The Piecewise Linear-Quadratic Model for Convex Bivariate Functions

by

Scott Ronald Fazackerley

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

Bachelor of Science Honours

in

The I. K. Barber School of Arts & Sciences

(Computer Science)

The University Of British Columbia

(Kelowna, Canada)

April, 2008

Abstract

A Piecewise Linear-Quadratic (PLQ) function is used to describe data that can be represented by continuous functions with a piecewise linear domain for which the function is either linear or quadratic on, each piece of its domain [11]. While extensive work has been done with PLQ models for convex univariate functions, my work investigates the development of a two dimensional model that allows the implementation of algorithms for computing fundamental convex transforms. PLQ functions have been described in the literature, the efficient implementation of the algorithms requires the careful selection of a data structure.

Initial investigation examined using a Voronoi diagram to represent the projection of the bivariate function into $\mathbb{R}^2$, to take advantage of efficient algorithms for the point location problem [3]. While suitable for representing a single PLQ, with operations for building a zero order model from data and evaluating the function over an irregular grid, we are uncertain if it can be extended to compute other fundamental convex transforms efficiently.

In examining the Voronoi model, we were able to extend the base concept to represent the bivariate PLQ using two different representations: a tessellation-based model and a linear inequality-based model. The tessellation model is restricted to representing a PLQ with a bounded domain. The model represents data through triangular faces in a tessellation in $\mathbb{R}^2$ where each face is defined by its vertices and the associated function value. This model allows for the efficient evaluation over an irregular grid, the addition of two PLQ functions with bounded domains and multiplication by a scalar, but does not allow for the representation of an unbounded domain. To allow for an unbounded domain, a dual model was developed using linear inequalities to represent each face in $\mathbb{R}^2$. Numerical results are presented for each model and computational complexity of model components are discussed.

Acknowledgments

I would like to thank Dr. Yves Lucet for providing me with the opportunity to do an Honours under his supervision. His feedback and insights have been invaluable in bringing my thesis to its current form. This has been an exceptional learning experience in areas of computing and mathematics which I would not have otherwise had the opportunity to explore.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Extensive work has been completed in regards to a model for a univariate Piecewise Linear-Quadratic (PLQ) function. A univariate PLQ is used to describe data that can be represented by continuous functions with a piecewise linear domain for which the function is either linear or quadratic on each piece of its domain [11]. Univariate PLQ functions appear in the fields of economic modeling, control theory and scientific computing, specifically computational convex analysis [10, 11, 13]. While work has been done with PLQ models for convex univariate functions, a bivariate PLQ model has not yet been presented that supports efficient implementation of fundamental convex transforms. PLQ functions are closed under standard convex operations, and can be manipulated with linear-time algorithms [11]. Efficient manipulation is defined as having a storage and run-time complexity that is at worst polynomial.

My work investigates the development of a two dimensional model that allows the implementation of algorithms for computing fundamental convex transforms. While PLQ functions have been described in the literature, the efficient implementation of the algorithms requires the careful selection of a data structure. The requirements for the model are presented in Section 1.2. Three models are examined and discussed in Section 2.

The models presented are zero-order models: only point data can be accepted as input to the model. The algorithms presented in Section 3 and Section 4 compute function values over the domain of the input data. Convexity is enforced over the domain of input values: points making the function non-convex are removed such that the function remains continuous and convex over the domain. Additionally, algorithms for converting from a bounded to an unbounded model, evaluating, adding and (scalar) multiplying are discussed. Complexity results are also presented, proven and validated with empirical data.

1.1 Preliminaries

In this section, we recall some basic notions from computational geometry and convex analysis.

Definition 1.1.1. *Convex Set [6] A set C is convex, if the line segment between any two points in C lies in C. That is, for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have*

$$\theta x_1 + (1-\theta)x_2 \in C. \tag{1.1.1}$$

Definition 1.1.2. *Convex Hull [6] The convex hull of a set C, denoted **conv** C is the set of all convex combinations of points in C*

$$\textbf{\textit{conv}}\ C = \{\theta_1 x_1 + ... + \theta_k x_k | x_i \in C, \theta_i \geq 0, i = 1, ..., k, \theta_1 + ... + \theta_k = 1\}. \tag{1.1.2}$$

The convex hull ***conv*** *C is always convex.*

Definition 1.1.3. *Convex Function [6] A function $f : R^n \rightarrow R \cup \{+\infty\}$ is convex if* ***dom*** *$f = \{x | f(x) < \infty\}$ is a convex set and if for all x, $y \in$* ***dom****f, and θ with $0 \leq \theta \leq 1$*

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y). \tag{1.1.3}$$

A function f is strictly convex if strict inequality holds in Equation (1.1.3) for every $x \neq y$ and $0 < \theta < 1$.

Definition 1.1.4. *Triangulation [8] A triangulation is a planar subdivision whose bounded faces are triangles. We note $\mathcal{P} := \{p_1, p_2, ..., p_n\}$ is the set of points in the plane forming the triangulation.*

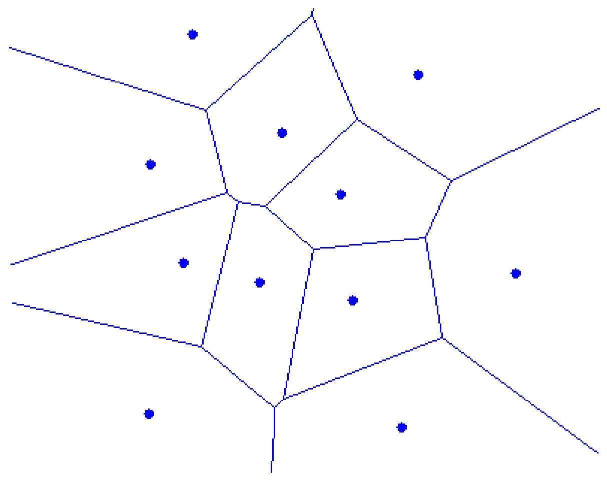
Definition 1.1.5. *Maximal Planar Subdivision [8] A maximal planar subdivision is a subdivision $\mathcal{S}$ such that no edge connecting two vertices can be added to $\mathcal{S}$ without destroying its planarity.*

Definition 1.1.6. *Euclidean Distance [8, pp. 148–149] The Euclidean distance between two points p and q is denoted as $dist(p, q)$. Thus, in the plane we have*

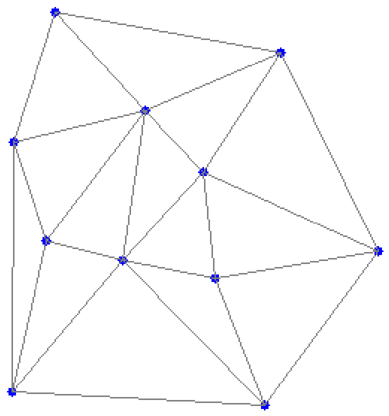
$$dist(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}. \tag{1.1.4}$$

Definition 1.1.7. *Voronoi Diagram [8, pp. 148–149] Let $P := \{p_1, p_2, ..., p_n\}$ be a set of n distinct points in the plane; these points are the sites. The Voronoi diagram of $\mathcal{P}$ is defined as the subdivision of the plane into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $dist(q, p_i) < dist(q, p_j)$ for all $p_j \in P$ with $i \neq j$. The Voronoi diagram of P is denoted by $Vor(P)$. The region of a site p is called the Voronoi cell of p and is denoted by $\mathcal{V}(p)$. A Voronoi diagram is pictured in Figure 1.1(a).*

Definition 1.1.8. *Delaunay Triangulation [8, pp. 190–191] Let the graph $\mathcal{G}$ be the dual graph of a Voronoi diagram constructed as follow: The graph $\mathcal{G}$ has a node for every Voronoi cell and it has an arc between two nodes if the corresponding cells share an edge. This means that $\mathcal{G}$ has an arc for every edge of Vor(P). There is a one-to-one correspondence between the bounded faces of $\mathcal{G}$ and the sites of Vor(P). Consider the straight-line embedding of $\mathcal{G}$, where the node corresponding to the Voronoi cell $\mathcal{V}(p)$ is the point p, and the arc connecting the nodes of $\mathcal{V}(p)$ and $\mathcal{V}(q)$ is the segment $\overline{pq}$. This embedding is called the Delaunay graph of $\mathcal{P}$ and is denoted by $\mathcal{DG}(P)$. A Delaunay triangulation is any triangulation obtained by adding edges to the Delaunay graph. The Delaunay triangulation of P is unique if and only if $\mathcal{DG}(P)$ is a triangulation. Thus a Delaunay triangulation can be characterized by:*



(a) Voronoi Diagram



(b) Delaunay Triangulation

Figure 1.1.1: Related Voronoi and Delaunay Diagrams

Let P be a set of n points in the plane.

i *Three points $p_i, p_j, p_k \in P$ are vertices of the same face of the Delaunay graph of P if and only if the circle through p_i, p_j, p_k contains no points of P in its interior.*

ii *Two points $p_i, p_j \in P$ form an edge of the Delaunay graph of P if and only if there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point of P.*

A Delaunay triangulation is pictured in Figure 1.1(b). This is the dual structure to the Voronoi diagram pictured in Figure 1.1(a).

Definition 1.1.9. *The centroid of a finite set of points $p_1, p_2, ..p_n$ is the arithmetic mean*

$$\frac{1}{n}\sum_{i=0}^{n} p_1. \tag{1.1.5}$$

It lies inside the convex hull [12, p. 106].

Definition 1.1.10. *Piecewise Linear-Quadratic (PLQ) Function [11] A function is a PLQ function if it can be represented by continuous functions with a piecewise linear domain for which the function is either linear or quadratic on each piece of its domain.*

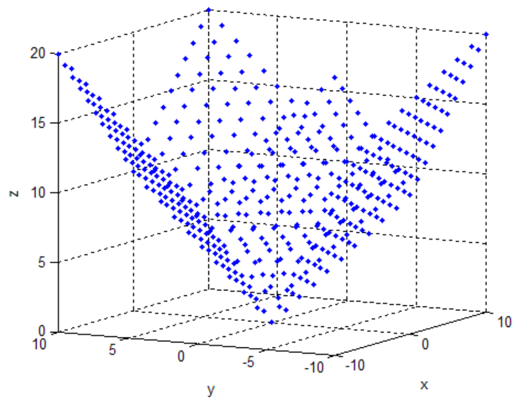
1.2 Problem Definition

We desire to produce a model for a bivariate PLQ function that will be both efficient in space and run-time complexity for building and representing the PLQ as a numerical model, as well as having an efficient run-time complexity under fundamental convex transforms. The initial goal is to represent zero order data, with the user providing input data in terms of a series of points as (x_i, y_i, z_i) in $\mathbb{R}^3$. The model is required to remove any co-planar points inside a face on the convex hull. This results in storage complexity of the model being less than or equal to that of the size of the initial data set. This will minimize the amount of storage required to represent the model without loss of generality.

