UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Development of a Tropical Algebraic Geometry package in the Haskell programming language

Autor:

Zhapa Camacho Fernando Patricio

Tutor:

Ph.D. Antón Castro Francesc

Urcuquí, julio de 2020

Urcuquí, 6 de marzo de 2020

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2020-00007-AD

En la ciudad de San Miguel de Urcuquí, Provincia de Imbabura, a los 6 días del mes de marzo de 2020, a las 09:30 horas, en el Aula CHA-01 de la Universidad de Investigación de Tecnología Experimental Yachay y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.
Miembro No Tutor	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Tutor	Dr. ANTÓN CASTRO , FRANCESC , Ph.D.

Se presenta el(la) señor(ita) estudiante **ZHAPA CAMACHO, FERNANDO PATRICIO**, con cédula de identidad No. **1104136138**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, con el objeto de rendir la sustentación de su trabajo de titulación denominado: **DEVELOPMENT OF A TROPICAL ALGEBRAIC GEOMETRY PACKAGE IN THE HASKELL PROGRAMMING LANGUAGE"** , previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. ANTÓN CASTRO , FRANCESC , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.	9,5
Tutor	Dr. ANTÓN CASTRO , FRANCESC , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.	10,0

Lo que da un promedio de: **9.8 (Nueve punto Ocho)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

ZHAPA CAMACHO, FERNANDO PATRICIO
Estudiante

Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.
Presidente Tribunal de Defensa

Dr. ANTÓN CASTRO , FRANCESC , Ph.D.
Tutor

Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
Miembro No Tutor

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

Autoría

Yo, **Fernando Patricio Zhapa Camacho**, con cédula de identidad 1104136138, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así como, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor(a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, julio 2020.

Fernando Patricio Zhapa Camacho
CI: 1104136138

Autorización de publicación

Yo, **Fernando Patricio Zhapa Camacho**, con cédula de identidad 1104136138, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además, que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, julio 2020.

Fernando Patricio Zhapa Camacho
CI: 1104136138

Dedication

To my parents Carmita and Patricio, for teaching me the value of real freedom and responsibility. Their unconditional support and love had led me to this moment and place.

To Andrés, who no matter how old he is, he will always be my little brother.

To Evelyn, who is a special part of my life.

To Atuk, my son, that piece of life that has remembered me how it is to be a child and, at the same time, has encouraged me to grow.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Francesc Antón Castro, for introducing me to the field of computational algebraic geometry and accepting to supervise this thesis project. The development of this work could not be possible without his continuous support, help and availability and patience, as well as his guidance in the development of the algorithms and encouragement of working in functional programming languages. I would like to thank the members of my thesis committee, Drs. Juan Mayorga and Fredy Cuenca, for their valuable comments and suggestions.

I would like to express my most sincere gratitude to my colleague Anthony Ramos for his continuous support, comments and help during the development of this project. His interest in my project was invaluable. Thanks to him, I knew about the Tropical Differential Algebra Workshop, in which I could present my thesis. I would like to acknowledge my colleague Darwin Tallana for his help in explanations of the mathematical concepts that were hard for me to understand.

I would like to thank Drs. Alex Fink and Zeinab Toghani for helping me to attend the Tropical Differential Algebra Workshop and present my thesis project. It was an enriching experience that allowed me to understand tropical geometry better. I would like to thank the people in the workshop who supported my project and helped me with all the doubts that I had about tropical geometry. Mainly, I would like to express my gratitude to Dr. Yue Ren for his valuable comments and suggestions about my thesis and his help in the Polymake software that was used for comparison.

I would like to express my gratitude to Dr. Hiromi Ishii for his help in functional programming and his availability to answer my questions. He explained to me the importance of dependent typing and his work on the `computational-algebra` package was the base for my package in terms of the polynomial data type.

Abstract

Tropical geometry is a growing area in mathematics because it has many applications in optimization, enumerative algebra, and combinatorics. On the other hand, functional programming has grown its popularity in recent years because its inherent properties like pattern matching, algebraic data types, parametric polymorphism make it more suitable than imperative programming for some purposes. Thus, although some packages exist for tropical geometry computations, there is not any developed in a functional paradigm.
The main objective of this work is to develop a tropical geometry package implemented in a functional paradigm by choosing the Haskell programming language. We start by describing the necessary mathematical background that we used to develop the package. This background includes topics in tropical numbers, algebraic geometry, and polyhedral geometry. After that, we describe the computational aspects such as the functional paradigm properties, data types, functions and testing. Furthermore, we provide explanations of the main algorithms used for the computation of tropical hypersurfaces, which is the main result obtained in this project. Finally, we observed that the computation of n-dimensional convex hull is required, and we proposed it as future work.

Keywords: tropical geometry, functional programming, tropical hypersurfaces

Resumen

La geometría tropical es un área creciente en matemática porque tiene muchas aplicaciones en optimización, álgebra enumerativa y combinatoria. Por otro lado, la programación funcional ha aumentado su popularidad en los últimos años debido a que sus propiedades como la búsqueda de patrones, los tipos de datos algebraicos y el polimorfismo paramétrico, lo hacen más adecuado que la programación imperativa para ciertos fines. Por lo tanto, a pesar del hecho de que existen algunas liberías para cálculos en geometría tropical, no hay ninguna desarrollada bajo un paradigma funcional. El objetivo principal de este trabajo es desarrollar un paquete de geometría tropical implementado en un paradigma funcional eligiendo el lenguaje de programación Haskell. Para esto, comenzamos describiendo el fundamento matemático necesario que utilizamos para desarrollar la librería. Entre los temas revisados se incluyen: números tropicales, geometría algebraica y geometría poliédrica. Asimismo, describimos los aspectos computacionales, como las propiedades del paradigma funcional, los tipos de datos, las funciones y los casos de prueba. Además, proporcionamos la explicacion de los principales algoritmos utilizados para el cálculo de las hipersuperficies tropicales, el cual es el resultado principal obtenido en este proyecto. Finalmente, mencionamos el posible trabajo futuro, en el cual se incluye principalemnte al cálculo del envolvente convexo n-dimensional.

Palabras clave: geometría tropical, programación funcional, hipersuperficies tropicales.

Índice general

Dedication **i**

Acknowledgments **iii**

Abstract **v**

Resumen **vii**

Índice general **ix**

Índice de cuadros **xi**

Índice de figuras **xiii**

1. Introduction **1**

1.1. Problem statement 1

1.2. Motivation 2

1.3. Objectives 2

1.4. Outline of research and development 3

2. Theoretical framework **5**

2.1. Mathematical background 5

2.1.1. Tropical Arithmetic 5

2.1.2. Tropical Polynomials 8

2.1.3. Tropical Algebraic Geometry 12

2.1.4. Polyhedral geometry 18

2.2. Computational background 20

2.2.1. Computational Geometry 20

2.2.2. Functional Programming 22

2.3. Software Engineering Background 23

3. Methodology **25**
3.1. The Data Model 25
3.1.1. Arithmetic data types 25
3.1.2. Algebraic Data Types 26
3.1.3. Geometric Data Types 26
3.2. Algorithms 27
3.2.1. Convex Hull algorithm 27
3.2.2. Subdivision algorithm 32
3.2.3. Tropical Hypersurface 33
3.3. Software Engineering Aspects 36
3.3.1. Dependencies 36
3.3.2. File structure of the library 37
3.3.3. Installation 37
3.3.4. Tutorial 38

4. Results **41**
4.1. Test set 41
4.2. Comparison with other software 45

5. Discussion and Conclusions **47**
5.1. Results analysis 47
5.2. Conclusion 48
5.3. Further work 48

Appendices **50**
References 63

Bibliografía **65**

Índice de cuadros

4.1. Vertices and rays for f_1 42
4.2. Vertices and rays for f_2 42
4.3. Vertices and rays for f_3 43
4.4. Vertices and rays for f_4 44
4.5. Vertices and rays for f_5 44
4.6. Comparison between Polymake and our package for polynomials $f_1, ..., f_8$. . 45

Índice de figuras

2.1. Graph of the tropical polynomial $p(x) = 5 \oplus 1x \oplus (-1)x^2 = \text{mín}\{5, x + 1, 2x - 1\}$ and its roots. 9
2.2. Curve of the tropical line $trop(l(x, y)) = x \oplus y \oplus a$ 17
2.3. Representation of a tropical variety generated by two tropical lines 18
2.4. Polyhedral complex and polyhedral fan 19
2.5. 1-skeleton of a polyhedral complex of dimension 2. 20
2.6. Example of a convex hull in $\mathbb{R}^2$ 21
2.7. Lower face of polyhedra 21

3.1. Representation of left and right turns 28
3.2. Representation of Graham Scan algorithm. 29
3.3. Four points forming a tetrahedron. 29
3.4. Generating new facets from the horizon to a new point 30
3.5. Representation of a point, p_{i+1}, that is coplanar to a face f. The new face is an extension of f 30
3.6. Twin edges that connect two facets. 31
3.7. Representation of a subdivision from a convex hull of dimension 3 into a subdivided polygon of dimension 2. 33
3.8. Representation of the inner normal fan of a triangle. 34
3.9. Lower faces of the Newton polytope of g 35
3.10. Representation of the tropical hypersurface of g from the subdivision of a triangle. 36
3.11. File structure of the package 37

4.1. Tropical hypersurface for f_1, $trop(f_1)$ 42
4.2. Tropical hypersurface for f_2, $trop(f_2)$ 43
4.3. Tropical hypersurface for f_3, $trop(f_3)$ 43
4.4. Tropical hypersurface for f_4, $trop(f_4)$ 44
4.5. Tropical hypersurface for f_5, $trop(f_5)$ 44

4.6. Comparison between Polymake and our package for polynomials $f_1, ..., f_8$. 46

Capítulo 1

Introduction

1.1. Problem statement

Tropical arithmetic is defined over the semi-ring[1] $(\mathbb{R} \cup \{\infty\}, \text{mín}, +)$. Thus, summing two tropical numbers means computing their minimum and multiplying them is equivalent to compute their sum [1]. This area is relatively new and has applications in problems such as shortest-path, assignment, linear optimization and topics in algebraic geometry. Tropical arithmetic is a well-known field compared to tropical algebra or tropical geometry due to its mathematical simplicity, which is given by the fact that tropical arithmetic is limited to numbers and matrices, leaving outside most of the aspects of the tropical world.

The disciplines built on tropical arithmetic are tropical algebra and tropical geometry. Research on these fields has been growing in the last years since the process of tropicalizing classical algebra problems, when the tropicalization exists, makes them easier to deal with [2]. Therefore, computational tools are starting to appear. However, there is not any package implemented in a functional programming language.

Several packages and libraries have been developed until now. *Sparse Tropical Algebra* [3] is a package developed for the R language, which subscribes to the imperative paradigm. This package consists of an implementation of tropical linear algebra operations. BTAS [4] is analogous of the BLAS library for tropical algebra. That is, BTAS is a parallel vector library to perform matrix computations using CUDA. For the testing of the library, they use the Floyd-Warshall algorithm, which is based on matrix multiplications to find the shortest path in a graph. Gfan [5] goes further: it is a package for computing Gröbner fans and tropical varieties written in C and C++. This package is more related to tropical algebra and tropical geometry and has been embedded into Singular [6]. Another related

[1] A formal definition for semi-rings is given in Chapter 2 in Definition 3.

package is Polymake [7], which computes hypersurfaces in n dimensions using the Gfan package.

The purpose of this research is to develop a package of tropical arithmetic, tropical algebra and tropical geometry for scientists that need a computing tool in this area. The language for the development is the functional programming language Haskell since it is easy to represent mathematical concepts, and function composition is the base of the flow of computations.

1.2. Motivation

In computer science, there exist two big groups of programming languages. The first group consists of those languages that come from the development of the Turing machine, and its computations are based on sequences of instructions: imperative languages. The second group is composed of those that come from the development of lambda calculus, and its computations are based on function composition: functional languages. Even though imperative languages are the most used in industry and academy, functional languages, especially Haskell, have been growing their popularity and are being used by many companies and institutions around the world [8]. To this end, many packages have started to appear for programmers in different areas: games, GUI, data analysis, artificial intelligence and computer algebra.

