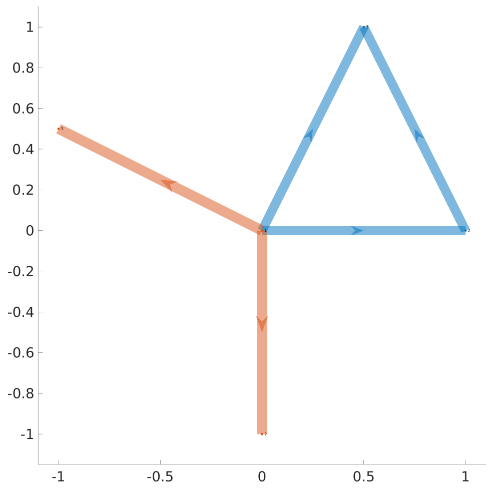


Figure 5.8: The edges in blue show the boundary of the triangle. The edges in orange show the subdifferential of the triangle at (0,0) which is a normal cone

Chapter 6

Conclusion and Future Work

Two contributions have been made to the CCA library. A plot domain algorithm was added to PLCVC, and a subdifferential computation algorithm was added as a basis for the PLCVP library. We showed how the vertex and coefficient representation of PLC functions used by the PLCVC class could be manipulated to work with a region of interest for functions with full domains. We used this restricted region of interest to plot the domain of unbounded PLC functions. This method may prove useful for future additions to the CCA library that work with PLC functions of full domains.

We introduced the subDiff algorithm to compute the subdifferential at any point of a PLC function. We showed how the algorithm uses subdifferential calculus and the structure of PLC functions to compute the subdifferential. This algorithm can be used to build a pointwise representation of PLC functions using GPH matrices.

Now that the subdifferential can be computed, the PLCVP class needs an algorithm to build GPH matrices to represent PLC functions. There is still research to be done on the optimal way to store the GPH matrices. In [Haq17] a GPH matrix was created for each entity of a PLC function and a hyper matrix was used to keep track of the entity types and arrangement. An alternative is to store the values in each GPH matrix as indices to larger matrices. For example, each x_i in a GPH matrix would be stored as an index to a matrix of all GPH matrix points X. By storing the matrix in this way, a single matrix operation can be done to transform the graph of the subdifferential as opposed to looping through each GPH matrix and transforming matrix.

In addition to computing transforms, PLCVP class will need to be able to convert back to PLCVC coefficient representations of functions. Since there is no unique way to store a function as a GPH matrix, we must ensure that we accurately recover the correct function.

Bibliography

[Bec17] Amir Beck. *First-Order Methods in Optimization.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017. → pages 3, 4, 17

[BJS11] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear programming and network flows: Fourth edition.* 01 2011. → pages 6

[CW05] Patrick L. Combettes and Valérie R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale modeling simulation*, 4(4):1168–1200, 2005. → pages 1

[GJL14] Bryan Gardiner, Khan Jakee, and Yves Lucet. Computing the partial conjugate of convex piecewise linear-quadratic bivariate functions. *Computational optimization and applications*, 58(1):249–272, 2014. → pages 5

[GL11] Bryan Gardiner and Yves Lucet. *Graph-Matrix Calculus for Computational Convex Analysis*, pages 243–259. Springer New York, New York, NY, 2011. → pages 1, 18

[Haq17] Tasnuva Haque. *Computation of convex conjugates in linear time using graph-matrix calculus.* PhD thesis, University of British Columbia, 2017. → pages 1, 6, 33

[HL18] Tasnuva Haque and Yves Lucet. A linear-time algorithm to compute the conjugate of convex piecewise linear-quadratic bivariate functions. *Computational Optimization and Applications*, 70(2):593–613, 06 2018. → pages 6

[Jak13] Khan M. K. Jakee. *Computational convex analysis using parametric quadratic programming.* PhD thesis, 2013. → pages 5

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

[MAT21] MATLAB. version 9.10.0 (r2021a), 2021. → pages 1

[Qu11] Gang-rong Qu. Singularities of the radon transform of a class of piecewise smooth functions in r 2. *Acta Mathematicae Applicatae Sinica*, 27(2):191–208, 2011. → pages 1

[Roc70] R. Tyrrell Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970. → pages 6

[RW98] R.T. Rockafellar and R.J.B. Wets. *Variational analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998. → pages 3, 4, 5, 19

[SL21] Shambhavi Singh and Yves Lucet. Linear-time convexity test for low-order piecewise polynomials. *SIAM Journal on Optimization*, 31(1):972–990, 2021. → pages 1