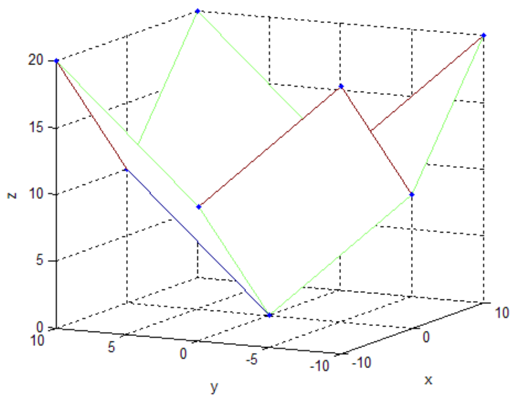
Figure 1.1(a) represents 401 input points over a regular grid. When represented with the PLQ model, as there are numerous co-planar points, the results are simplified such that the model is defined by only 9 vertices as in Figure 1.1(b). With a strictly convex input set as in Figure 1.1(c), which represents 401 input points over a regular grid, no simplification occurs in the representation of the model. Figure 1.1(d) contains 401 vertices due to the fact that the function is strictly convex and no simplification can occur. With the zero-order model, limitations in terms of complexity will be encountered with data that represents a strictly convex function as no more than three points are co-planar on the convex hull. Thus no simplification will occur, presenting a upper bound for storage complexity.

In addition to supporting algorithms to allow for the efficient transformation of the model from one form to another, the model must support the following operations in an efficient manner:

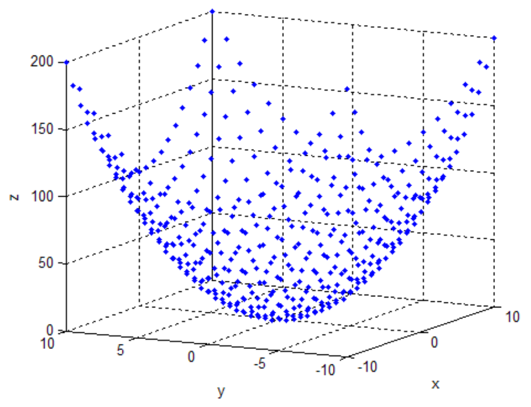
- building a model from zero-order data,
- evaluating the model over an irregular grid of points,
- multiplying by a scalar value, and
- adding of two PLQ functions.



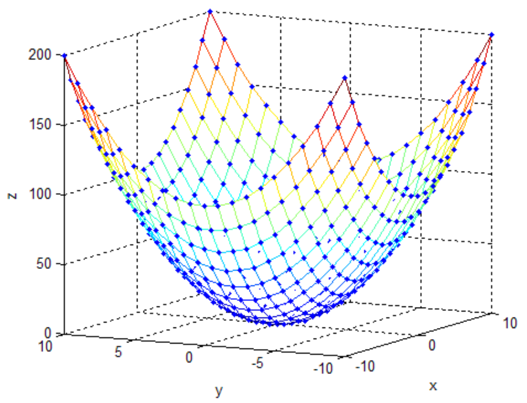
(a) Convex input points for $f(x,y) = |x| + |y|$



(b) Zero-Order model of $f(x,y) = |x| + |y|$



(c) Strictly Convex input points for $f(x,) = x^2+y^2$



(d) Zero-Order Model of $f(x,y) = x^2 + y^2$

Figure 1.2.1: Model Representation over a Convex Set

Chapter 2

Model Selection

2.1 Tessellation-based Model

In examining the problem, it was felt that a PLQ function could be encoded using a Delaunay triangulation, where each data point in the zero-order model would represent a vertex. The Delaunay triangulation is a dual structure of the Voronoi diagram, which has previously been used to encode a convex hull as a finite set of intersecting half-spaces [3], where each face is defined by a Voronoi center $\mathcal{V}$.

The Delaunay triangulation was chosen over the Voronoi diagram as we could explicitly define the domain over which a function is defined. A user would define the vertex point value in the model and the algorithm would compute and project triangulations into $\mathbb{R}^2$. Encoded with each triangulation would be the value of the function as a linear-quadratic equation. Thus a PLQ function in $\mathbb{R}^3$ is encoded as a matrix containing triangulations as $\mathcal{T}$ in $\mathbb{R}^2$ associated with the corresponding linear-quadratic equation as

$$f(X_i) = \begin{pmatrix} a_{i,1} & a_{i,2} \\ a_{i,3} & a_{i,4} \end{pmatrix} X_i^2 + \begin{pmatrix} b_{i,1} \\ b_{i,2} \end{pmatrix} X_i + c_i. \tag{2.1.1}$$

where X_i is the point (x_i, y_i) at which the linear-quadratic function is to be evaluated. Equation (2.1.1) encodes the function value of each part of the PLQ to the boundaries of the polytope. To encode the information as a one-dimensional array we rewrite Equation (2.1.1) as

$$\begin{pmatrix} a^i_{1,1} & a^i_{1,2} & a^i_{2,1} & a^i_{2,2} & b^i_{1,1} & b^i_{1,2} & c^i \end{pmatrix}. \tag{2.1.2}$$

By definition, the points $\mathcal{P} := \{p_1, p_2, ..., p_n\}$ that form the set of points in the plane, have a one-to-one mapping onto $\mathcal{T}$, but now we can explicitly define the boundaries over which Equation (2.1.1) applies. What we need is not unlike the process of building a terrain map, but with the constraints applied by the conditions of a bivariate PLQ.

The model that we propose, is a function $proj : A \subset \mathbb{R}^3 \to \mathbb{R}^2$, that assigns a linear-quadratic function value, $f(p)$ to every point within a planar subdivision in the *domain* $\mathcal{A}$. As with a terrain map, the value of the bivariate PLQ is not necessary known at every point in the zero-order model, but defined at certain points such that we only know the value of the function $proj$ at a finite set $\mathcal{P} \subset \mathcal{A}$ [8].

The initial challenge is how to determine a triangulation of $\mathcal{P}$, such that each $p_i \in \mathcal{P}$ is a vertex for each planar sub-division and that each planar sub-division is a triangle. Following that, we then need to project the triangulation from $\mathbb{R}^3 \to \mathbb{R}^2$. Additionally, we need to compute the function coefficient values for Equation (2.1.1) for each planar subdivision.

Initial investigation was completed with the Scilab mathematical package [2]. Unfortunately, Scilab does not have the necessary functionality to support the operations we were looking for without a significant amount of time invested into writing the basic algorithms. The mathematical package MATLAB [1] was found to offer the necessary functionality to support the desired operations. The nature of the specific algorithms will be expanded in the following sections.

Using MATLAB as the computational platform, we were able to develop a data structure that met the requirements to encode a bivariate PLQ. Each planar subdivision is defined by a tessellation which is composed of two separate parts: one being the set of points $(x_i, y_i) \in \mathcal{P}$, which is now a *maximal planar subdivision*, and the other being a *triangulation of* $\mathcal{P}$, which is defined as follows

Definition 2.1.1. *Triangulation of $\mathcal{P}$ [8, p. 185] A triangulation $\mathcal{T}$ of $\mathcal{P}$ is a maximal planar subdivision whose vertex set is $\mathcal{P}$*

To build the bivariate PLQ from zero-order data, a convex hull algorithm was run on the data set in $\mathbb{R}^3$. The resulting convex hull was then projected into $\mathbb{R}^2$, providing the required triangulations. Additionally, the coefficient values for Equation (2.1.1) were computed from vertex points in the convex hull, forming the necessary components to represent the bivariate PLQ. The data structure has two distinct components: a matrix of $(x_i, y_i) \in \mathcal{P}$ where $\mathcal{P}$ contains the location in $\mathbb{R}^2$ of each vertex. A second matrix encodes triangulation $t_i \in \mathcal{D}$ as a set of indexes into $\mathcal{P}$ along with the corresponding linear-quadratic as in Equation (2.1.2). Let $\mathcal{D}$ be a triangulation $\mathcal{T}$ that satisfies the properties of a Delaunay tessellation. The model can be written

$$\mathcal{P} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ x_{n_v} & y_{n_v} \end{pmatrix} \tag{2.1.3}$$

$$\mathcal{D} = \begin{pmatrix} i_{1,1} & i_{1,2} & i_{1,3} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & b_{1,1} & b_{1,2} & c_1 \\ i_{2,1} & i_{2,2} & i_{2,3} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & b_{2,1} & b_{2,2} & c_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ i_{n_t,1} & i_{n_t,2} & i_{n_t,3} & a_{n_t,1} & a_{n_t,2} & a_{n_t,3} & a_{n_t,4} & b_{n_t,1} & b_{n_t,2} & c_{n_t} \end{pmatrix} \tag{2.1.4}$$

where $i_{j,k}$, ($k \in \{1, 2, 3\}$ and $j = 1..n_t$) are the indexes of the associated points in $\mathcal{P}$ with n_v and n_t being the number of vertexes and triangulations respectively.

This model supports the desired convex transformation operations efficiently. The description and analysis of each component is discussed in Chapter 3. The storage complexity of the model is $O(n)$ in terms of the number of triangulations as there is one entry for each triangulations and at most 3 vertexes for each triangulation. The only shortcoming to representing a bivariate PLQ with this data structure is that it cannot represent an unbounded bivariate PLQ due to the fact that the union of the bounded faces of a triangulation $\mathcal{D}$ is the convex hull of $\mathcal{P}$ [8]; hence, it cannot encode an unbounded region and is limited to representing bounded domains.

2.2 Inequality-based Model

In order to support the inclusion of unbounded domains into the model, a dual model was investigated using inequalities instead of vertices in $\mathcal{P}$ to define each triangulation $t_i \in \mathcal{T}$.