The latter hides a great feature; it turns out that the Haskell programming language is based on category theory, which has eased the binding of mathematical concepts. One example is the package `algebra` [9], which implements algebraic structures as data types such as fields, groups or rings. From this package, some other packages of computer algebra have been developed. One example is the `computational-algebra` package [10] that is useful to compute Gröbner basis.

Tropical geometry is a growing area in mathematics. Computations are needed to be done in some aspects, and packages are starting to appear. However, in functional languages, there is not any package of tropical geometry. Then, the motivation of this project is to start a package of tropical geometry, which is a raising area in mathematics, in a functional language like Haskell, which is a raising programming language in computer science.

1.3. Objectives

The objectives of the package developed in this project are:

- Implement the necessary data types to deal with tropical numbers, matrices and polynomials.
- Implement the necessary functions and class instances to make the data types behave properly in terms of tropical operations.
- Implement an algorithm to compute the tropical hypersurface of a polynomial.

1.4. Outline of research and development

In Chapter 2, we present the necessary theoretical aspects in terms of mathematics and computer science. We go through concepts in algebra like polynomials rings or fields with a valuation, concepts in algebraic geometry such as variety and ideal. Also, we review some aspects of polyhedral geometry like polyhedral complex, cells, k-skeleton, k-face, which are essential to deal with tropical polynomials and the tropicalization from the Laurent polynomials.

The other part of the research, which is the computational part, is presented in Chapter 2 as well. We start by introducing what functional programming is and why we use the Haskell programming language for this package. Since the final product of this research is a package, we introduce some software engineering concepts like Test-Driven Development (TDD) and package versioning.

Then, in Chapter 3, we first present the Algebraic Data Types used in the package. We present the main implemented algorithms in order to compute the hypersurfaces of a tropical polynomial: convex hull in 2 dimensions, convex hull in 3 dimensions, subdivision and tropical hypersurface of a polynomial. Finally, we present the libraries of Haskell used for the testing, the tools used for documentation, and where the package will be hosted.

In Chapter 4, we present the test examples performed in the package. We show the results and the corresponding figures to help to a better understanding.

Finally, in Chapter 5, we refer to the limitations of the package and what further research and development should focus on. We also remark the advantages of using functional programming features like ADTs, dependent typing, laziness and immutability.

Capítulo 2

Theoretical framework

In this chapter, we introduce the necessary concepts. The chapter starts with the necessary mathematical background. Then, we introduce the computational background, and then we finish with some concepts about software engineering that we used in the implementation of the code.

2.1. Mathematical background

In this section, we introduce the mathematical concepts that will be covered in the package. As tropical algebra is a new area, some definitions are still under study using classical algebra as a base to define properties for tropical algebra.

2.1.1. Tropical Arithmetic

Definition 1 (Binary or internal operation [11]) *A binary or internal operation $\odot$ on a set $\mathcal{A}$ is the following mapping:*

$$\odot \colon \mathcal{A} \times \mathcal{A} \to \mathcal{A}$$
$$(x, y) \mapsto x \odot y$$

Therefore, for each pair of elements $a, b \in \mathcal{A}$, it corresponds a unique element $a \odot b \in \mathcal{A}$

Definition 2 (Monoid [12]) *A monoid $(\mathcal{M}, \odot)$ is a nonempty set $\mathcal{M}$ with a binary operation $\odot$ that fulfills the following axioms:*

- *(Associativity)* $\forall m, n, p \in \mathcal{M} : m \odot (n \odot p) = (m \odot n) \odot p$
- *(Identity element)* $\exists e \in \mathcal{M}, \forall m \in \mathcal{M} : m \odot e = e \odot m = m$

In a monoid $(\mathcal{M}, *)$, if the operation $*$ commutes (i.e, $\forall a, b \in \mathcal{M}, a * b = b * a$), then $\mathcal{M}$ is called a **commutative monoid**.

Definition 3 (Semiring [12]) *A semiring $(\mathcal{R}, \oplus, \otimes)$ is a nonempty set $\mathcal{R}$ on which the additive and multiplicative operations are defined such that: (1) $(\mathcal{R}, \oplus)$ is a commutative monoid with identity element 0, (2) $(\mathcal{R}, \otimes)$ is a monoid with identity element 1, (3) multiplication distributes over addition, and (4) the neutral element 0 is absorbing (i.e, $0 \times a = a \times 0 = 0, \forall a \in \mathcal{R}$).*

Tropical arithmetic is defined over the semiring $(\mathbb{R} \cup \{\infty\}, \text{mín}, +)$. Thus, the first internal operation corresponds to compute the minimum and the second internal operation is addition [1]. Here,

$$\forall a \in \mathbb{R} : \text{mín}(a, \infty) = a$$

$$\forall a \in \mathbb{R} : a + \infty = \infty$$

For all $a, b \in \mathbb{R} \cup \{\infty\}$, we denote $a \oplus b = \text{mín}\{a, b\}$ and $a \otimes b = a + b$. We can see that the tropical semiring meets the four properties of a semiring since:

1. $(\mathbb{R} \cup \{\infty\}, \otimes)$ is a commutative monoid with identity element ∞:
   - (*Associativity*) For all $a, b, c \in \mathbb{R} \cup \{\infty\}$,

$$\begin{aligned}(a \oplus b) \oplus c &= \text{mín}(\text{mín}(a, b), c) \\ &= \text{mín}(a, b, c) \\ &= \text{mín}(a, \text{mín}(b, c)) \\ &= a \oplus (b \oplus c).\end{aligned} \tag{2.1}$$

   - (*Commutativity*) For all $a, b \in \mathbb{R} \cup \{\infty\}$,

$$\begin{aligned}a \oplus b &= \text{mín}(a, b) \\ &= \text{mín}(b, a) \\ &= b \oplus a.\end{aligned} \tag{2.2}$$

- (*Identity*) For all $a \in \mathbb{R} \cup \{\infty\}$, we have the identity element ∞, such that

$$\begin{aligned} a \oplus \infty &= \text{mín}(a, \infty) \\ &= a. \end{aligned} \tag{2.3}$$

2. $(\mathbb{R} \cup \{\infty\}, \otimes)$ is a monoid with identity element 0:

- (*Associativity*) For all $a, b, c \in \mathbb{R} \cup \{\infty\}$,

$$\begin{aligned} (a \otimes b) \otimes c &= (a + b) + c \\ &= a + b + c \\ &= a + (b + c) \\ &= a \otimes (b \otimes c). \end{aligned} \tag{2.4}$$

- (*Commutativity*) For all $a, b \in \mathbb{R} \cup \{\infty\}$,

$$\begin{aligned} a \otimes b &= a + b \\ &= b + a \\ &= b \otimes a. \end{aligned} \tag{2.5}$$

- (*Identity*) For all $a \in \mathbb{R} \cup \{\infty\}$, we have the identity element 0, such that

$$\begin{aligned} a \otimes 0 &= a + 0 \\ &= a. \end{aligned} \tag{2.6}$$

3. Multiplication distributes over addition. For all $a, b, c \in \mathbb{R} \cup \{\infty\}$,

$$\begin{aligned} a \otimes (b \oplus c) &= a + \text{mín}(b, c) \\ &= \text{mín}(a + b, a + c) \\ &= (a \otimes b) \oplus (a \otimes c). \end{aligned} \tag{2.7}$$

4. The neutral element ∞ is absorbing: $\infty \otimes a = a \otimes \infty = a + \infty = \infty, \forall a \in \mathbb{R}$.

2.1.2. Tropical Polynomials

Based on classical polynomials, tropical polynomials in one variable have the form $p\colon \mathbb{R} \cup \{\infty\} \to \mathbb{R} \cup \{\infty\}$ given by

$$p(x) = a_0 \oplus a_1 x \oplus a_2 x^2 \oplus \cdots \oplus a_n x^n \tag{2.8}$$

where each term $a_i x^i$ means the tropical product $a_i \otimes x^i$. Thus, $p(x)$ is equivalent in classical algebra to

$$\text{mín}\{a_0, x + a_1, 2x + a_2, \ldots, nx + a_n\}, \tag{2.9}$$

which is the minimum of n linear functions.

For the sake of illustration, let us represent the quadratic polynomial:

$$p(x) = a \oplus bx \oplus cx^2 = \text{mín}\{a, b + x, c + 2x\} \tag{2.10}$$

The points where the function has sharp edges anymore are considered as the roots of the polynomial. In [1], it is shown that if $b - a \leq c - b$, then the roots of the quadratic polynomial are given by the set:

$$V(p) = \{a - b, b - c\}. \tag{2.11}$$

Example 1 *Consider the following tropical quadratic polynomial:*

$$p(x) = 5 \oplus 1x \oplus (-1)x^2 = \text{mín}\{5, x + 1, 2x - 1\}$$

In this polynomial $a = 5$, $b = 1$, $c = -1$. It can be seen that $b - a = -4$ and $c - b = -2$. Therefore, we can say, recalling Equation 2.11 that the roots are:

$$V(p) = \{2, 4\}$$

In Figure 2.1 we can see the representation of this example.

In general, the set of roots of a polynomial p is called the **hypersurface** of p and is denoted as $V(p)$. When we deal with polynomials with a higher number of variables, the computation of the hypersurface is not as simple as in the case of polynomials in one

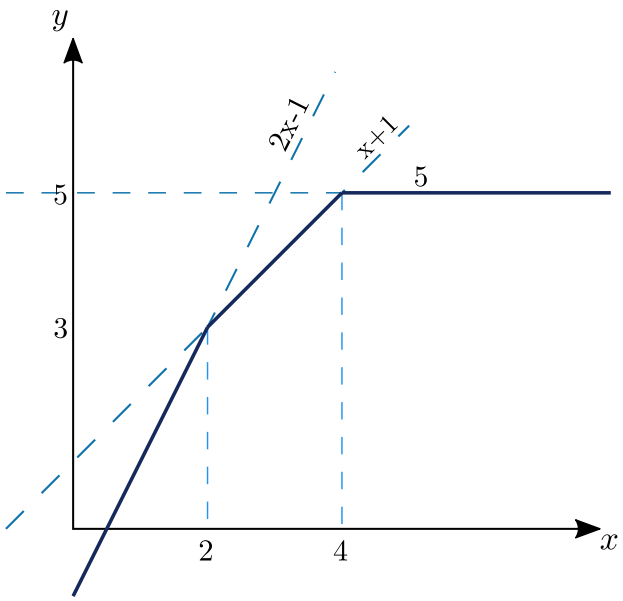


Figura 2.1: Graph of the tropical polynomial $p(x) = 5 \oplus 1x \oplus (-1)x^2 = \min\{5, x+1, 2x-1\}$ and its roots.

variable. To introduce the concept of tropical hypersurface more formally, let us define two previous concepts: field and ring.

Definition 4 (Field [13]) *A field $(\mathcal{K}, \oplus, \otimes)$ is a nonempty set $\mathcal{K}$ and two binary operations $\oplus$ and $\otimes$ defined on $\mathcal{K}$ for which the following conditions are satisfied:*

i) (Associativity) For all $a, b, c \in \mathcal{K}$,

$$(a \oplus b) \oplus c = a \oplus (b \oplus c),$$

$$(a \otimes b) \otimes c = a \otimes (b \otimes c).$$

ii) (Commutativity) For all $a, b \in \mathcal{K}$,

$$a \oplus b = b \oplus a,$$

$$a \otimes b = b \otimes a.$$

iii) (Distributivity of $\otimes$ over $\oplus$) For all $a, b, c \in \mathcal{K}$,

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

iv) (Identities) There are $e, \tilde{e} \in \mathcal{K}$ *such that* $a \oplus e = a \otimes \tilde{e} = a, \forall a \in \mathcal{K}$.

v) (Additive inverses) Given $a \in \mathcal{K}$*, there is* $b \in \mathcal{K}$ *such that* $a \oplus b = e$.

vi) (Multiplicative inverses) Given $a \in \mathcal{K}$*, there is* $b \in \mathcal{K}$ *such that* $a \otimes b = \tilde{e}$.

Definition 5 (Ring [13]) *A ring* $(\mathcal{R}, \oplus, \otimes)$ *is a nonempty set* $\mathcal{R}$ *and two binary operations* $\oplus$ *and* $\otimes$ *defined on* $\mathcal{R}$ *for which the following conditions are satisfied:*

i) (Associativity) For all $a, b, c \in \mathcal{R}$,

$$(a \oplus b) \oplus c = a \oplus (b \oplus c),$$

$$(a \otimes b) \otimes c = a \otimes (b \otimes c).$$

ii) (Commutativity) For all $a, b \in \mathcal{R}$,

$$a \oplus b = b \oplus a,$$

$$a \otimes b = b \otimes a.$$

iii) (Distributivity of $\otimes$ *over* $\oplus$*) For all* $a, b, c \in \mathcal{R}$,

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

iv) (Identities) There are $e, \tilde{e} \in \mathcal{R}$ *such that* $a \oplus e = a \otimes \tilde{e} = a, \forall a \in \mathcal{R}$.

v) (Additive inverses) Given $a \in \mathcal{R}$*, there is* $b \in \mathcal{R}$ *such that* $a \oplus b = e$.

If a ring $\mathcal{R}$ happens to be commutative, then it is called a **commutative ring**.

The set of regular polynomials forms a ring. It is important to remark that the set of regular polynomials does not form a field because the condition of the multiplicative inverse is missing, e.g, the multiplicative inverse of the polynomial $f(x) = x$ is $\frac{1}{x}$, which is not a polynomial. We denote a polynomial ring as $\mathcal{K}[x_1, ..., x_n]$ where $\mathcal{K}$ is the field where the coefficients of the monomials belong and $x_1, ..., x_n$ are the variables considered in the polynomial.

Definition 6 (Tropical hypersurface [1]) *Let us $f \in \mathcal{K}[x_1, \ldots, x_n]$ be a regular polynomial. The tropical hypersurface $trop(V(f))$ is the set*

$$trop(V(f)) = \{a \in \mathcal{K}^n : \text{ such that the minimum in } trop(f)(a) \text{ is achieved at least twice.}\} \tag{2.12}$$

Example 2 *For this example, we use the concept of valuation that is introduced in the next section. Consider the polynomial $f \in \mathbb{Q}[x]$, $f = 32 + 2x + \frac{1}{2}x^2$. Its tropicalization using the 2-adic valuation is $trop(f) = 5 \oplus 1x \oplus (-1)x^2$ which can be expressed as in Example 1:*

$$trop(f)(x) = \text{mín}\{5, 1 + x, -1 + 2x\}$$

. Notice that in Example 1 the polynomial p is the tropicalization of f in this example.

In Example 1 we say that the hypersurface of p is the set $\{2, 4\}$. According to the Definition 6, the set $trop(V(f)) = \{2, 4\}$ corresponds to the points in which the function $trop(f)$ gets its minimum at least twice. Let us see how this happens:

$$\begin{aligned} trop(f)(2) &= \text{mín}\{5, 1 + 2, -1 + 2 \cdot 2\} \\ &= \text{mín}\{5, 3, 3\} \\ &= 3 \end{aligned}$$

$$\begin{aligned} trop(f)(4) &= \text{mín}\{5, 1 + 4, -1 + 2 \cdot 4\} \\ &= \text{mín}\{5, 5, 8\} \\ &= 5 \end{aligned}$$

We can see that in $trop(f)(2)$, the minimum is 3 and is achieved in two of the three linear functions. The same thing happens in $trop(f)(5)$ where the minimum is 5. If we test other values, we can see that the minimum is only achieved one time, therefore they are not roots, e.g:

$$\begin{aligned} trop(f)(0) &= \text{mín}\{5, 1+0, -1+2\cdot 0\} \\ &= \text{mín}\{5, 1, -1\} \\ &= -1 \end{aligned}$$

2.1.3. Tropical Algebraic Geometry

We recall the definition in [13] of a variety in classical algebraic geometry:

Let $\mathcal{K}$ be a field and $f_1, \ldots, f_s$ be polynomials in $\mathcal{K}[x_1, \ldots, x_n]$ such that $f_i : \mathcal{K} \to \mathcal{K}$, then the set

$$\mathbf{V}(f_1, \ldots, f_s) = \left\{(a_1, \ldots, a_n) \in \mathcal{K}^n : f_i(a_1, \ldots, a_n) = 0, \text{ for all } 1 \leq i \leq s\right\}$$

is called the variety defined by $f_1, \ldots, f_s$.

Thus, a variety is the set of points on which all the polynomials of the system vanish. In general, s should be less than n to ensure a nonempty variety. Following this definition, tropical varieties are formed through a process of tropicalization of a classical variety. In [1], it is shown how to tropicalize a variety. For this purpose, some previous concepts are necessary. The concepts described further in this subsection are taken from [1] unless another reference is explicitly referred.

Definition 7 (Totally ordered set [14]) *A set $\mathcal{S}$ is said to be totally ordered if there is a binary relation $\leq$, called total order, that fulfills the following conditions:*

i) (Transitivity) If there exist elements $a, b, c \in \mathcal{S}$ such that $a \leq b$ and $b \leq c$, then $a \leq c$.

ii) (Antisymmetry) If there exist $a, b \in \mathcal{S}$ such that $a \leq b$ and $b \leq a$, then $a = b$.

iii) (Connexity) For all $a, b \in \mathcal{S}$, $a \leq b$ or $b \leq a$

Remark 1 *In Definition 7, if only the first condition is fulfilled, $\mathcal{S}$ is called a **preordered set**. If the first and second conditions are fulfilled $\mathcal{S}$ is called a **partially ordered set**.*

Remark 2 *A **totally ordered group** is a group equipped with a total order.*

Definition 8 (Valuation [15]) *Let us consider a ring $\mathcal{R}$ (not neccessarily commutative) and Γ, a totally ordered group. We call valuation of $\mathcal{R}$ with values in Γ any application $val : \mathcal{R} \to \Gamma$ satisfying the following three conditions:*

i) $val(a) = \infty \quad \Leftrightarrow \quad a = 0,$

ii) $val(ab) = val(a) + val(b), \quad \forall a, b \in \mathcal{R},$

iii) $val(a + b) \geq \min\{val(a), val(b)\}, \quad \forall a, b \in \mathcal{R}.$

Remark 3 *In general, we need to consider the valuation val with its restriction* $\mathcal{K}^* \to \mathbb{R} \cup \{\infty\}$.

Remark 4 *Valuations are also defined for fields. Indeed, for tropical geometry, we use valuations over fields such as Piuseux series and rational numbers. Every field* $\mathcal{K}$ *has the trivial valuation* $val(a) = 0, \quad \forall a \in \mathcal{K} - \{0\}$.

In the case of the Puiseux series, the valuation corresponds to the lowest exponent that appears in the series.

Example 3 *Let's take a Puiseux series P with coefficients in the complex numbers. P has the form:*

$$P(t) = c_1 t^{\alpha_1} + \cdots c_n t^{\alpha_n},$$

where c_i *are the coefficients and* $\alpha_1 < \cdots < \alpha_n$. *Then*

$$val(P) = \alpha_1 \tag{2.13}$$

For the case of rational numbers, we use *p-adic* valuation. This is, given a prime number p and an element $q \in \mathbb{Q}^*$, we rewrite q as

$$q = p^n \frac{a}{b},$$

where $a, b \in \mathbb{Z}$, p does not divide neither a nor b. Then,

$$val(q) = n \tag{2.14}$$

Example 4 *Consider a 2-adic valuation of the following numbers: 6,* $\frac{8}{3}$, $\frac{1}{4}$. *The valuations*

are:

$$\begin{aligned} val(6) &= 2^1 \cdot 3 &&= 1, \\ val(8/3) &= 2^3 \cdot \frac{1}{3} &&= 3, \\ val(1/4) &= 2^{-2} \cdot 1 &&= -2. \end{aligned}$$

It is important to say that the image of *val* is an additive subgroup[1] Γ_{val} of the real numbers $\mathbb{R}$, called the **value group** of $(\mathcal{R}, val)$ [1]. It is also important to mention that the valuation allows us to pass from classical arithmetic to tropical arithmetic, this will help us in the future with the process of tropicalization of a classical variety.

The concept of valuation is essential to tropicalize a polynomial because it works on the field or ring of coefficients. To tropicalize the monomials, we need to use the Laurent polynomials because they allow negative exponents.

Definition 9 (Laurent polynomials) *A Laurent polynomial is defined in the polynomial ring* $\mathcal{K}[x_1^{\pm 1}, \ldots, x_n^{\pm 1}]$ *and is of the form*

$$f = \sum_{i=0}^{n} a_i x_1^{\alpha_{i1}} \cdots x_n^{\alpha_{in}} \quad \alpha_{ij} \in \mathbb{Z}. \tag{2.15}$$

Example 5 *An example of a Laurent polynomial can be the the following polynomial* $f \in \mathbb{R}[x^{\pm 1}, y^{\pm 1}]$*:*

$$f(x, y) = x + y^2 + x^{-1} - 4xy^{-1}$$

The point here is that the polynomial accepts negative exponents for the variables.

With definitions 8 and 9 we can now go to the concept of tropicalization of a polynomial.

Definition 10 (Tropicalization) *Let* $f = \sum_{i=0}^{n} a_i x_1^{\alpha_{i1}} \cdots x_n^{\alpha_{in}}$ *belong to the polynomial ring* $\mathcal{K}[x_1, \ldots, x_n]$*, where* $\mathcal{K}$ *is a field with a non trivial valuation. The tropicalization of* f*, denoted as* $trop(f)$*, is:*

[1]This is because, according to *ii* in Definition 8, a valuation is a homomorphism from $(\mathcal{R}^*, \cdot)$ to $(\mathbb{R} \cup \{\infty\}, +)$.

$$trop(f) = \min_{0 \leq i \leq n} \{val(a_i) + \alpha_{i1}x_1 + \cdots + \alpha_{in}x_n.\} \tag{2.16}$$

Example 6 *Let $f \in \mathbb{R}[x, y, z]$ be a polynomial given by the equation $f = 4x^2 + 8y^2 - 6xy + 2x + 12y - 16$. Let's assume the 2-adic valuation for the field $\mathbb{R}$. The tropicalization of f, denoted as $trop(f)$, is showed in the following process:*

$$f = 4x^2 + 8y^2 - 6xy + 2x + 12y - 16$$

The coefficients are 4,8,−6,2,12,−16. Then, using the valuation showed in Equation 2.14, the valuations are:

$$\begin{aligned} val(4) &= 2^2 \cdot 1 &&= 2, \\ val(8) &= 2^3 \cdot 1 &&= 3, \\ val(-6) &= 2^1 \cdot (-3) &&= 1. \\ val(-2) &= 2^1 \cdot (-1) &&= 1. \\ val(12) &= 2^2 \cdot 3 &&= 2. \\ val(16) &= 2^4 \cdot 1 &&= 4. \end{aligned}$$

which produce the following tropical polynomial:

$$\begin{aligned} trop(f) &= \min\{2 + 2x, 3 + 2y, 1 + x + y, 1 + x, 2 + y, 4.\} \\ trop(f) &= (2 \otimes x^2) \oplus (3 \otimes y^2) \oplus (1 \otimes x \otimes y) \oplus (1 \otimes x) \oplus (2 \otimes y) \oplus (4) \end{aligned} \tag{2.17}$$

Example 7 *Let $g \in \mathcal{K}[x, y]$ be a polynomial given by the equation $g = 3tx^2 + 5xy - 7ty^2 + 8x - y + t^2$ where the field $\mathcal{K}$ is the field of Puiseux series, which consists on polynomials over the variable t. The tropicalization of g, denotes as $trop(g)$, is showed below:*

$$g = 3tx^2 + 5xy - 7ty^2 + 8x - y + t^2$$

The coefficients are $3t, 5, -7t, 8, -1, t^2$. Then, using the valuation showed in Equation

2.13, the valuations are:

$$val(3t) = 1$$
$$val(5) = 0$$
$$val(-7t) = 1$$
$$val(8) = 0$$
$$val(-1) = 0$$
$$val(t^2) = 2$$

which produce the following tropical polynomial:

$$trop(g) = \text{mín}\{val(3t) + 2x, val(5) + x + y, val(-7t) + 2y,$$
$$val(8) + x, val(-1) + y, val(t^2)\}$$

$$trop(g) = \text{mín}\{1 + 2x, x + y, 1 + 2y, x, y, 2\}$$
$$trop(g) = (1 \otimes x^2) \oplus (x \otimes y) \oplus (1 \otimes y^2) \oplus (x) \oplus (y) \oplus (2)$$

This kind of polynomials are used in Chapter 4 for showing the results.