This is accomplished through a series of operations that convert a tessellation-based PLQ to an inequality-based PLQ. Each $t_i \in \mathcal{T}$ is now represented by several half-spaces in $\mathbb{R}^2$ with the associated linear-quadratic equation. The matrix $\mathcal{P}$ is replaced by a list of inequalities $\mathcal{I}$, where $(m_i, s_i, b_i) \in \mathcal{I}$ are the set of half-spaces that define $\mathcal{T}$. Each component of $(m_i, s_i, b_i) \in \mathcal{I}$ encodes the slope m, the sign of the inequality s and the intercept b respectively. Let $\mathcal{I}$ be a set of inequalities, $\mathcal{T}$ be a set of triangulations, $\mathcal{C}$ be a set of centroids and $\mathcal{N}$ be a set of neighbouring faces. The model becomes

$$\mathcal{I} = \begin{pmatrix} m_1 & s_1 & b_1 \\ m_2 & s_2 & b_2 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ m_{n_s} & s_{n_s} & b_{n_s} \end{pmatrix} \tag{2.2.1}$$

$$\mathcal{T} = \begin{pmatrix} i_{1,1} & i_{1,2} & i_{1,3} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & b_{1,1} & b_{1,2} & c_1 \\ i_{2,1} & i_{2,2} & i_{2,3} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & b_{2,1} & b_{2,2} & c_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ i_{n_t,1} & i_{n_t,2} & i_{n_t,3} & a_{n_t,1} & a_{n_t,2} & a_{n_t,3} & a_{n_t,4} & b_{n_t,1} & b_{n_t,2} & c_{n_t} \end{pmatrix} \tag{2.2.2}$$

$$\mathcal{C} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ x_{n_t} & y_{n_t} \end{pmatrix} \tag{2.2.3}$$

$$\mathcal{N} = \begin{pmatrix} i_{1,1} & \cdot & i_{1,n_{n_1}} \\ i_{2,1} & \cdot & i_{2,n_{n_2}} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ i_{n_t,1} & \cdot & i_{n_t,n_{n_t}} \end{pmatrix} \tag{2.2.4}$$

where $i_{j,k}, k \in \{1, 2, 3\}$ in the second matrix are the indexes of the associated inequalities in $\mathcal{I}$ and n_s, n_t, n_{n_t} are the number of half-spaces, the number of triangulations and the number of neighbours for a specific triangulation respectively. The array of lists $\mathcal{N}$ maintains the indexes of each neighbouring triangulation is $\mathcal{T}$ to a triangulation t_i.

The inequality model introduces two additional components that are used to support the efficient transformations as a third and fourth element. The third element is a matrix of centroids. Each row stores the centroid for the triangulation defined by the given row. The fourth element is a cell list, and stores a list of indexes to the neighbours of the triangulation defined by a given row. A cell list is a data structure specific to MATLAB which allows each row to have a different number of elements. This is critical and is done to minimize the storage complexity as each triangulation can have a different number of neighbour. This list must also be able to grow in case new neighbouring triangulations are added.

This model supports the desired convex transformation operations in terms of efficient evaluation, addition and multiplication, but must be constructed from a bounded bivariate PLQ model. The description and analysis of each component is discussed in Chapter 4.

Chapter 3

Algorithms for the Bounded Tessellation-Based Bivariate PLQ Model

3.1 Build Algorithm

In constructing a bivariate PLQ, both the tessellation and the inequality-based models rely on the same initial build function. The build function, working only with zero-order data, receives as input an irregular grid of points X in $\mathbb{R}^2$ and a vector of corresponding function values, $f(X)$. The build algorithm is

Algorithm 1: PLQ2_build Algorithm

```
Input: $X, f$
Output: a tessellation-based bounded PLQ2
begin
    initialization;
    Compute the convex hull, $\mathcal{H}$ of the data set in $\mathbb{R}^3$;
    Project the vertices $\mathcal{P}$ that form $\mathcal{H}$ into $\mathbb{R}^2$;
    Compute the Delaunay triangulation $\mathcal{D}$ $\forall p \in \mathcal{P}$;
    forall $t_i \in \mathcal{D}$ do
        compute the coefficients for linear-quadratic function;
    end
    Return PLQ;
end
```

The convex hull algorithm used in this implementation, is based on the Quickhull algorithm [4]. The algorithm will remove extra points that are located on the interior of each triangulation [8, 4].

Let us recall some facts

- The number of points processed by the convex hull algorithm is proportional to the number of vertices in the output and runs in $O(n \log h)$, where h is the output size [7].
- A convex polytope $\mathcal{P}$ with n vertices has at most $2n - 4$ faces [8, p. 237].
- The implementation of Quickhull in MATLAB produces a convex hull with triangular faces with three vertices.

- The expected number of triangles created by the Delaunay Triangulation algorithm is as most $9 * n + 1$ [8, p. 197].

Proposition 3.1.1. *The PLQ2_build algorithm runs in $O(n \log h)$.*

Proof. Let $\mathcal{P}$ be a convex polytope produced by the algorithm in $O(n \log h)$ which has at most $2h - 4$ facets. The projection of P from $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ requires each face to be visited only once and for each face, only three points are projected. Thus, let $\mathcal{P}$ be the set of points in $\mathbb{R}^2$ and the total number of operations to project into $\mathbb{R}^2$ is $3 * (2n - 4) = 6n - 12 = O(n)$.

By Definition 1.1.8, the Delaunay triangulation $\mathcal{D}$ of a set $\mathcal{P}$ of $6n - 12$ points can be computed in $O(n \log n)$ and produces at most $9 * (6n - 12) - 1 = 54n - 107$ faces. For a single face, the linear-quadratic equation coefficients is computed in $O(1)$. It then follows that the coefficients for all triangulations in $\mathcal{D}$ can be computed in $O(54n - 107) = O(n)$. So the total number of operations required for the PLQ2_build algorithms is $O(n\text{log}h) + O(n) + O(n\text{log}n) + O(n) = O(n\text{log}h)$. □

3.2 Evaluation Algorithm

The evaluation of the bounded bivariate PLQ function over a set of points $(x_i, y_i) \in X$ is achieved exploiting the fact that the model is based on a tessellation. MATLAB includes a function called tsearch in the convex hull package that allows a point or set of points to be evaluated over a Delaunay triangulation $\mathcal{D}$ and will return the index of the face that the point is included in. By definition $\mathcal{D}$ encodes a convex hull which is a continuous piecewise function. If a point (x, y) falls on the boundary of a triangulation, it can be evaluated as part of either triangulation. If a point that is being evaluated is not in the domain of the model, the value returned is $+\infty$.

Algorithm 2: PLQ2_eval Algorithm

```
Input: PLQ2,$X$
Output: a vector of values $Z$
begin
    initialization;
    forall $(x_i, y_i) \in \mathcal{D}$ do
        Compute facet indexes $j \in \mathcal{I}$ from $\mathcal{D}$;
    end
    forall $i$ do
        evaluate the linear-quadratic function at $(x_i, y_i)$ associated with the facet at
        index $j_i$;
        insert results of evaluation at $Z(i)$;
    end
    Return Z;
end
```

The computational complexity of the tsearch algorithm as published in MATLAB was not available through the literature. Thus, an empirical approach was taken to evaluate the run-time complexity of the bounded evaluation algorithm. A series of experiments were run with two different scenarios: the first being evaluating a fixed number of points over an increasing number of facets in the model. The second being the evaluation of an increasing number of points over a fixed number of facets in the model.

Conjecture 3.2.1. *Tsearch runs in $O(nk)$, where n in the number of facets in $\mathcal{D}$ and k is the number of points being evaluated.*

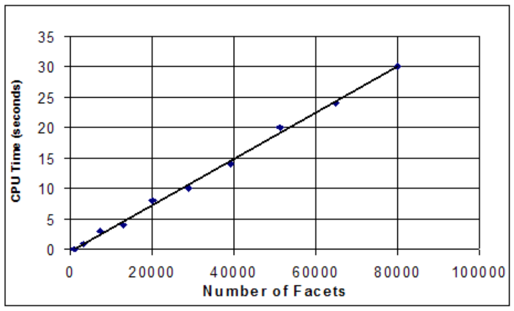
In Figure 3.1(a), 100 points were evaluated in a PLQ model that had a maximal number of n facets (strictly convex data) with n growing to 80,000. It was observed that the time for evaluating a fixed number of points increased linearly with the number of facets in the model. In Figure 3.1(b), the number of input points to be evaluated was increased up to 250,000 points over a model with 80,000 facets in $\mathcal{D}$. The computational time was found to increase linearly as the number of points being evaluated. Thus the tsearch algorithm appears to run in $O(nk)$.

Returning to the PLQ2_eval algorithm, we continue with analyzing the remainder of the algorithm.

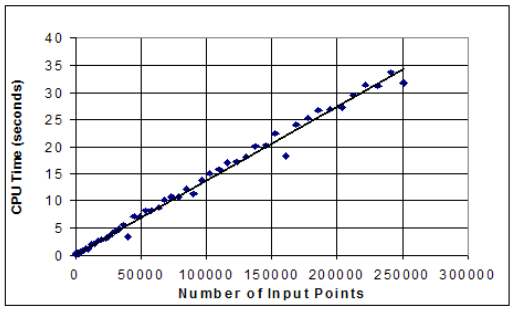
Proposition 3.2.1. *Assuming that tsearch runs in $O(nk)$, the PLQ2_eval algorithm runs in $O(nk)$.*

Proof. Let X be a set of input points of size k over $\mathcal{D}$ with n facets. From Conjecture 3.2.1, the runtime complexity will be $O(nk)$ producing an index list $\mathcal{I}$ of size k as per the definition of the function in MATLAB. To evaluate the function value at each point in X, the linear-quadratic function to be evaluated can be accessed and evaluated in $O(1) + 7 = O(1)$. It then follows that accessing k linear-quadratic functions can be completed in $O(k)$ and the runtime complexity of the algorithm is $O(nk) + O(k) = O(nk)$. □

Note: A better implementation of tsearch may improve its complexity to $O(n \log k)$ therefore improving the complexity of PLQ2_eval to $O(n \log k + k)$.



(a) Timing results for evaluating a fixed number of points in the tessellation over an increasing number of facets



(b) Timing results for evaluating a increasing number of input points over 80,000 facets

Figure 3.2.1: Empirical results from the tsearch algorithm in MATLAB

3.3 Addition Algorithm

The PLQ2_add algorithm takes as input, two bounded bivariate models and returns a bivariate PLQ that is the addition of the two input models. The addition algorithm is a union of the vertex points $\mathcal{P}$ in $\mathcal{D}$ under evaluation of both input models. The algorithm is as follows.

Algorithm 3: PLQ2_add Algorithm

```
Input: $PLQa, PLQb$
Output: a PLQ that is a summation of the two input models
begin
    initialization;
    for each vertex point $p_i \in \mathcal{P}_a$ in PLQ2a do
        evaluate the function value of $p_i$ under PLQb to form a vector $Z_a$;
    end
    for each vertex point $p_j \in \mathcal{P}_b$ in PLQ2b do
        evaluate the function value of $p_j$ under PLQa to form a vector $Z_b$;
    end
    Concatenate $P_a$ and $P_b$ to form a new point list $P_c$;
    Concatenate $Z_a$ and $Z_b$ to form a new point list $Z_c$;
    Evaluate $P_c$ and $Z_c$ under the PLQ2_build to build PLQ2c;
    Return PLQ2c;
end
```

Proposition 3.3.1. *The PLQ2_add algorithm runs in $O(n^2)$.*

Proof. Let n be the number of triangulations in PLQa and m be the number of triangulations in PLQb. Let j be the number of vertex points in PLQa (at most $j = 3n$) and k be the number of vertex points in PLQb (at most $k = 3m$). It then follows that the evaluation of PLQa in PLQb is $O(3nm)$ and the evaluation of PLQb in PLQa is $O(3mn)$. When $n = m$, the total cost is $O(3n^2) + O(3n^2) = O(n^2)$ with $j + k = 3m + 3n = 6n$ points. The computational complexity of building the resulting new PLQ from the data points by Proposition 3.1.1 is $O(6n \log 6n) = O(n \log n)$. It then follows that the computational complexity for the addition algorithm is $O(n^2) + O(n \log n) = O(n^2)$. □

3.4 Scalar Multiplication Algorithm

Scalar multiplication for a PLQ function is a straightforward linear-time algorithm in which the linear-quadratic equation for each face in the PLQ model is multiplied by a scalar value.

Algorithm 4: PLQ2_mult Algorithm

```
Input: PLQ2,$c$
Output: a PLQ2 that has been scalar multiplied by $c$
begin
    initialization;
    forall triangulations $t_i \in \mathcal{D}$ in PLQ2 do
        multiply the linear-quadratic equation at index $i$ by the constant $c$;
    end
    Return PLQ2;
end
```

Proposition 3.4.1. *The PLQ2_mult algorithm runs in $O(n)$.*

Proof. Let n be the number of triangulations in the PLQ and let x be a scalar value. Looping on n, indexing the value of the linear-quadratic function for each triangulation in R^2 can be completed in $O(1)$ by the fact that indexing into a matrix can be done in linear time. Computing the new linear-quadratic equation from the scalar multiplication of k with the linear-quadratic equation, Equation (2.1.2), is completed in $O(1)$. Updating the linear-quadratic equation at index i can be done in linear time. In then follows that each triangulation's linear-quadratic equation must be accessed taking n operations. Hence the total number of operations is $n * (O(1) + O(1) + O(1)) = O(n)$. □

Chapter 4

Algorithms for the Unbounded Inequality-Based Bivariate PLQ Model

4.1 Algorithm for Converting a Tessellation-Based Model to an Inequality-Based Model

The algorithm to convert a bounded tessellation-based bivariate PLQ to an unbounded bivariate inequality-based PLQ takes as an argument a bounded PLQ and returns and unbounded PLQ as IPLQ2. The algorithm is composed of three distinct operations: first, the bounded tessellation model is converted to an inequality model. Second, the unbounded region is constructed as a IPLQ2 model, and third, the two models are joined together.

The first set of operations is computed by treating each vertex point of a specific triangulation as the intersection of two inequalities formed by the intersection point and the remaining two vertices. This method exploits the property of the Delaunay Triangulation in MATLAB, in such that each face is triangular and will only have three vertices. The slope m and intercept value b of each inequality are computed from the vertex information. Each inequality *ineq* is encoded in the point-slope form of $y = mx + b$ as $(m\ sign\ b)$. The inequality value *sign* must then be determined. By the properties of a closed convex polytope, any point that is on the boundary of the polytope is within the convex hull, thus as each inequality *ineq* is uniquely formed by only two vertices. The third vertex that is not taking part in the formation of the inequality *ineq* is used as a test point as it will, by definition, be in the half-space that forms part of the initial triangulation. Values of *sign* are encoded as in Table 4.1.1.

Special consideration must be given to cases where the slope m of the line is $+\infty$ or $-\infty$ as the inequality is in the form of $y = mx + b$. To capture and encode this information, a special case is allowed where m can take on the values of $+\infty$ or $-\infty$. In this case, the value of b encodes the value of the x-intercept.

Inequalities, as described in Equation (2.2.1) are stored in a matrix of inequalities and each face then maintains a list of indices to the required inequality. As the model is being constructed, duplicate checking is conducted to ensure that existence of an equality in the matrix is unique.

The unbounded region around the convex polytope that represents the triangulation in $\mathbb{R}^2$ is computed by touring around the exterior of the convex hull. Based on the properties of

Sign		Value
$\geq$	encoded as	1
$=$	encoded as	0
$\leq$	encoded as	-1

Table 4.1.1: Sign value encoding for inequalities for the bounded polytope

the convex hull, for a convex hull tour, the first point past the current segment being tested, will be used as a test point to determine the unbounded regions. The computation of the convex hull of the bounded model is performed using the convhull function in MATLAB [1, 4] and the vertex points from the Delaunay triangulation is $\mathbb{R}^2$. This returns the vertices that form the convex hull in a strict ordering, such that indexing through the vertices will form a tour around the convex hull returning to the original vertex. The assumption is made that the hull consists of at least three points. If the convex hull has only two points, then it is a line and is treated as a special case.

Unlike the bounded region where a point on the boundary of a polytope is inclusive of the region, in the unbounded region a point on the boundary of the convex polytope may not belong to the half-space defined by the inequalities defining the region. Thus $<$ and $>$ were added as special cases for unbounded conditions: if a point is sitting on the convex hull of the tessellation in $\mathbb{R}^2$, it is in the bounded model. As there could exist a discontinuity between the convex set and the unbounded region, a point on the hull cannot be equal to the linear-quadratic equation for both spaces. Thus, the unbounded region as expressed by the inequalities, must bound the convex hull with a strict inequality, hence the need for the introduction of $<$ and $>$ as in table 4.1.2.

In the calculation of the inequalities forming the unbounded region, the tour around the hull will form a series of unbounded faces consisting of two inequalities; The first, will be the strict inequality that forms the boundary of the convex hull. The second point may either be an inequality or strict inequality as it is only forming a partition between two unbounded faces that evaluate to $+\infty$. Each unbounded region will be formed by no more than two separate inequalities. Again, as with the bounded region, cases of infinite slope are encoded in the same fashion. The tessellation of the unbounded regions are encoded in the same fashion with the IPQ2 data structure.

With conversion of the bounded model to an unbounded model, being represented by two separate parts, the unbounded and bounded inequality models are merged into a single data structure. This can be accomplished by a concatenation of the two models, but as at least half of the inequalities that form the unbounded region take part in the formation of the bounded inequality model, duplication of inequalities is inevitable. Thus, duplicate inequalities are removed and existence of inequalities in one list need to be verified in the other.

MATLAB does not contain a native binary search method, which would be a suitable algorithm for checking the existence of an inequality in a matrix of inequalities. In the MATLAB file exchange, a binary search algorithm (that given a value in a sorted vector,

Sign		Value
$>$	encoded as	2
$\geq$	encoded as	1
$=$	encoded as	0
$\leq$	encoded as	-1
$<$	encoded as	-2

Table 4.1.2: Sign value encoding for inequalities for bounded and unbounded regions

returns the index of location where the value is) was found [9]. As this algorithm only supports vectors, the method was expanded to support a row search in a matrix, such that the algorithm returns the index of a row match for a given inequality. Utilizing this method, the two models can be merged and for duplicate entries, as the binary search returns the index of the initial duplicate row, the reference is updated such that the model being merged has its duplicated entries updated with the indexes from the base model.

In looking forward to other parts of the algorithm, additional information needs to be encoded during the conversion algorithm. Unlike the Voronoi diagram, the Delaunay tessellation does not maintain uniqueness in terms of using a center to treat the problem as a point location problem. In order to accelerate the evaluation, additional methods have been introduced such that each triangulation also has an associated center of gravity or centroid as seen in Model (2.2.3). Thus, the centroid is computed for each face and stored in the triangulation row entry.

Additionally, a neighbour list is constructed during the conversion that allows each triangulation to maintain a list of other triangulations it is next to. For each triangulation t_i, an index list of neighbouring triangulations is maintained such that the neighbour will only appear in the list if and only if it has a common edge with t_i. The centroid and neighbour list methods have not been fully implemented for the unbounded regions.

Algorithm 5: PLQ2_convert_to_IPLQ2 Algorithm

```
Input: PLQ2
Output: as inequality-based PLQ2
begin
    initialization;
    forall triangulations $t_i \in \mathcal{D}$ in PLQ2 do
        Compute the linear-equalities that form the triangulation;
        Compute the centroid for the face;
        forall edges in the triangulation do
            record the index of the neighbouring triangulation;
        end
        Check for duplicate entries in the existing list of inequalities;
    end
    Compute the convex hull on $\mathcal{P}$ forming $k + 1$ hull points and $k$ edges;
    forall edges k in the convex hull do
        compute the 2 bounding inequalities and build $2k$ unbounded faces with at
        most $2k$ inequalities;
    end
    forall unbounded inequalities do
        check to see if it exists in the list of bounded equalities and if it does update
        the index in the face inequality reference list;
    end
    Concatenate the two models;
    Return the inequality-based IPLQ2;
end
```

Proposition 4.1.1. *The PLQ2_convert_to_IPLQ2 algorithm runs in $O(n^2)$.*

We now recall some facts

- Let $\mathcal{P}$ be a set of n points in the plane, not all collinear, and let k denote the number of points in $\mathcal{P}$ that lie on the boundary of the convex hull of $\mathcal{P}$. Then any triangulation of $\mathcal{P}$ has $2n - 2 - k$ triangles and $3n - 3 - k$ edges.

- The Delaunay triangulation contains at most $O(n^{d/2})$ simplexes, thus in $\mathbb{R}^2$, for n input points, there will be at most $O(n)$ triangulations. For $d = 2$ the model will have at most $O(k)$ triangulations.

- In the plane $\mathbb{R}^2$, if there are b vertices on the convex hull, then any triangulation of the points has at most $2n - 2 - b$ triangles, plus one exterior face [8].

- The centroids of a set of n points in d dimensions can be computed trivially in $O(dn) = O(n)$ [12, p. 106].

We are now ready to prove Proposition 4.1.1

Proof. Let Q be a bounded bivariate PLQ model with n input points with at most n triangulations $t_i \in \mathcal{D}$. For each t_i, there are three vertices that form the face thus the face has three edges for which three linear-inequalities are needed to represent the triangulation. The equation of each edge is computed in linear time using the point slope and intercept form for each set of points. Each face can be represented by 3 inequalities in $O(1)$ and all inequalities calculated are $n * O(1) = O(n)$. Checking to see if a single inequality exist in the inequality list is in $O(\log m)$ where m is the number of inequalities. It follows, as the model has at most n triangulation, and each has 3 edges, that $m = 3n$ and $O(\log m) = O(\log(3n)) = O(\log n)$; thus, for n faces, checking for duplicates is $O(n \log n)$.