To define a tropical variety, we use the tropicalization of a classical variety given by an ideal $\mathcal{I}$ [16]. That is, given a variety $V(\mathcal{I})$, its tropicalization is the intersection of the tropicalizations of all the polynomials that generate the ideal $\mathcal{I}$.

Definition 11 (Ideal [13]) *A subset $\mathcal{I} \subset \mathcal{K}[x_1, ..., x_n]$ is and ideal if the following conditions are satisified:*

i) $0 \in \mathcal{I}$

ii) For all $f, g \in \mathcal{I}$, $f + g \in \mathcal{I}$

iii) For all $f \in \mathcal{I}$ *and for all* $h \in \mathcal{K}[x_1, ..., x_n]$, $fh \in \mathcal{I}$ *and* $hf \in \mathcal{I}$

Definition 12 (Tropical Variety) *Let $\mathcal{I}$ be an ideal in $\mathcal{K}[x^{\pm 1}]$ and let $\mathcal{X} = V(\mathcal{I})$ be its variety. The tropicalization $trop(\mathcal{X})$ of the variety $\mathcal{X}$ is the intersection of all the tropical hypersurfaces defined by Laurent polynomials in the ideal $\mathcal{I}$:*

$$trop(\mathcal{X}) = \bigcap_{f \in \mathcal{I}} trop(V(f)). \tag{2.18}$$

However, Equation 2.18 does not always hold. More specifically, if it holds, then the set $\{f_1, \ldots . f_n\} \in I$ is called a *tropical basis* for I. Thus, it is more precise to say that a finite intersection of tropical hypersurfaces is a tropical prevariety.

To understand how a tropical variety or prevariety looks like, let us introduce a simple tropical geometric construction: the tropical line. Consider the tropical line $trop(l(x, y)) = x \oplus y \oplus a$. The tropical hypersurface $trop(V(l))$ is the set in which the minimum is achieved at least twice. For the tropical line we have the following:

$$trop(l(x, y)) = \text{mín}\{x, y, a\}$$

which translates to the following systems of inequalities:

$$\begin{aligned} x = y \leq a \\ x = a \leq y \\ y = a \leq x \end{aligned} \tag{2.19}$$

The Equation 2.19 produces the curve seen in Figure 2.2. Using the concept of the tropical line, we are going to show the tropical variety generated by two tropical lines.

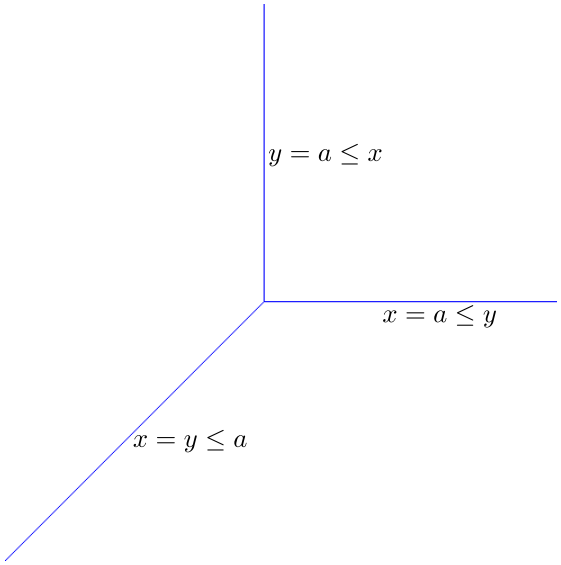


Figura 2.2: Curve of the tropical line $trop(l(x, y)) = x \oplus y \oplus a$

Example 8 *Consider the tropical lines* $l_1 = x \oplus y \oplus 0$ *and* $l_2 = (2 \otimes x) \oplus (-1 \otimes y) \oplus (1)$. *In Figure 2.3 we can see a graphical representation of these tropical lines. The hypersurface of* l_1 *is the red tropical line. The hypersurface of* l_2 *is the green tropical line. The variety/-prevariety generated by the two lines is the single point (0,2). Therefore, we can say that Equation 2.18 is fulfilled here:*

$$trop(V(l_1, l_2)) = trop(V(l_1)) \cap trop(V(l_2))$$

The decidability of whether $trop(V(l_1, l_2))$ *is a variety or prevariety is out of the scope of this work. Remember that* $trop(V(l_1, l_2))$ *would be a variety if the ideal generated by* l_1, l_2 *is a basis.*

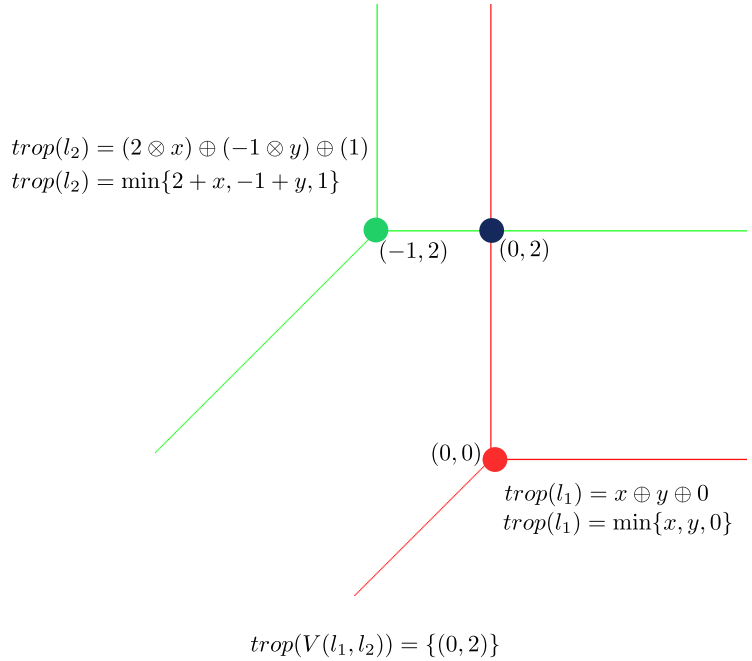


Figura 2.3: Representation of a tropical variety generated by two tropical lines

2.1.4. Polyhedral geometry

The principal objective of this work is to implement an algorithm to compute the tropical hypersurface of a polynomial. Then, by intersecting hypersurfaces from several polynomials of an ideal I, we can compute the tropical variety I. For this purpose, we need to understand how to form the tropical curves. Then, in this section, the necessary concepts about polyhedral geometry are introduced in order to explain the algorithms in

the methodology section. The definitions and concepts given in this section are taken from [1] and [17].

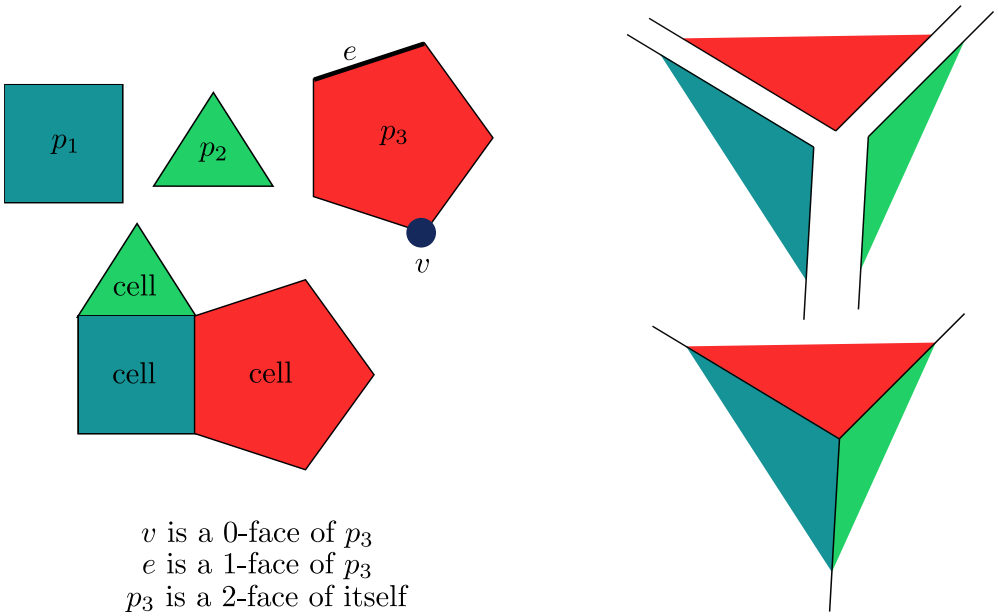
Definition 13 (Polyhedral Complex) *Let us consider a polyhedron $\mathcal{P}$ of $\mathbb{R}^n$ and let us consider a face $\mathfrak{f}$ of this polyhedron. Then, a* ***polyhedral complex*** *is a collection Σ of polyhedra satisfying the following conditions:*

i) If $\mathcal{P} \in \Sigma$, then for all $\mathfrak{f} \in \mathcal{P}, \mathfrak{f} \in \Sigma$,

ii) If $\mathcal{P} \cap \mathcal{Q} = \mathfrak{f}$, then $\mathfrak{f} \in \mathcal{P} \wedge \mathfrak{f} \in \mathcal{Q}, \forall \mathfrak{f} \neq \emptyset$.

Definition 13 says that every face of a polyhedron that is in the polyhedral complex must be a face in the complex. The next condition says that the intersection of two polyhedra $\mathcal{P}, \mathcal{Q}$ can result into a non-empty face $\mathfrak{f}$, and that face must belong to $\mathcal{P}$ and $\mathcal{Q}$

When we work with tropical polynomials, we deal with a particular case of a polyhedron, a *cone*, and the collection of polyhedral cones forms a **polyhedral fan**. In Figure 2.4a, we can see some polyhedra that are part of a polyhedral complex, and in Figure 2.4b, we can see some cones that are part of a polyhedral fan.



(a) Polyhedra and polyhedral complex (b) Cones and polyhedral fan

Figura 2.4: Polyhedral complex and polyhedral fan

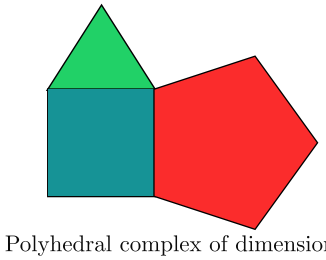
Definition 14 (Cells) *The polyhedra in a polyhedral complex Σ are called the cells of Σ.*

Definition 15 (k-face) *Given a polytope of dimension d. A **k-face** of a polyhedral complex Σ is a polytope of dimension $k \leq d$.*

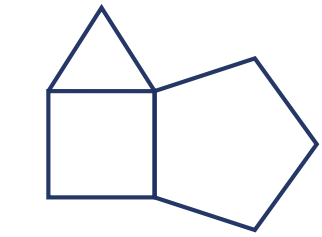
Remark 5 *A 0-face is called a vertex, a 1-face is called an edge, a $d-1$ face is called a facet and a $d-2$ face is called a ridge [17].*

Definition 16 (k-skeleton) *The **k-skeleton** of a polyhedral complex Σ of dimension d are all the cells $\sigma \in \Sigma$ such that $dim(\sigma) \leq k \leq d$.*

Remark 6 *A k-skeleton of a polytope is the union of its faces of dimensions $k, k-1, \ldots, 0$ (See Figure 2.5).*



Polyhedral complex of dimension 2



1-skeleton of the polyhedral complex

Figura 2.5: 1-skeleton of a polyhedral complex of dimension 2.

2.2. Computational background

2.2.1. Computational Geometry

In [1], it is mentioned that a curve $V(p)$ of a polynomial can be constructed as a subdivision of the Newton polytope of p where the coefficients of p give that subdivision. The subdivision is given by a projection $\pi : \mathcal{K}^{n+1} \to \mathcal{K}^n$ of the lower faces of the polytope. Thus, we need to define those concepts.