To compute the centroid of the new triangulation, the centroid for a single triangulation can be computed in $O(p)$ where $p = 3$ and is the number of vertices in the triangulation. Thus, for n triangulations, the complexity is $O(3n) = O(n)$.

To compute the neighbour list, each triangulation maintains three vertices. For each set of vertex that participates in an edge, we index into the triangulation matrix and recover the members in at worst $O(n)$. A bottleneck is introduced by the fact that a single vertex could be a member of every triangulation in the model; hence its triangulation membership is n. This cycle is repeated for each vertex in the triangulation; thus for n triangulations, the worst case complexity for building the neighbour list is $O(n * n) = O(n^2)$.

To compute the unbounded regions, by Definition 1.1.8 and 1.1.5 the Delaunay tessellation is a maximal planar subdivision, thus each point in $\mathbb{R}^2$ is a vertex. It then follows that there are b vertices and b number of points, thus $b = n$ and the number of triangles is

$$\begin{aligned} 2n - 2 - b + 1 &= 2n - 2 - n + 1 \\ &= n - 2 + 1 \\ &= n - 1. \end{aligned}$$

Thus, the number of triangles is at most $n - 1$ for $\mathcal{D}$. Let k be the number of edges in the convex hull of $\mathcal{D}$. The number of edges in the convex hull is be related to the number of triangles as $2m - 2 - k$ where m is the number of input points. Thus, for the unbounded PLQ where n is the number of triangles, the number of edges is $n = 2m - 2 - k$. With $m = n$ we have $n = 2n - 2 - k$ so $k = n - 2 = O(n)$.

There are at most $n - 2$ edges in the convex hull of $\mathcal{D}$ and for each edge, two inequalities are required which can be each computed in $O(1)$. Hence, at most $2k = 2n - 4$ inequalities are required, each taking $O(1)$, and the inequalities for the unbounded region are computed in $O(n)$ with at most $2n - 4$ inequalities.

In joining the two models, the existence of an unbounded inequality in bounded equality list can be checked in $O(\log n)$. Thus $2n-4$ inequalities can be checked in $O((2n-4)\log n) = O(n \log n)$. With any duplicate inequalities removed, the two models can be concatenated in $O(1)$. It follows that the run time complexity for the conversion of a bounded tessellation-based bivariate PLQ to an unbounded inequality-based bivariate PLQ without the neighbour list structure or centroid list is $O(n) + O(n \log n) + O(n) + O(n \log n) + O(1) = O(n\mathrm{log}n)$. Additionally, the run time complexity for the conversion of a bounded tessellation-based bivariate PLQ to an unbounded inequality-based bivariate PLQ with the neighbour list structure or centroid list is $O(n) + O(n) + O(n^2) + O(n \log n) + O(n) + O(n \log n) + O(1) = O(n^2)$. $\square$

4.2 Evaluation Algorithm

The evaluation of an unbounded inequality-based bivariate PLQ over a set of points $(x_i, y_i) \in X$ is achieved by searching through the triangulations formed by the inequalities and determining if a specific point is within a given region. MATLAB does not support functionality to search through triangulations as defined by inequalities, thus for the initial method, the triangulations will be toured through in order. Fortunately, the triangulations have an inherent natural ordering from the build function. By definition, if a point (x, y) falls on the boundary of a triangulation, it can be evaluated as part of either facet. If a point that is being evaluated is not in the domain of the model, the index is returned as $+\infty$. Two different methods are presented. Only Method A will be proven as Method B is not fully implemented.

In order to accelerate the evaluation, points are ordered in lexicographic order to exploit the fact that in the process of the build operation, there is an inherent ordering of triangulations. Thus, it is possible, if both X and $\mathcal{T}$ share the same ordering, to determine the triangulation in which each point is located in a linear fashion. Once a triangulation is found to bound the point under evaluation, it is evaluated using the associated linear-quadratic function.

Algorithm 6: IPLQ2_eval Algorithm

```
Input: IPLQ2,$X$
Output: a vector of values $Z$
begin
    initialization;
    Order $X$ in increasing lexicographic order;
    Set $i, j = 0$;
    for each $x_i \in X$ do
        if $x_i$ satisfies the constraints of the triangulation $t_j$ then
            evaluate with associated linear-quadratic to $Z$ and check $x_{i+1}$;
        end
        check $x_i$ in $t_{j+1}$;
        if $j \geq$ size($\mathcal{T}$), the number of triangulations then
            reset $j = 0$;
        end
    end
    Reorder $Z$ to original order;
    Return $Z$;
end
```

A second approach was investigated to improve the performance. Using the centroids for each face, it was conjectured that the problem could be framed as a point location type problem. Even though locating a point to a given center may not absolutely locate a point within a face, we conjecture that this significantly decrease the search space. Once a point has been located to a centroid, the point is evaluated under the bounds of the triangulation associated with the given centroid. If the point is not in the bounded region, we take the next closest centroid and so on. The points under evaluation are still ordered in increasing lexicographic order, and once a triangulation is found that contains a point, then all subsequent points are evaluated under that triangulation until a point is found that is not in the bounded face. At this point, the centroid search is computed again and repeated until all points have been evaluated.

Algorithm 7: IPLQ2_eval_v2 Algorithm

```
Data: IPLQ2,$X$
Result: a vector of values $Z$
begin
    initialization;
    Order $X$ in increasing lexicographic order;
    Set $i, j = 0$;
    forall $x_i \in X$ do
        Search the $j$ triangulations for the closest centroid;
        if the point is in triangulation $t$ then
            evaluate with associated linear-quadratic to $Z$;
            check $x_{i+1}$ is in the face;
        end
        check $x_i$ in the next closest face;
    end
    Reorder $Z$ to original order;
    Return $Z$;
end
```

We first recall known results

- Sorting a set of items has a computational run-time complexity of $O(n\log n)$ and a storage complexity of $O(n)$.
- Searching a set of sorted items has a computational run-time complexity of $O(\log n)$.
- If a point p is inside the convex hull of a Delaunay triangulation, its nearest vertex need not be one of the vertices of the triangle containing p.

Proposition 4.2.1. *The evaluation of k points in a model of n triangulations using Algorithm 6 takes $O(kn)$.*

Proof. Let k be the number of points $(x_i, y_i) \in X$ where $0 \leq i < k$ to be evaluated in a bivariate inequality-based model IPLQ2, that contains n triangulations. Ordering of X is accomplished using a standard sorting algorithm which runs in $O(k\log k)$. For a single (x_i, y_i), the worst case is that n triangulations must be searched until a legitimate bounded triangulation can be found. If this is the case for every $x_i \in X$, then locating the valid triangulation for each point is $O(kn)$.

Once a legitimate bounded triangulation is found for a point, the linear-quadratic function can be evaluated in $O(1)$ as stated in Proposition 3.2.1. It then follows that the evaluation of k points can be completed in $O(k)$ which is also a tight bound as all k points must be evaluated; the linear-quadratic function evaluation is $\Theta(k)$. Returning the points to their original order is $O(k)$. The worst-case run time complexity for evaluating k points is $O(k\log k) + O(kn) + O(8) + O(k) = O(kn)$. □

Conjecture 4.2.1. *Algorithm 7 has a run-time complexity of $O(n)$*

4.3 Addition

4.3.1 Algorithm

The IPLQ2_add algorithm takes as input, two unbounded inequality-based bivariate PLQ models of size n and returns an inequality-based bivariate PLQ that is the addition of the two input models. The addition method for the inequality-based model is functionally very similar to the bounded addition but operationally it is quite different. The addition algorithm starts with one inequality-based PLQ as a base model $PLQ2a$. To $PLQ2a$, the inequalities from the second PLQ, $PLQ2b$ are added one-by-one. As each inequality is added, it will intersect n triangulations and divide into $2n$ convex polytope tessellations. The resulting split may not be triangular and on each new tessellation, the centroid and neighbour list is calculated. Once all of the inequalities have been added, the resulting model contains the union of the tessellations from $PLQ2a$ and $PLQ2b$, but the linear-quadratic function value for each tessellation must be computed.

Using the new centroid list $\mathcal{C}'$, the centroids are evaluated under $PLQ2a$ and $PLQ2b$ except instead of evaluating the point, the linear-quadratic function is extracted from each and the summation of the two linear-quadratic equations stored in the relative tessellation associated with each $c_i \in \mathcal{C}'$. We conjecture that this problem can be formulated as a map overlay problem which Becker et al. have shown to have a complexity of $\Theta(n^2)$[5]. This method is not fully implemented at this time.

The addition method utilizes the neighbour list to determine the direction of intersection when an inequality is added to streamline the addition process. The algorithm is as follows:

```
Algorithm 8: IPLQ2_add Algorithm
Input: IPLQ2a,IPLQ2b
Output: IPLQ2 which is the summation of the two input IPLQ2
begin
    initialization;
    Order $X$ in increasing lexicographic order;
    Set $i, j = 0$;
    forall inequalities $i_i \in \mathcal{I}$ in PLQ2b do
        Add the inequality to PLQ2a as PLQ2c;
        Locate a triangulation cut by the inequality;
        forall triangulations bisected by the inequality do
            form two new tessellations for each face split;
            Compute the centroids for each new face formed;
        end
    end
    Evaluate the centroids for each face under PLQ2a and extract the linear-quadratic
    functions;
    Evaluate the centroids for each face under PLQ2b and extract the linear-quadratic
    functions;
    Sum the two sets of linear-quadratic functions;
    Update the linear-quadratic functions for each tessellation in PLQ2c;
    Return PLQ2c;
end
```

Proposition 4.3.1. *The addition algorithm has a computational run time complexity of $O(n^3)$ and a computational storage complexity of $O(n^2)$.*

Proof. Let PLQ2a be an unbounded inequality-based bivariate PLQ model of n triangulations and let PLQ2b be an unbounded inequality-based bivariate PLQ model of m triangulations. As PLQb have n triangulations, by conjecture 4.1.1, it contains no more than $2n-4 = O(n)$ inequalities. For each inequality in PLQb, it is added to PLQa which contains n triangulations and if n triangulations are bisected, $2n$ tessellations are formed from the addition of a single inequality. Thus adding n inequalities forms at most n^2 new faces

$$\begin{aligned} n + \sum_{i=n+1}^{2n} i &= n + \sum_{i=1}^{n}(i+n) \\ &= n + \sum_{i=1}^{n} i + \sum_{i=1}^{n} n \\ &= n + \frac{(n+1)(n)}{2} + n^2 \\ &= n + \frac{n^2}{2} + \frac{n}{2} + n^2 \\ &= \frac{3n^2}{2} + \frac{3n}{2} = O(n^2) \end{aligned}$$

□

For each inequality that is added, $2n$ tessellation are formed and $2n$ centroids are computed in $O(n)$ as stated in Section 5; thus for n inequalities added, n^2 centroids $\mathcal{C}'$ are computed. To extract the linear-quadratic function from PLQ2a and PLQ2b, $\mathcal{C}$ is evaluated under each model, returning the function instead of the resultant of the function. By Proposition 4.3.1 the extraction of the linear-quadratics functions is in $O(kn)$, where $k = n^2$ thus the complexity is $O(2n^3)$. Updating the linear-quadratic function in PLQ2c can be completed in $O(n^2)$ as it is the operation of indexing into a matrix of size n^2. Thus, the complete complexity for adding two unbounded inequality-based bivariate PLQ's is $O(n^2) + O(n^3) + O(n^2) = O(n^3)$.