Definition 17 (Convex set/region) *A set $\mathcal{S} \subset \mathbb{R}^n$ is said to be convex if for all $x, y \in \mathcal{S}$ it also contains the line segment $[x, y] = \{\lambda x + (1-\lambda)y : 0 \leq \lambda \leq 1\}$.*

Definition 18 (Convex hull) *Given a set $\mathcal{S} = \{x : x \in \mathbb{R}^n\}$, the* ***convex hull*** *of $\mathcal{S}$ denoted as $conv(\mathcal{S})$ is the smallest convex region containing the set $\mathcal{S}$.*

In Figure 2.6, we can see an example of a convex hull in $\mathbb{R}^2$. The convex hull forms a polygon since it is the set of lines that match the outermost points of a set $\mathcal{S}$. The *Newton polytope* gives the relation between convex hulls and tropical polynomials.

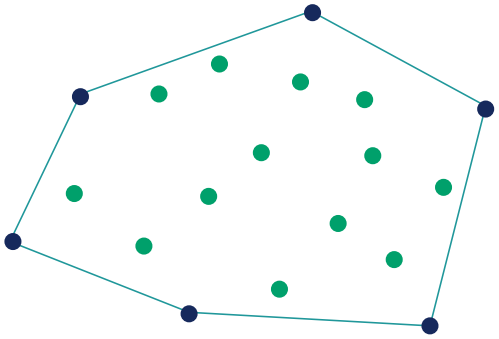


Figura 2.6: Example of a convex hull in $\mathbb{R}^2$

Definition 19 (Lower face) *Given a polytope $\Sigma \subset \mathcal{K}^{n+1}$, its lower faces are those whose inner normal vector $v \in (\mathcal{K}^{n+1})^*$ with last coordinate positive*

In Figure 2.7, we can see an example of polyhedra in (x, y, z) that has only one lower face, the one whose inner normal is positive in z. In this case, as we are in the 3-dimensional case, having the last coordinate positive means that the inner normal points up in the z-axis.

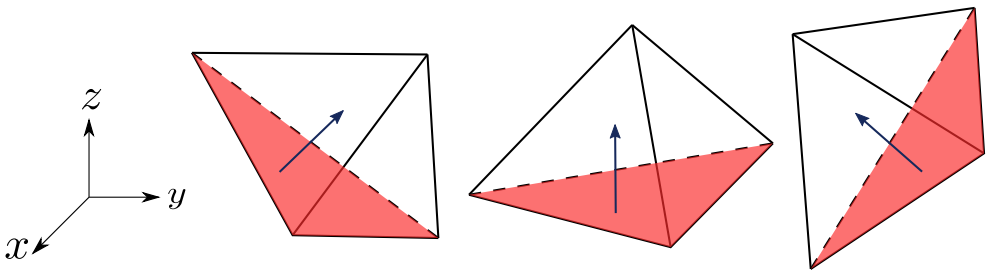


Figura 2.7: Lower face of polyhedra

Definition 20 (Newton polytope) *Given a polynomial $f \in \mathcal{K}[x_1, \ldots, x_n]$ such that $f = \sum_{i=0}^{s} a_i x_1^{\alpha_{i1}} \cdots x_n^{\alpha_{in}}$, the Newton polytope, denoted as $New(f)$ is the set*

$$New(f) = conv\{(\alpha_{i1}, \ldots, \alpha_{in}, val(a_i)), 0 \leq i \leq s\}. \tag{2.20}$$

The Newton polytope of a polynomial is defined as the convex hull of the points whose coordinates are the exponents of its monomials. For this reason, computing the convex hull of a set S is one of the main algorithms that will be present in the package.

2.2.2. Functional Programming

We developed this package using a functional programming paradigm with the Haskell programming language.

Functional programming is a programming paradigm based on *lambda calculus* that performs the flow of computations as function compositions rather than sequences of actions in imperative programming. The fundamental concepts of functional programming are immutability, functions that are first-class citizens, lazy evaluation, and no side effects [18].

Immutability ensures the inexistence of side effects; this feature is called pure programming since every function becomes pure in the sense that for every call of the function with a set of arguments, the result will always be the same, like a mathematical function. In imperative languages, side effects are produced by the change of state of variables that are global or have outer scope than the functions that call them. That is why immutable variables avoid the side effects of functions. Another consequence of immutable variables is that every function becomes independent from each other. The importance of this is that functions can be computed in parallel because the state of a function `f` does not affect the state of a function `g`.

Lazy evaluation consists on the property of not performing every single computation at once. Instead, computations are performed when needed. This feature is great because it can speed the computations by not getting stuck in some parts of the code. To understand laziness better, let us define the following infinite list using Haskell syntax: `a = [1..]`. Now imagine that we have a function that requires the first element of `a` using the function `head`. Without laziness, the computation `head a` will never end since the program should compute all the list first before asking for the first element. Indeed, the memory will fill up because `a` is an infinite list. On the other hand, using laziness, the computation `head a` gives the result immediately because the compiler does not care about the whole list `a`, but it understands that it is only required the first element and computes only the first element.

Finally, having functions as first-class citizens means that functions can be bound to variables. Indeed, in Haskell, functions are treated as types by themselves. Functional programming also supports high-order functions, which means that a function can receive another function as an argument or return a function as output.

The Haskell Programming Language

Haskell is a functional programming language. Its main features are polymorphically statically typing, laziness and pure functional programming. The language is named after Haskell Brooks Curry, whose work in mathematical logic serves as a foundation for functional languages [19].

Haskell implements *strong static typing*, which is very useful when a program needs type safety [18]. Particularly, for the tropical algebra package, it is vital to have a type-safe system. One example could be operations between polynomials. One can define the polynomial $p_1(x) = x$ and the polynomial $p_2(x, y) = x$. With p_1 and p_2, one could not perform any operation like sum or product because those polynomials belong to different rings. That is why having a weak-typing system could produce mathematical errors while performing computations.

Another property that is useful for algebraic operations is the concept of *dependent typing*. This concept is used in the Tropical Algebra Package and also in [10] to check the arity of polynomials. An example of why *dependent typing* is important is that, classically, $V(p_1) = \{0\}$, whereas $V(p_2)$ is the y-axis. However, dependent typing in Haskell is not a primary feature, but it can be emulated.

Parametric polymorphism is also an essential feature in Haskell. This type of polymorphism makes that the same function can be implemented for any type, rather that *ad-hoc* polymorphism in which you need to overload a function for each type [20]. Let us consider the reverse function, a function that takes a list and returns its reverse:

```
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = (reverse xs) ++ [x]
```

Notice that the function takes as a parameter a list of elements of type `a`. No matter which type it is, the function only takes the elements and reorders them. The reverse function is general and powerful because it is parametric polymorphic.

2.3. Software Engineering Background

As the end product of this project is a package, some software engineering aspects must be considered. Testing, documentation, and versioning are the main aspects to take into account to get a reliable and scalable product.

Testing

Test-Driven Development (TDD) is a software engineering practice that consists of designing tests before writing any code [21]. The purpose of this practice is to help the programmer to write a more transparent, more concise and more robust code. It is possible to obtain those features because the code is written based on the tests and not vice-versa. Also, having tests for every unit of code makes debugging faster, since if a bug appears in the code, the test structure finds it immediately and pinpoints its location [21].

TDD works very well in unit testing. Functional programming works as a composition of functions, then using TDD on each of these functions ensures the correct functioning of the software.

For the programmer, TDD improves the confidence and emotional stability of the code writer [22]. Since the test accumulates, when the project grows, the programmer increases its confidence over the behavior of the system. It does not happen in non-TDD approaches since the confidence of the code may be decaying as the project grows.

Documentation and versioning

Hackage is a repository of Haskell projects. In this repository, the documentation is generated using a powerful tool called Haddock [18]. Haddock works by checking every `.hs` (the Haskell source file extension) file and generating a `.html` file with the structure of the `.hs` file. For doing this, some special comments must be written in the `.hs` file. Some types of comments are:

- `-- | comment`: Works for commenting before the element.
- `-- ^comment`: Works for commenting after the element.
- `{-| comment in several lines -}` : Works for commenting a block of lines.
- `-- * comment`: Works for defining an item in the table of contents of the `.html` file.

For the versioning, Hackage also provides policies. Those can be found on the Hackage web page [23].

Capítulo 3

Methodology

In this section, we explain the development of the package by showing the data model of the system and explaining the main algorithms. From this point on, Haskell syntax is used to show data types and functions.

3.1. The Data Model

For this work, we created the definition of the following data types that are classified into three categories: arithmetic, algebraic and geometric.

3.1.1. Arithmetic data types

- **Tropical numbers:** this data type works for tropical numbers that will conform the coefficients of a tropical polynomial. The `Tropical` data type is polymorphic and the parameter `a` stands for any data type. Furthermore, this data type admits `Inf` as another value of the type, which emulates the set of the tropical semiring: $\mathbb{R} \cup \{\infty\}$

```
data Tropical a = Tropical a | Inf
```

- **Tropical matrices:** This data type works for tropical matrices. As well as de data type for tropical numbers, tropical matrices are also polymorphic on the type `a`.

```
newtype TMatrix a = TMatrix [[a]]
```

In [1], it is shown how the multiplication, $n-1$ times, of a $n \times n$ tropical matrix M is equivalent to the Floyd-Warshall algorithm to compute the shortest path for all pair of vertices. Then, the matrix M must be the adjacency matrix.

3.1.2. Algebraic Data Types

- **Monomials:** This data type works for the definition of a monomial that depends on the arity and the monomial order.

```
newtype Monomial ord n = Monomial {getMonomial :: Mon n}
```

- **Polynomials:** This data type works for defining a polynomial as a map (or a dictionary) in which there exist key-value pairs, where the monomial is the key and a value of type `k` is the value. The parameter `k` is usually the data type `Tropical Int`, but it can be any type that meets tropical requirements.

```
newtype Polynomial k ord n = Map Monomial k
```

3.1.3. Geometric Data Types

- **Points in $\mathcal{K}^2$:** This data type stores the possible points on a 2-dimensional affine space $\mathcal{K}^2$ on $\mathcal{K}$. $\mathcal{K}$ is generally the field of real numbers or the ring of integers. In this latter case, the points are conformed by integers of finite precision because we use this data type in the computations of Newton polytopes, in which the variable exponents give the coordinates. Remember that we work with integer numbers and not only with natural numbers because we deal with Laurent polynomials (see Definition 9).

```
type Point2D = (Int, Int)
```

- **Points in $\mathcal{K}^3$:** This data type has the same functionality as the previous one. The only difference in that lies in the affine space $\mathcal{K}^3$. As well as `Point2D`, this data type is used with finite-precision integers.

```
type Point3D = (Int, Int, Int)
```

- **Vertices:** The `Vertex` data type is a wrapper of the data type `Point3D`. It is called vertex because it represents a 0-face of a polytope.

```
newtype Vertex = Vertex {coordinates :: Point3D}}
```

- **Edges:** The `Edge` data type represents a 1-face of a polyhedron and is conformed by a pair of vertices,

```
newtype Edge = Edge {vertices :: (Vertex, Vertex)}}
```

- **Facets:** The `Facet` data type represents a n-1 face. In this case, a facet represents a 2-face because we are dealing with polyhedra, which are polytopes of dimension 3. A facet is conformed by a set of edges that must be ordered counterclockwise.

```
newtype Facet = Facet {edges :: [Edge]}}
```

- **Convex Hull:** This datatype represents a convex hull of a set of points in $\mathcal{K}^3$. As mentioned before for `Point2D` and `Point3D`, $\mathcal{K}$ can be the unitary ring of integers.

```
newtype ConvexHull = ConvexHull {facets :: [Facet]}}
```

- **Conflict Graph:** This data type is used in the algorithm of computing the convex hull in $\mathcal{K}^3$. It stores two maps or dictionaries.

  The first dictionary has key-value pairs of a vertex and a list of facets. The list of facets consists on those seen by the vertex when the vertex points to the convex hull.

  The second dictionary has key-value pairs of a facet and a list of vertices. The list of vertices consists on those vertices that see the facet when they point to the convex hull.

  This data structure helps to ask for information when computing the convex hull. The idea of using a conflict graph is taken from [24].

```
data ConflictGraph = ConflictGraph{
verticesF :: MS.Map Vertex [Facet],
facetsV :: MS.Map Facet [Vertex]
}
```

3.2. Algorithms

3.2.1. Convex Hull algorithm

Two dimensional case

For the case of the two-dimensional convex hull, we selected the Graham scan algorithm presented in [25], which takes a point p_0 and orders the other points taking the polar angle with respect to p_0 and an axis. However, this requires changing to polar coordinates, which can affect the exact computation. Thus, a variation of this algorithm is also presented in [25], which computes the convex hull by dividing the set of points in two subsets. To achieve

this, the right-most point p_r and the leftmost point p_l are selected. A line m from p_r to p_l is constructed, the points above this line will form the upper hull and the points below this line will form the lower hull. To compute the upper hull, we take the points in the order of the abscissae and select three points p_1, p_2, p_3. If the internal angle $\angle p_1p_2p_3$ is greater than or equal than 180°, then $\angle p_1p_2p_3$ is called a "right turn", otherwise is it a "left turn" (see Figure 3.1). In case $\angle p_1p_2p_3$ is a "left turn", we can advance in the scanning with $\angle p_2p_3p_4$, otherwise p_2 is dropped out from the convex hull and the checking is done for $\angle p_0p_1p_3$. Computationally, to determine whether a triple of points is a left or right turn, we can compute the determinant of these points. More specifically, in the case of the upper hull, we compute the determinant of points p_1, p_2, p_3 as seen in equation 3.1.

$$\begin{vmatrix} p_1 & 1 \\ p_2 & 1 \\ p_3 & 1 \end{vmatrix} = \begin{vmatrix} p_{1x} & p_{1y} & 1 \\ p_{2x} & p_{2y} & 1 \\ p_{3x} & p_{3y} & 1 \end{vmatrix} = a \tag{3.1}$$

If the value a in equation 3.1 is less than or equal to 0, it is considered a left turn; otherwise, it is considered a right turn.

For the case of the lower hull, we check for the value $a \geq 0$ to be a left turn; otherwise, it is a right turn.

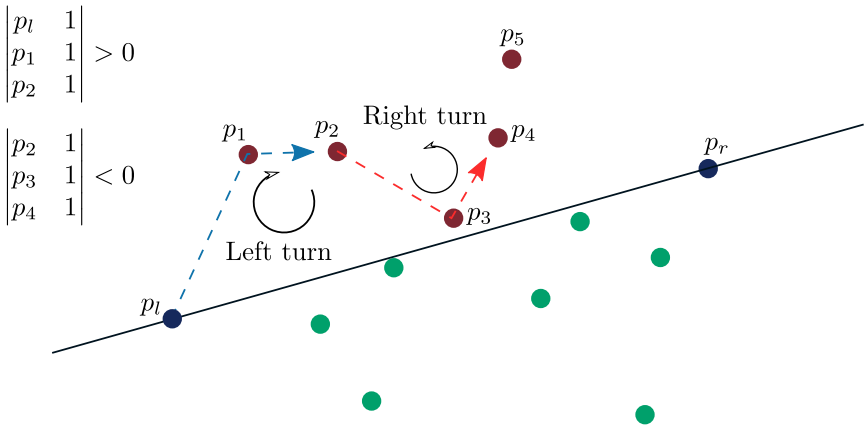


Figura 3.1: Representation of left and right turns

Finally, the total convex hull is the union of the upper semi-hull and the lower semi-hull. In Figure 3.2, we show a representation of the Graham Scan algorithm.

Three dimensional case

For the case of the three-dimensional convex hull, we chose the incremental algorithm presented in [24].

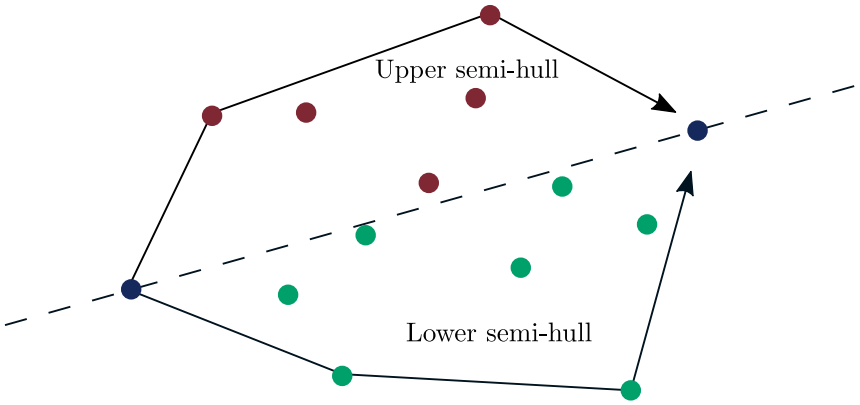


Figura 3.2: Representation of Graham Scan algorithm.

The idea of this algorithm is easy to follow: take four points p_1, p_2, p_3, p_4 that are neither colinear nor coplanar and form a tetrahedron (see Figure 3.3). That initial tetrahedron is considered the convex hull at step 0 denoted as $\mathcal{CH}_0$. One thing to consider is that the formation of the facets of the initial tetrahedron must take care of the ordering of their vertices because they must be ordered counterclockwise.

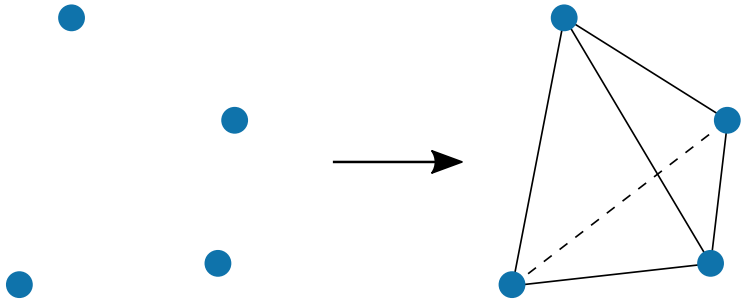


Figura 3.3: Four points forming a tetrahedron.

Then, for a convex hull $\mathcal{CH}_i$ take another point p_{i+1} and check whether it is inside or outside $\mathcal{CH}_i$. If it is inside, drop it out from the convex hull. Otherwise, we need to look for all the facets that are in front of the point p_{i+1}. Facets in front of the point p_{i+1} are only those that p_{i+1} sees when it looks at the polyhedron. If we look from the point p_{i+1} to the $\mathcal{CH}_i$, we can notice that there exists a horizon formed by the outer edge of each facet in front of p_{i+1}. The purpose is to construct a new facet from each edge that is in the horizon to the p_{i+1} (see Figure 3.4). After constructing these new facets, we need to drop the facets that remain inside the new convex hull $\mathcal{CH}_{i+1}$

It is important to notice that if the point p_{i+1} is coplanar to a face f that is not in front of p_{i+1} (i.e., behind the horizon), then, the new facet from the edge that belongs to f is an extension of it, and must be treated as that. Otherwise, we can get facets that are divided by an edge (see Figure 3.5).

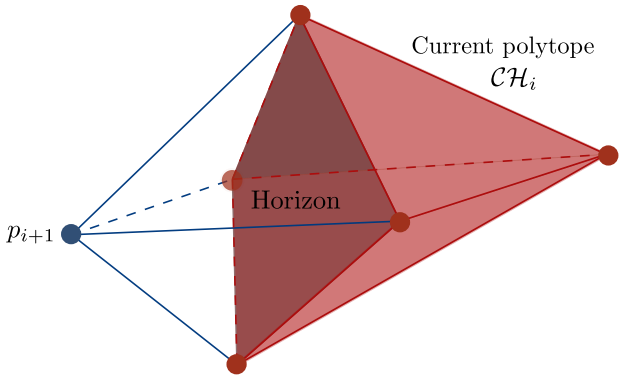


Figura 3.4: Generating new facets from the horizon to a new point

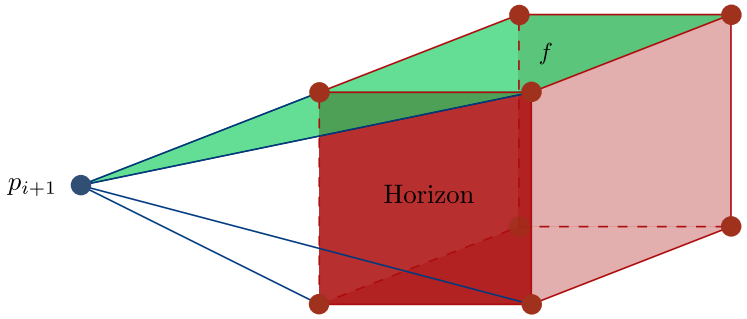


Figura 3.5: Representation of a point, p_{i+1}, that is coplanar to a face f. The new face is an extension of f

Now, once we presented the idea of the algorithm, we explain the algorithms used to accomplish the incremental algorithm to compute the convex hull.

- First, to check if a point p_{i+1} is inside the convex hull $\mathcal{CH}_i$, we have to compute, for every facet f_k if p_{i+1} is in front or behind f_k. A point inside a polyhedron must be behind all the facets composing that tetrahedron. If there is at least one facet for which the point is not behind, then the point is outside the polyhedron. To compute the *behindness* of a point concerning a facet, we need to have the vertices of the facet ordered counterclockwise and we compute the determinant of three arbitrary points p_1, p_2, p_3 of the facet and the point p_{i+1}, as in equation 3.2.

$$\begin{vmatrix} p_1 & 1 \\ p_2 & 1 \\ p_3 & 1 \\ p_{i+1} & 1 \end{vmatrix} = a \tag{3.2}$$

  It is very similar to the criterion used in Graham scan for the two-dimensional case. If $a \leq 0$ then we say p_{i+1} is in front of the facet with vertices p_1, p_2, p_3. Otherwise,

it is behind. Notice that a facet can have more than three vertices; in that case, we can take only three vertices as long as they are ordered counterclockwise.

With this method, we can know if a point is inside or not of a polyhedron.

- For every point p_{i+1} to be added to the convex hull $\mathcal{CH}_i$, we have to check for which facets the point p_{i+1} is in front of. The list of those facets are denoted as $\mathcal{F}$. Then, we have to compute which in edges the facets in $\mathcal{F}$ are on the horizon. It is achieved by flattening the list $\mathcal{F}$ into a list of edges $\mathcal{E}$ and taking out the edges that have a twin in the list $\mathcal{E}$.

Definition 21 (Twin Edges) *Given and edge e_1 of the form* `Edge (v1,v2)`*, we say that another vertex e_2 is a* ***twin*** *of e_1 if it is of the form* `Edge (v2,v1)`

Twin edges occur because the points are ordered counterclockwise and an edge shared by two facets is stored in a different order in each facet (see Figure 3.6). It is important to note here that the edges of the horizon do not have twins, because their twins are behind the horizon; therefore, they do not appear in the list of facets $\mathcal{F}$ that is in front of p_{i+1}

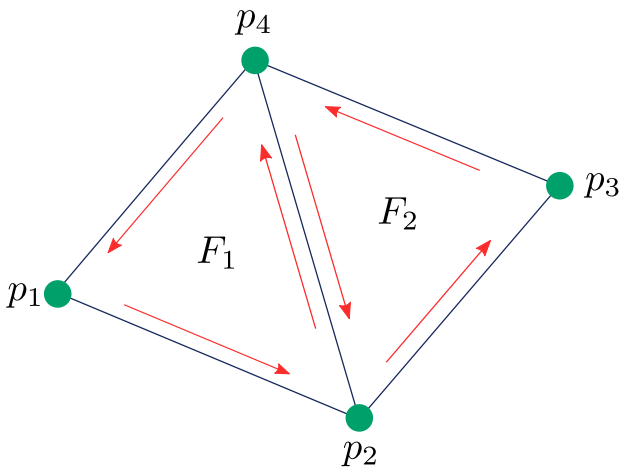


Figura 3.6: Twin edges that connect two facets.

- Now, once the horizon has been found, the construction of the new facets is straightforward. As the horizon is a list of edges, and each edge is of the form `Edge` $(v_1,\ v_2)$, we only need to add the point p_{i+1} to the end of each edge and form new the facet. That is, the new facet will have the vertices v_1, v_2, p_{1+1} ordered counterclockwise.

The only thing to take care of here is the case in which the new facet f is coplanar to another facet g that is behind the horizon. In that case, it is necessary to find the facet g and merge f and g. After all the new facets are generated, we have to update the conflict graph. First, we have to delete the facets that are now inside the polyhedron and drop their information from the registers of the remaining points $p_{i+2}, p_{i+3}, ..., p_n$. After that, we have to add the new facets to the `ConflictGraph`. Then, we have to compute the points are in front of the new facets and add their information in the registers of $p_{i+2}, p_{i+3}, ..., p_n$.