Corollary 4.3.1. *If the complexity of IPLQ2_eval is $O(n+k)$ then the complexity of IPLQ2_add is $O(n^2)$.*

4.4 Scalar Multiplication Algorithm

Scalar multiplication for the inequality-based PLQ is a straightforward linear-time algorithm in which the linear-quadratic equation for each face in the PLQ model in multiplied by a scalar value. It does not effect any other component of the data structure.

Algorithm 9: IPLQ2_mult Algorithm

```
Input: IPLQ2,$c$
Output: an IPLQ2 that has been scalar multiplied by $c$
begin
    initialization;
    forall each triangulation $t_i \in \mathcal{D}$ in PLQ2 do
        multiply the linear-quadratic equation at index $i$ by the constant $c$;
    end
    Return PLQ2;
end
```

Proposition 4.4.1. *The IPLQ2_mult algorithm runs in $O(n)$.*

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The following complexities were found for each model:

	Operation			
Model	build	eval	add	mult
Tessellation	$O(n\text{log}n)$	$O(nk)$	$O(n^2)$	$O(n)$
Inequality	$O(n^2)$	$O(nk)$	$O(n^3)$	$O(n)$
Voronoi[1]	$O(n\text{log}n)$	$O(n+k)^2$	$O(n^2)$	$O(n)$

Table 5.1.1: Complexity Results for Fundamental Convex Transforms

It was observed empirically that, for functions that contained large amounts of linear data the algorithms tended to perform better than the upper bound. Overall, for linear encoded data, the models perform empirically better than the worst case complexity. Encoding strictly convex functions does present a challenge for this model as little or no simplification or reduction in the size of the input compared to the output can occur, leading to a performance bottleneck.

It was also observed that most methods that generate or permeate the model are output sensitive in terms of the number of faces generated. As previously mentioned, as this is a zero order model, strictly convex data cannot be efficiently encoded. Thus, as the model improves and is able to encode quadratic information, a performance increase may be seen as the number of faces required to encode data over a given region may decrease.

Limitations also exist with the tessellation-based model for strictly convex functions with a large number of parts in a small area. The convex hull algorithm starts to reach a performance barrier when the angles of each triangulation become extremely small, such that trouble is encountered when trying to resolve the location of a point. With the introduction of a model that allows for the encoding of quadratic information, this should no longer be a barrier to performance. A possible way to extend the tessellation-based model to functions

[1]Results for the Voronoi model are conjectured.

[2]Since both the Vertices and the list of points to evaluate can are sorted, this amounts to merging to sorted sequences (n+k) + sorting : n log n + k log k but the vertices can be maintained sorted i.e. sorted in the build. So the cost drops to n+ k log k. If the set of k points is a grid (as is most often the case), it is assumed already sorted along each axis (so trivially sorted in the plane) so the cost is now n+k.

If nothing is sorted, evaluation can still be done in k log n by repeatedly searching in the sorted list of vertices.

with unbounded domains would be to store vertices and half-lines separating the unbounded faces (e.g. each vertex is associated with a list of directions corresponding to the edges of the tessellation going through that vertex). Such information is redundant for vertices in the interior of the domain (since it can be extracted from the other vertices) but is critical for vertices on the boundary of the domain to handle unbounded faces.

Additional work needs to be carried out to complete the implementation for the IPLQ2_eval and IPLQ2_add methods as well as verifying and improving the run time complexity. We know that the addition algorithm when formulated as a map overlay type problem can have a run time complexity of $O(n^2)$, while our method is still bounded by $O(n^3)$ with the bottleneck being the extraction and summation of the linear-quadratic equation for each new face. Future Work will focus on streamlining and improving this functionality to bring the method within the $\Omega(n^2)$ bound. Further work will also be conducted into improving the evaluation method with improved search methods.

Further work will investigate if a PLQ function can be encoded using a Voronoi diagram, which will be computed from the initial Delaunay tessellation. Voronoi diagrams have previously been used to encode a convex hull as a finite set of intersecting half-spaces [3]. A user would define the point value in the model and the algorithm would compute and project a Voronoi diagram into $\mathbb{R}^2$. Encoded with each center would be the value of the function as a linear-quadratic equation. Thus a PLQ function in $\mathbb{R}^3$ is encoded as a matrix containing Voronoi centers $(x_i, y_i) \in \mathcal{V}$ in $\mathbb{R}^2$ associated with the corresponding linear-quadratic equation. Equation (2.1.1) encodes the function value of each part of the PLQ to the boundaries of the polytope for the Voronoi diagram in $\mathbb{R}^2$ containing the Voronoi centers $\mathcal{V}$.

Adding the location of each Voronoi center $(x_i, y_i) \in \mathcal{V}$ with Equation (2.1.2) a PLQ function is encoded into a single matrix as

$$\begin{pmatrix} x_1 & y_1 & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & b_{1,1} & b_{1,2} & c_1 \\ x_2 & y_2 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & b_{2,1} & b_{2,2} & c_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ x_i & y_i & a_{i,1} & a_{i,2} & a_{i,3} & a_{i,4} & b_{i,5} & b_{i,6} & c_i \end{pmatrix}. \tag{5.1.1}$$

This model has desirable characteristics due to the properties of Voronoi diagrams. As previously stated, the Voronoi diagram of a set on n sites in the plane can be computed with a line sweep algorithm in $O(n\log n)$ and with a storage complexity of $O(n)$ [8, 3] with the storage complexity for the complete model being $O(n)$.

The desirability of this model is further enhanced in terms of evaluating the bivariate PLQ over a set of points, $x_i, y_i \in X$. This problem can be treated as a variant of the *post-office problem* [8, 3] with solutions for a single point query in $O(n\log n)$ [3], noting that with a Voronoi diagram, each Voronoi cell is uniquely defined by $x_i, y_i \in \mathcal{V}$, and by definition, a given cell contains all points closer to v_a than to any other center $v_i \in V$.

We feel that this model may provide a good basis for a bivariate PLQ in $\mathbb{R}^3$, but uncertainty exists regarding the addition function. We anticipate that the addition of two Voronoi models may be treated as a map overlay problem and future work will focus on investigation of this function for this possible model. Provided that this is the case, then we will have two possible models to represent bivariate PLQ functions.

As the model presented only represents zero order data, the data structure can become large for strictly convex functions. With the introduction of a first and second order model, improved encoding will occur to reduce the storage complexity. Additional limitations exist in the current model for dense tessellation where the linear-quadratic function value between triangulations is very small, such that they can be joined into a single face to reduce the overall complexity of the model; thus, a clean function needs to be introduced to allow for the accommodation of such functionality.

Questions still exist in terms of the ratio between the input size k and the output size n and further work needs to be done to investigate this relationship. With respect to the code library, continual improvement is needed to add support for degenerate cases in addition to improving test coverage and unit testing. Additionally, support for other fundamental convex transforms needs to be expanded. Work to parallelize methods may be undertaken to improve the run time performance of addition and conversion methods.

Chapter 6

APPENDIX

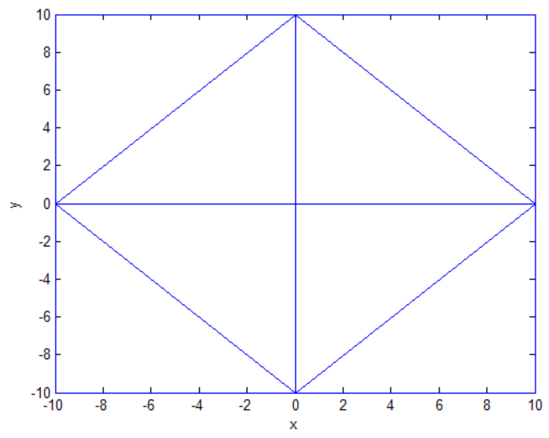
Numerical Examples The following sections include numerous examples of using the specific methods previously introduced. All examples are run in MATLAB R2007b.

6.1 Building a Tessellation-Based PLQ from Pointwise Data

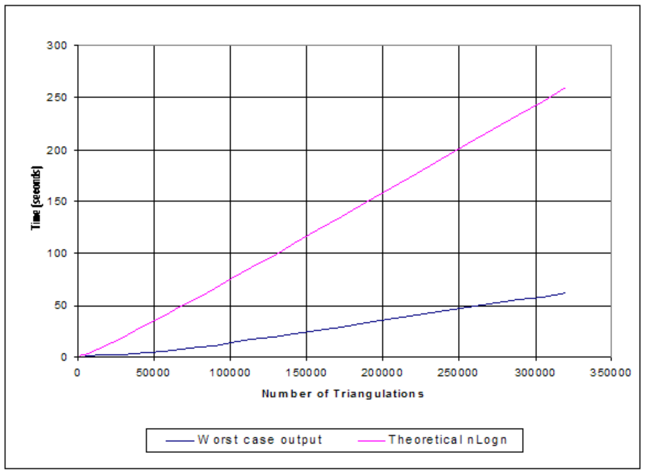
6.1.1 Example

Building a tessellation-based PLQ from pointwise data involves two steps. First, the function data must be formed as two separate elements; the first, X, is a $2xn$ matrix that represents the points in $\mathbb{R}^2$ that the function is defined over. The point sampling does not have to be regular. The second element is a vector $f(X)$ which represents the function value over X. Building the function involves a call to PLQ2_build with X and $f(X)$ being passes as parameters. The function will return a structure the contains two elements. The first element is a matrix X, the vertices for the triangulations that define the PLQ. The second element is a matrix TES that contains the vertex indices for each triangulation and the linear-quadratic function associated with each triangulation. It should be noted that MATLAB uses 1 based indexing for arrays and matrices. A helper method PLQ2_show_tes which takes as an argument a tessellation-based PLQ2, will produce a graph is $\mathbb{R}^2$ that graphically represents that triangulation produced.

```
%define a series of points and the function value
>> [x y] = meshgrid(-10:2:10);
>> x = reshape(x,size(x,1)^2,1);
>> y = reshape(y,size(y,1)^2,1);
>> f = abs(x) + abs(y);
>> X = [x y];
>> plq2 = PLQ2_build(X ,f);

%The structure of the model
>> plq2

plq2 =

    TES: [8x10 double]
      X: [9x2 double]

%The vertexes
>> plq2.X
```



(a) Tessellation in $\mathbb{R}^2$ produced by the build method



(b) Run Times for the PLQ2_build Method

Figure 6.1.1: Build Method Results

```
ans =

   -10   -10
   -10     0
   -10    10
     0   -10
     0     0
     0    10
    10   -10
    10     0
    10    10

%The faces
>> plq2.TES

ans =

     2     4     5     0     0     0     0    -1    -1     0
     1     4     2     0     0     0     0    -1    -1     0
     5     4     8     0     0     0     0     1    -1     0
     8     4     7     0     0     0     0     1    -1     0
     6     2     5     0     0     0     0    -1     1     0
     3     2     6     0     0     0     0    -1     1     0
     5     8     6     0     0     0     0     1     1     0
     6     8     9     0     0     0     0     1     1     0

>> PLQ2_show_tes(plq2);
```

The results of the call to the helper function PLQ2_show_tes can be seen in figure 6.1(a).