- Once we have done all those steps, we can add a new point and follow all the steps again.

3.2.2. Subdivision algorithm

Given a convex hull $\mathcal{CH} \in \mathcal{K}^3$, we perform a projection $\pi : \mathcal{K}^3 \to \mathcal{K}^2$. This will produce a polygon with internal subdivisions. The subdivision is given by the last coordinate of the points in $\mathcal{K}^3$. The projection π is only performed to the lower faces (definition 19) of $\mathcal{CH}$.

The subdivision algorithm is very similar to the computation of a Delaunay triangulation; the only thing that changes is how the last coordinate of each point is formed. In a Delaunay triangulation, the coordinates vector in $\mathbb{R}^3$ have the form $(x, y, x^2 + y^2)$, which defines a paraboloid, and in that way, the subdivision is going to form only triangles. In this case, given a Laurent polynomial in $\mathcal{K}[x, y]$ of the form

$$f = \sum_{i=0}^{s} a_i x^{\alpha_{ix}} y^{\alpha_{iy}} \quad \alpha_{ij} \in \mathbb{Z}, \tag{3.3}$$

the coordinate vectors take the form $(\alpha_{ix}, \alpha_{iy}, val(a_i))$. This way of forming the coordinates vectors does not ensure that the subdivision is made of triangles. Sturmfels and Maclagan in [1] state that if each cell in the lattice area is a triangle of area equal to $\frac{1}{2}$, then the subdivision is called *unimodular* and it will produce a smooth tropical curve.

Implementing the projection algorithm is not that hard. Given a convex hull $\mathcal{CH}$, we identify the inner normal of all the facets of it. Then we filter those facets whose inner normals are positive. After that, we will have all the facets formed by vertices of the form `v1,v2,v3`. To project each vertex, the following Haskell anonymous function was used:

```
\(v1,v2,_) -> (v1,v2)
```

That function takes a 3-tuple of three elements and returns a 2-tuple with the two first. Notice that the symbol _ means that we do not care about the third value of the input 3-tuple.

Then, applying the aforementioned function to the vertices of each edge of each lower facet of $\mathcal{CH}$ will project a polyhedron into a polygon with subdivisions (see Figure 3.7).

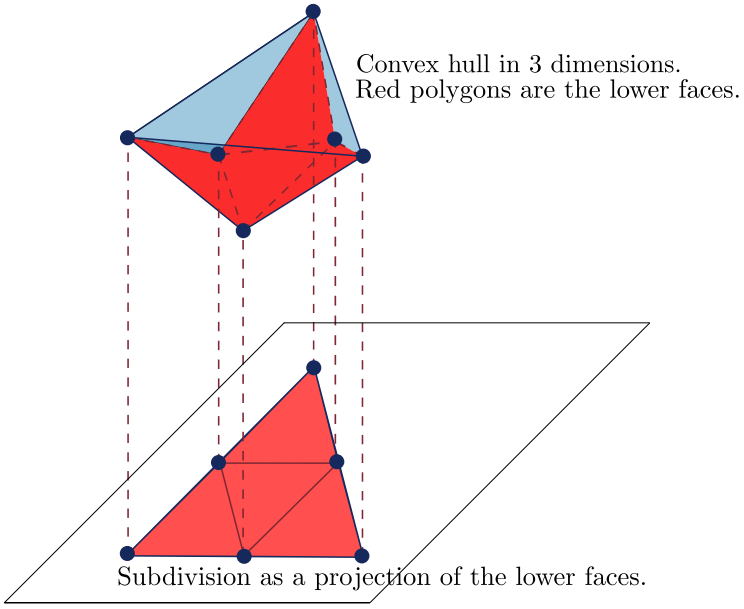


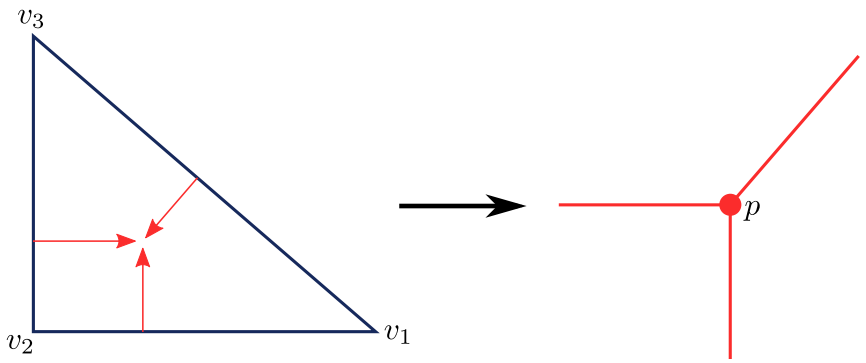
Figura 3.7: Representation of a subdivision from a convex hull of dimension 3 into a subdivided polygon of dimension 2.

3.2.3. Tropical Hypersurface

The tropical hypersurface of a polynomial f is the **k-skeleton** of the dual of the subdivision presented in the last section. To construct it, we must compute the inner normal fan of each polygon, generally triangle, in the subdivision. In Figure 3.8, we can see how this inner normal fan looks. To compute the inner normal vector of an edge of the form e_1 `= Edge (`v_1`,` v_2`)` we have to take an adjacent edge of the form e_2 `= Edge (`v_2`,` v_3`)`. Then we have to compute the dot product $w_1^N \cdot w_2$, where $w_1 = v_1 - v_2$, $w_2 = v_3 - v_2$ and w_1^N is the normal of w_1. If that product is positive, then the normal of w_1, which is also the normal of e_1 is w_1^N; otherwise, it is $-w_1^N$. The following code is the Haskell function that computes the inner normal of an edge. It takes three vertices: the first is the common vertex of both edges, the second is the second vertex of the edge to compute the normal, and the third is the second vertex of the adjacent edge. The result is a vector of the direction of the inner normal.

```
innerNormal :: Point2D -> Point2D -> Point2D -> Point2D
innerNormal a@(x1,y1) b@(x2,y2) c@(x3,y3)
 | dot > 0 = nab
```

```
| dot < 0 = (y1-y2,x2-x1)
where
ab = (x2-x1,y2-y1)
ac = (x3-x1,y3-y1)
nab = (y2-y1,x1-x2) -- (y,-x)
dot = (y2-y1)*(x3-x1) + (x1-x2)*(y3-y1)
```



Triangle with its inner normals

Inner normal fan that forms a tropical line

Figura 3.8: Representation of the inner normal fan of a triangle.

As seen in the left side of Figure 3.8, a triangle of the subdivision is formed by three vertices v_1, v_2, v_3, coming from a monomial of the Laurent polynomial. Then, to know in which point p the inner normals converge, we have to recover those three monomials and solve the system that they form. Remember that a monomial of the form $ax^{\alpha}y^{\beta}$ becomes, after tropicalization, into $a+\alpha x+\beta y$. Then, having three monomials m_1, m_2, m_3 produces the system 3.4, whose solution is the intersection of the three inner normals of the triangle. In Figure 3.8 we can see that from a polygon we pass to a normal fan. When this polygon is a triangle, the edges of the produced fan forms a tropical line as seen in the right side of the Figure 3.8.

$$\begin{cases} m_1 & = m_3 \\ m_2 & = m_3 \end{cases} \tag{3.4}$$

In the previous algorithm, we mentioned that if the subdivision is conformed by triangles with area $\frac{1}{2}$, the tropical curve will be smooth. Then the hypersurface algorithm computes tropical hypersurfaces assuming that the subdivision is unimodular [1]. That is, only if the tropical curve is smooth. To extend the algorithm to treat non-smooth curves, we should solve a system that is not $n \times n$.

Workflow of the tropical surface algorithm

In this part, we are going to show how the algorithm works for a certain polynomial. Let us consider the polynomial g used in Example 7:

$$g = 3tx^2 + 5xy - 7ty^2 + 8x - y + t^2$$

g is a polinomial in $\mathcal{K}[x, y]$ where $\mathcal{K}$ is the field of Puiseux series. In Example 7 we have shown that the tropicalization of g is

$$trop(g) = (1 \otimes x^2) \oplus (x \otimes y) \oplus (1 \otimes y^2) \oplus (x) \oplus (y) \oplus (2) \tag{3.5}$$

$$trop(g) = \text{mín}\{1 + 2x, x + y, 1 + 2y, x, y, 2\} \tag{3.6}$$

From Equation 3.5, we can define the Newton polytope of $trop(g)$ as:

$$New(trop(g)) = conv\{(2, 0, 1), (1, 1, 0), (0, 2, 1), (1, 0, 0), (0, 1, 0), (0, 0, 2)\} \tag{3.7}$$

The convex hull in Equation 3.7 has its graphical representation in Figure 3.9. We only show the lower faces (red) because those are the ones that will be projected.

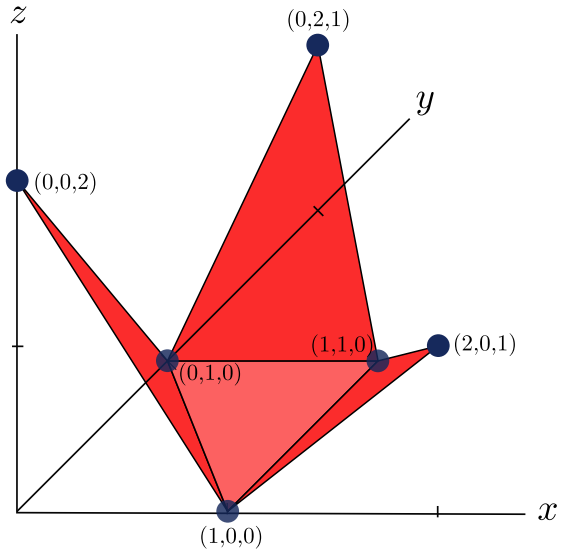
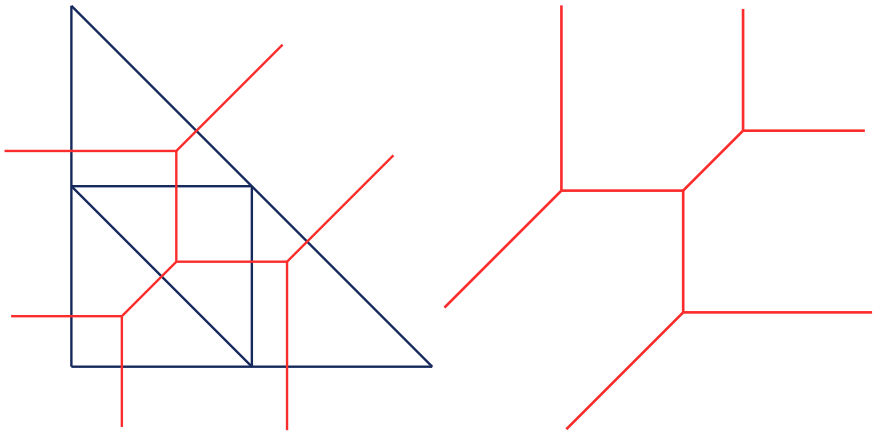


Figura 3.9: Lower faces of the Newton polytope of g

The projection of the lower faces of Figure 3.9 is showed in Figure 3.10a including its dual. The dual is the collection of red segments. The tropical hypersurface of g is the dual of the subdivision rotated 180° (See Figure 3.10b).



(a) Subdivision and its dual (b) Tropical hypersurface of g

Figura 3.10: Representation of the tropical hypersurface of g from the subdivision of a triangle.

3.3. Software Engineering Aspects

3.3.1. Dependencies

The package has the following dependencies that are in Hackage [23]:

- **containers:** This package is needed for those data types used to store data such as trees, dictionaries, sets. For this project, we use a data type from this package called `Map`, which is an ordered list whose elements are key-value pairs. We use the `Map` to store the polynomials, where the monomial is the key, and the coefficient is the value.
- **semiring-simple:** This package includes the classes to implement a data type that behaves as a semiring. It is used for tropical numbers.
- **algebra:** This package includes all the algebraic structures features like monoids, rings or fields. We need it to make our data type behave as algebraic structures. One example is that we can create a data type and define ring properties like two binary operations, identity elements and distribution of product over addition.
- **singletons:** This package is used for implementing polynomials as dependent data types. When we say dependent, we refer dependent to arity. That is, we can define a polynomial in two variables as `Polynomial k ord 2`, where `k` is the field of coefficients, `ord` is the monomial ordering and `2` is the arity. The interesting part is that the parameter `2` is not seen as a value but as a data type instead. Dependent typing provides several advantages that we will show further.