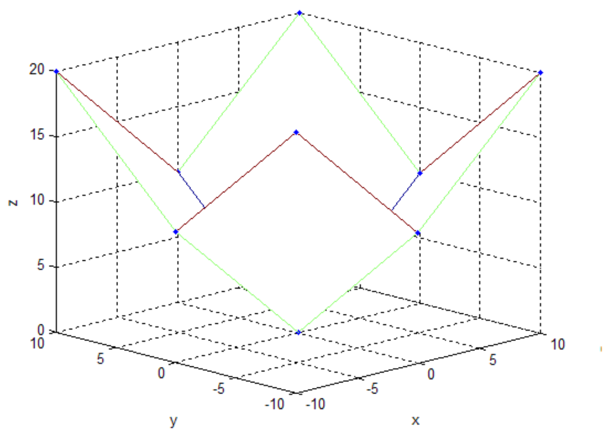


Figure 6.1.2: Evaluation of a tessellation-based PLQ2

6.1.2 Comments

input size	25	441	1681	6561	40401	160801
number of triangulations	32	800	3200	12800	80000	320000
time(seconds)	0.0071	0.1943	0.4179	1.6626	9.8149	61.9158

Table 6.1.1: Complexity Results of the Build Method for a Strictly Convex Function

In looking at the empirical run time complexity from timing results from MATLAB as in Table 6.1.1 and in Figure 6.1(b), the function appears to progress beneath the conjectured run time complexity of $O(n \log n)$. The storage complexity also appears to be in $O(n)$ as the number of triangulations generated increases linearly with the number of input points.

6.2 Evaluating a Bounded PLQ

6.2.1 Example

Evaluating tessellation-based PLQ from pointwise data involves two steps. First, a matrix of the points X in 2 x n over which the function is to be evaluated. Once the points X have been defined, the second step is to invoke the function involves with call to PLQ2_eval with $PLQ2$ and X being passed as parameters. The parameters $PLQ2$ is the model over which X is being evaluated. The function will return a vector of function values for each $x_i \in X$

```
%using the model PLQ2 defined previously
%define a matrix of points to be evaluated
>> ti = -10:10:10;
>> [XI,YI] = meshgrid(ti,ti);
>> xx = reshape(XI,size(XI,1)^2,1);
>> yy = reshape(YI,size(YI,1)^2,1);
>> XX = [xx yy];
%and evaluate
>> z = PLQ2_eval(plq2,XX);
>> z

z =

    20
    10
    20
    10
     0
    10
    20
    10
    20
```

To plot the results graphically, a regular grid can be defined as above and plotted as

```
>> ZI = griddata(xx,yy,z,XI,YI);
>> mesh(XI,YI,ZI), hold
>> g2 = plot3(xx,yy,z,'.'),hold off
Current plot held

g2 =

  160.0221
```

with the results of the plot show in figure 6.1.1.

6.2.2 Comments

The empirical results have previously been presented in Section 3.2 and it was seen that the evaluation is bounded by $O(nk)$

6.3 Adding Two Bounded PLQ's

6.3.1 Example

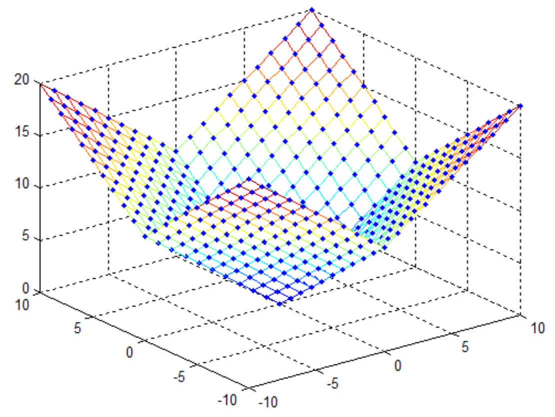
Adding two tessellation-based PLQ's is accomplished with call to PLQ2_add with two PLQ's, PLQa and PLQb being passed as parameters. The function will return a PLQ that is the summation of the two models. The two models do not have to be defined over the same domain.

```
%define model
>> [x y] = meshgrid(-10:5:10);
>> x = reshape(x,size(x,1)^2,1);
>> y = reshape(y,size(y,1)^2,1);
>> f = abs(x) + abs(y);
>> X = [x y];
>> plq2a = PLQ2_build(X ,f);
>> size(plq2a.TES)

ans =

     8    10
%define model 2

>> [x y] = meshgrid(-10:5:10);
>> x = reshape(x,size(x,1)^2,1);
>> y = reshape(y,size(y,1)^2,1);
>> f = x.^2 + y.^2;
>> X = [x y];
>> plq2b = PLQ2_build(X ,f);
>> size(plq2b.TES)

ans =

    32    10

%and add the two models
>>plq2c = PLQ2_add(plq2a,plq2b);
>>size(plq2c.TES)

ans =

    32    10
```

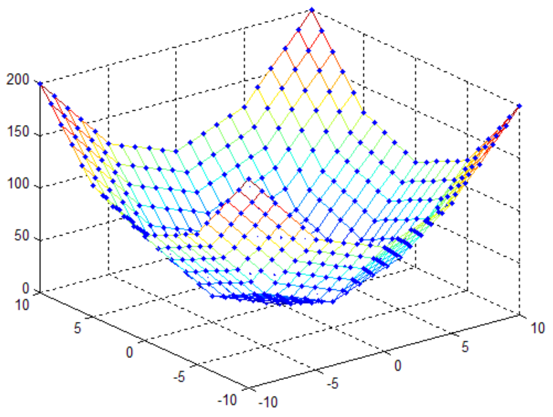
6.3.2 Comments

In this example, two functions are added producing a third. Due to the increasing size of the data structure, the results are show graphically. Figure 6.1(a) and Figure 6.1(b) represent the two functions that are being added together. The results of the summation are shown in Figure 6.1(c).

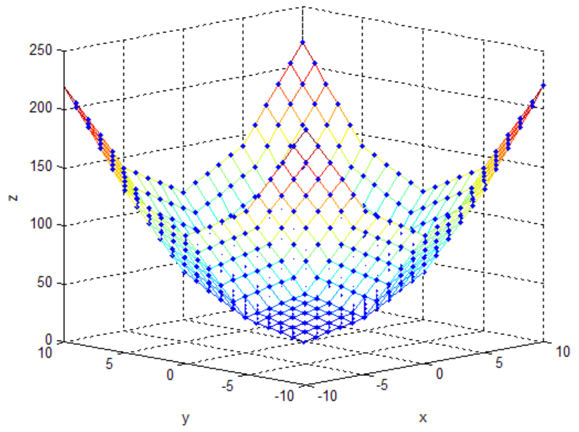
An important observation must be made with the bounded tessellation-based PLQ2_add; if the two models have different domains, the resulting domain will be the intersection of the two models as the model only represents values that are defined. Hence, it is possible to have a reduction in number of faces in the model due to this property. Another observation was that in testing, the bounded tessellation-based PLQ2_add tended to run in $O(n)$ as in Figure 6.1(d) suggesting that the average run time complexity may be less that the theoretical worst case run time complexity of $O(n^2)$.



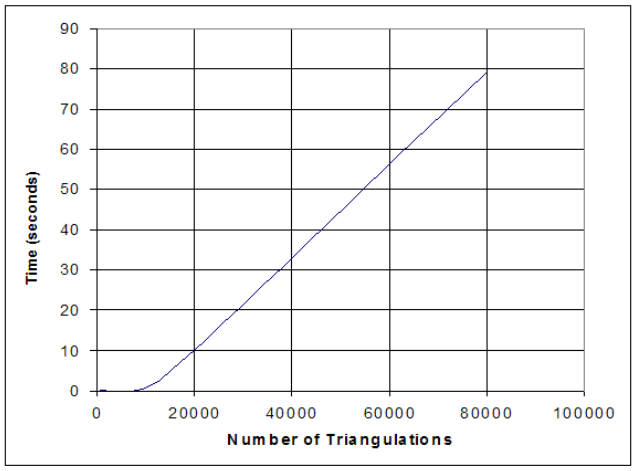
(a) PLQa



(b) PLQb



(c) PLQc: the Results of the Addition of PLQa and PLQb



(d) Run Time for the Addition of Two PLQ Models

Figure 6.3.1: The Addition of Two PLQ Models

6.4 Scalar Multiplication of a tessellation-based PLQ

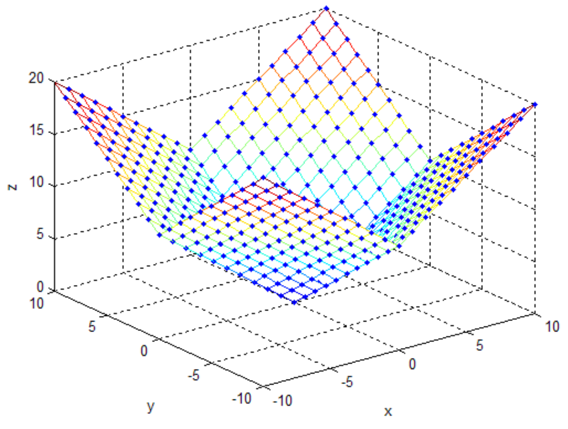
6.4.1 Example

Multiplying a tessellation-based PLQ by a scalar constant is computed by a call to PLQ2_mult with $PLQ2$ and k being passes as parameters where k is the scalar constant. The function will return a tessellation-based PLQ.

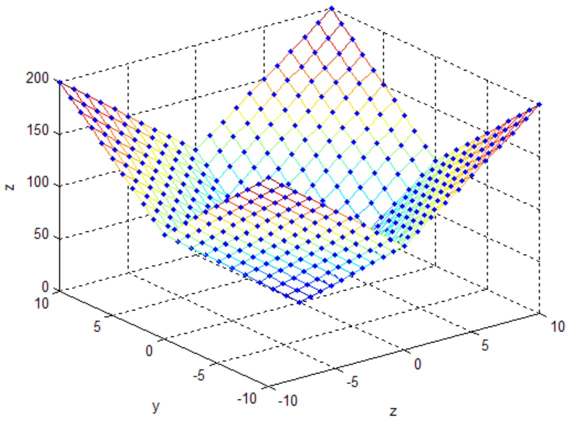
```
>> plq2

plq2 =

    TES: [8x10 double]
      X: [9x2 double]

>> plq2.TES

ans =

     2     4     5     0     0     0     0    -1    -1     0
     1     4     2     0     0     0     0    -1    -1     0
     5     4     8     0     0     0     0     1    -1     0
     8     4     7     0     0     0     0     1    -1     0
     6     2     5     0     0     0     0    -1     1     0
     3     2     6     0     0     0     0    -1     1     0
     5     8     6     0     0     0     0     1     1     0
     6     8     9     0     0     0     0     1     1     0

>> plq2 = PLQ2_mult(plq2,10);
>> plq2.TES

ans =

     2     4     5     0     0     0     0   -10   -10     0
     1     4     2     0     0     0     0   -10   -10     0
     5     4     8     0     0     0     0    10   -10     0
     8     4     7     0     0     0     0    10   -10     0
     6     2     5     0     0     0     0   -10    10     0
     3     2     6     0     0     0     0   -10    10     0
     5     8     6     0     0     0     0    10    10     0
     6     8     9     0     0     0     0    10    10     0
```

6.4.2 Comments

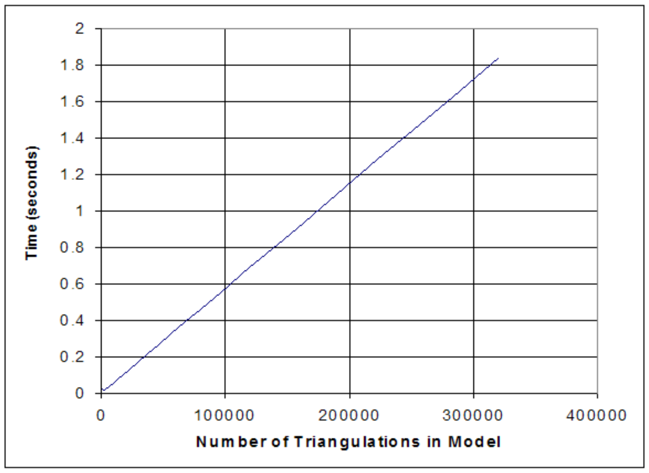
Visually, the results can seen in figure 6.1(a) and 6.1(b) where a function has been multiplied by a constant, $k = 10$. In looking at the empirical run time complexity as in figure 6.1(c) is seen to be linear with an increasing number of triangulations in the model. This supports conjecture 3.4.1 that the algorithm runs in $O(n)$ in terms of the number of triangulation in the model.



(a) PLQ Before Multiplication by a Scalar Constant



(b) PLQ After Multiplication by a Scalar Constant



(c) PLQ2_Mult Run Time Complexity

Figure 6.4.1: PLQ Multiplication

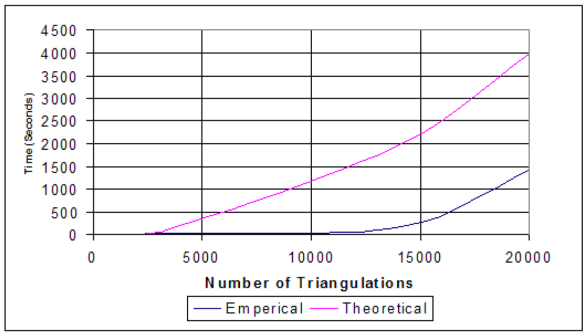


Figure 6.5.1: Run Time Complexity for Model Conversion

6.5 Converting a tessellation-based PLQ to an inequality-based PLQ

6.5.1 Example

Converting from a bounded tessellation-based PLQ model to an inequality-based PLQ model can be done with a single call to the method PLQ2_convert_to_IPLQ2 which will return an inequality-based PLQ. In the example, a simple tessellation-based PLQ is converted to an inequality-based PLQ.

```
>>plq2 =

    TES: [8x10 double]
      X: [9x2 double]

>> iplq2 = PLQ2_convert_to_IPLQ2(plq2);

>> iplq2

iplq2 =

     ineq: [20x3 double]
    faces: {12x2 cell}
```

It should be observed that the number of faces in the model has increased from 8 to 12 for this example. This is due to the fact that the tessellation-based model is a bounded model and when converted to an inequality-based model, the domain is expanded such that the entire $\mathbb{R}^2$ space is encoded; hence the need for additional faces.

6.5.2 Comments

A simulation was run over a series of models with an increasing number of tessellations up to 20,000 faces. The model that was used was a strictly convex function such that it would generate a worst case scenario. The results are seen in figure 6.5 which shows processor time plotted against the size of the model. Additionally, the theoretical $O(n^2)$ is shown as a comparison. It appears that the empirical run time complexity supports Proposition 4.1.1 such that the worst case run time complexity is $O(n^2)$.

6.6 Evaluating an inequality-based PLQ

6.6.1 Example

Evaluating tessellation-based IPLQ from pointwise data involves two steps. First, a matrix of the points X in 2 x n over which the function is to be evaluated. Once the points X have been defined, the second step is to invoke the function involves with call to IPLQ2_eval with $IPLQ2$ and X being passes as parameters. The parameter $ILQ2$ is the model over which X is being evaluated. The function will return a vector of function values for each $x_i \in X$

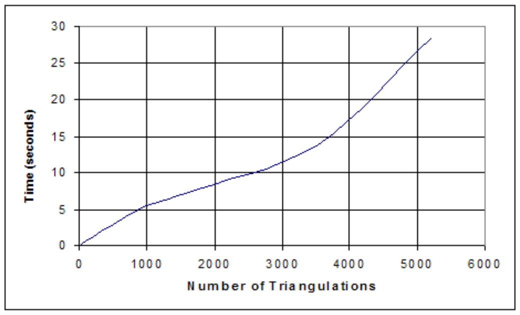
```
%using the model IPLQ2 defined previously
>> ti = -10:10:10;
>> [XI,YI] = meshgrid(ti,ti);
>> xx = reshape(XI,size(XI,1)^2,1);
>> yy = reshape(YI,size(YI,1)^2,1);
>> XX = [xx yy];
%and evaluate
>> z = IPLQ2_eval(iplq2,XX);
>> z

z =

    20
    10
    20
    10
     0
    10
    20
    10
    20
```

6.6.2 Comments

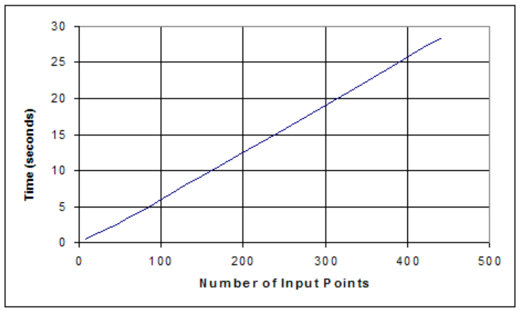
Two separate tests were run, one looking at the empirical run time complexity with a varying number of input points over a fixed number of tessellation. The second evaluated the run time complexity with a fixed number of points over a varying number of tessellations. Both exhibited linear behavior over the scope of the test. A third test was run looking at varying both the number of points k and the number of tessellations k. Empirically, the inequality evaluation performs in a linear fashion over nk as seen in figure 6.1(c) which supports conjecture 4.2.1

6.7 Additional Methods

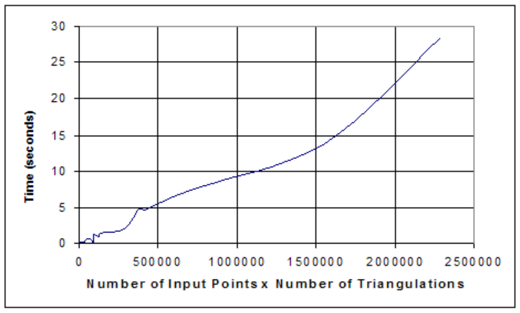
Usage of the additional methods supporting multiplication and addition for the inequality model follow the examples for the tessellation-based model.



(a) Evaluation with a Constant Number of Points



(b) Evaluation with a Constant Number of Faces



(c) IPLQ2_eval Run Time Complexity Over nk

Figure 6.6.1: IPLQ Evaluation

Bibliography

[1] *Matlab r2007b*, 2008. http://www.mathworks.com/.

[2] *Scilab*, 2008. http://www.scilab.org/.

[3] F. Aurenhammer, *Voronoi diagrams - a survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, *The quickhull algorithm for convex hulls*, ACM Trans. Math. Softw., 22 (1996), pp. 469–483.

[5] L. Becker, A. Giesen, K. Hinrichs, and J. Vahrenhold, *Algorithms for performing polygonal map overlay and spatial join on massive data sets*, "Lecture Notes in Computer Science", "1651" (1999), pp. 270–285.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge; United Kindgdom, 2006, c2004.

[7] T. M. Chan, *Optimal output-sensitive convex hull algorithms in two and three dimensions*, Discrete Comput. Geom., 16 (1996), pp. 361–368. Eleventh Annual Symposium on Computational Geometry (Vancouver, BC, 1995).

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag Berlin Heidelberg, New York; New York, second ed., c2000, c1997.

[9] M. Khan, *Binary search for numeric vector*, 2006. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=11287.

[10] A. J. King and D. L. Jensen, *Linear-quadratic efficient frontiers for portfolio optimization*, Applied Stochastic Models and Data Analysis, 8 (1992), pp. 195–207. http://dx.doi.org/10.1002/asm.3150080309.

[11] Y. Lucet, H. H. Bauschke, and M. Trienis, *The piecewise linear-quadratic model for computational convex analysis*, Comput. Optim. Appl., (2007). Online publication.

[12] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag Berlin Heidelberg, New York; New York, c1985.

[13] A. Rantzer and M. Johansson, *Piecewise linear quadratic optimal control*, 1997. citeseer.ist.psu.edu/rantzer97piecewise.html.