- **matrix:** This package is used to implement tropical matrices and some algorithms like the shortest path.
- **gloss:** This package is used to make pictures of tropical curves in two variables using OpenGL.

3.3.2. File structure of the library

The package has been distributed in a way that arithmetic, algebraic, geometric and graphical things are separated. The following tree-diagram show the file structure:

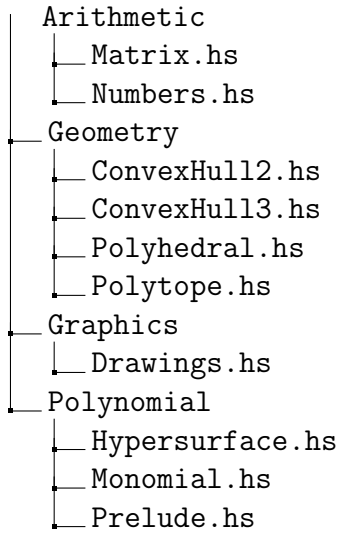


Figura 3.11: File structure of the package

The Arithmetic folder includes all the code to implement tropical numbers and tropical matrices. The Geometry folder has the code for geometric algorithms like convex hull, subdivisions, computation of the dual of a subdivision. The Graphics folder includes the code for drawing tropical curves, and the Polynomial folder has the code to implement the tropical polynomial data type.

3.3.3. Installation

To use this package as a dependency of another Haskell project, we strongly recommend to have the project built with either Cabal [26] or Stack [27]. We assume that the reader knows about the aforementioned build systems.

In the case of Cabal, you have to include the package name in the `.cabal` file in the `build-depends` section.

For Stack users, besides adding the package name to the `.cabal` file, it must be also added in the `stack.yaml` file in the `extra-deps` section.

In any option, the package will be downloaded from Hackage and bound to the project when it is compiled.

3.3.4. Tutorial

The following tutorial will show how to define tropical numbers, matrices, polynomials and the functionalities of them.

The following script called `Numbers.hs` shows the use of tropical numbers:

```haskell
module Numbers where


a,b,c :: Tropical Int
a = Tropical 2
b = Tropical 3
c = Inf


main :: IO()
main = do
    print $ show (a+b) -- prints "Tropical 2"
    print $ show (a*b) -- prints "Tropical 5"
    print $ show (a+c) -- prints "Tropical 2"
    print $ show (a*c) -- prints "Inf"
```

The next script, called Matrix.hs, shows how to use tropical matrices:

```haskell
module Matrix where

m1 :: TMatrix (Tropical Integer)
m1 = TMatrix
            [
                [ 0,   2, Inf,  3],
                [ 1,   0, Inf,  4],
                [ 8, Inf,   0,  1],
                [ 4,   1, Inf,  0]
            ]

-- Consider m1 as an adjacency matrix of some graph.
shortestPath = foldr1 mmult $ replicate 3 m1 -- Here we do m1*m1*m1


main :: IO()
main = do
    print $ show shortestPath

{-
Prints the following:

    TMatrix {toList = [
        [0,2,Inf,3],
        [1,0,Inf,4],
        [3,2,0,1],
        [2,1,Inf,0]]
    }
where each position (i,j) is the shortest path from the node i to
the node j. It is equivalent to the Floyd-Warshall algorithm.
-}
```

Finally, the next script shows how to deal with tropical polynomials.

```
module Polynomial where


-- definition of the variables as polynomials. The Polynomial data
-- type has 3 type parameters:
-- coefficient type
-- monomial ordering of variables
-- number of variables
x, y :: Polynomial (Tropical Integer) Lex 2
x = variable 0
y = variable 1


f1,f2 :: Polynomial (Tropical Integer) Lex 2


f1 = 1*x^2 + x*y + 1*y^2 + x + y + 2
f2 = 3*x^2 + x*y + 3*y^2 + 1*x + 1*y + 0


main :: IO()
main = do
    print $ show (f1 + f2)
    -- prints 1X_0^2 + X_0X_1 + X_0 + 1X_1^2 + X_1 + 0


    makeFig f1 -- will produce the tropical hypersurface
```

The last line in the previous code produces the tropical hypersurface seen in Figure 3.10b and also in Figure 4.1.

Capítulo 4

Results

4.1. Test set

We have taken five polynomials taken from [1] to run our examples. The following polynomials have variables x, y and their coefficients belong to the field of Puiseux series that are polynomials in the variable t. Those polynomials are:

- $f_1 = 3tx^2 + 5xy - 7ty^2 + 8x - y + t^2$
- $f_2 = 3t^3x^2 + 5xy - 7t^3y^2 + 8tx - ty + 1$
- $f_3 = 5t^3x^3 + 7tx^2y - 8txy^2 + 9t^3y^3 + 8tx^2 + 5xy - ty^2 + 4tx + 8ty + t^3$
- $f_4 = 5x^3 + 7x^2y + 8xy^2 + 9y^3 + 8x^2 + 5xy - y^2 + 4x + 8y + 1$
- $f_5 = 2xy^{-1} + 2y^{-1} + (-2)$

And their respective tropicalizations:

- $trop(f_1) = \text{mín}\{2x + 1, x + y, 2y + 1, x, y, 2\}$
- $trop(f_2) = \text{mín}\{2x + 3, x + y, 2y + 3, x + 1, y + 1, 0\}$
- $trop(f_3) = \text{mín}\{3x+3, 2x+y+1, x+2y+1, 3y+3, 2x+1, x+y, 2y+1, x+1, y+1, 3\}$
- $trop(f_4) = \text{mín}\{3x, 2x + y, x + 2y, 3y, 2x, x + y, 2y, x, y, 0\}$
- $trop(f_5) = \text{mín}\{x - y + 2, -y + 2, -2\}$

For a clear example of how to tropicalize a polynomial, the tropicalization of f_1 can be found in the Example 7 of Chapter 2.

After we perform all the computations, we get our tropical curve defined as a set of vertices and three emanating rays from each vertex. The representation of the emanating rays is given by normalized vectors. We show the results for the five polynomials aforementioned in the following tables and figures.

- **For f_1:**

Vertex	**Ray 1**	**Ray 2**	**Ray 3**
(-1,0)	(-1,-1)	(0,1)	(1,0)
(0,-1)	(-1,-1)	(0,1)	(1,0)
(0,0)	(-1,0)	(0,-1)	(1,1)
(2,2)	(-1,-1)	(0,1)	(1,0)

Cuadro 4.1: Vertices and rays for f_1

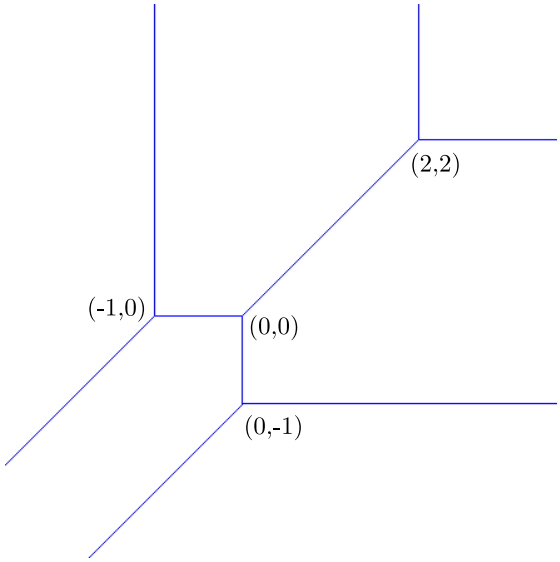


Figura 4.1: Tropical hypersurface for f_1, $trop(f_1)$

- **For f_2:**

Vertex	**Ray 1**	**Ray 2**	**Ray 3**
(-2,1)	(-1,-1)	(0,1)	(1,0)
(-1,1)	(-1,0)	(0,1)	(1,-1)
(1,-2)	(-1,-1)	(0,1)	(1,0)
(1,-1)	(-1,1)	(0,-1)	(1,0)

Cuadro 4.2: Vertices and rays for f_2

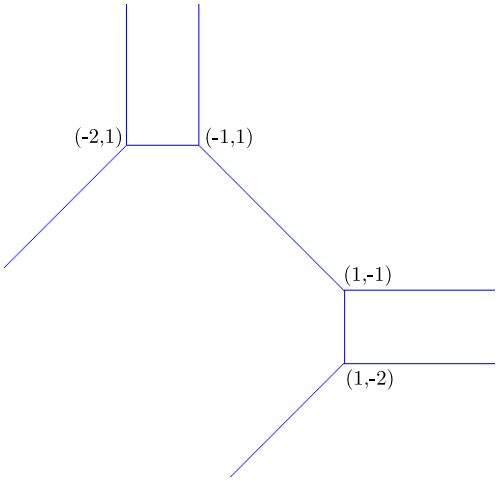


Figura 4.2: Tropical hypersurface for f_2, $trop(f_2)$

- **For f_3:**

Vertex	**Ray 1**	**Ray 2**	**Ray 3**
(-2,0)	(-1,-1)	(0,1)	(1,0)
(-1,-1)	(-1,-1)	(0,1)	(1,0)
(-1,0)	(-1,0)	(0,-1)	(1,1)
(0,-2)	(-1,-1)	(0,1)	(1,0)
(0,-1)	(-1,0)	(0,-1)	(1,1)
(0,1)	(-1,-1)	(0,1)	(1,0)
(1,0)	(-1,-1)	(0,1)	(1,0)
(1,1)	(-1,0)	(0,-1)	(1,1)
(2,2)	(-1,-1)	(0,1)	(1,0)

Cuadro 4.3: Vertices and rays for f_3

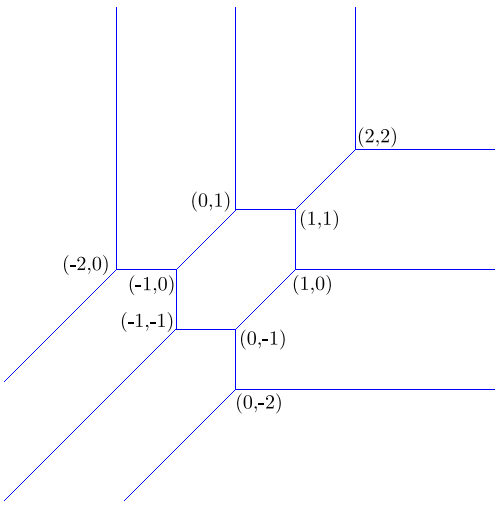


Figura 4.3: Tropical hypersurface for f_3, $trop(f_3)$

- **For f_4:**

Vertex	**Ray 1**	**Ray 2**	**Ray 3**
(0,0)	(-1,-1)	(0,1)	(1,0)

Cuadro 4.4: Vertices and rays for f_4

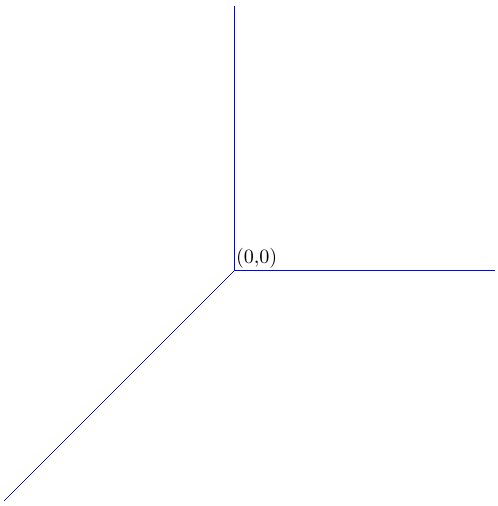


Figura 4.4: Tropical hypersurface for f_4, $trop(f_4)$

- **For f_5:**

Vertex	**Ray 1**	**Ray 2**	**Ray 3**
(0,4)	(-1,-1)	(0,1)	(1,0)

Cuadro 4.5: Vertices and rays for f_5

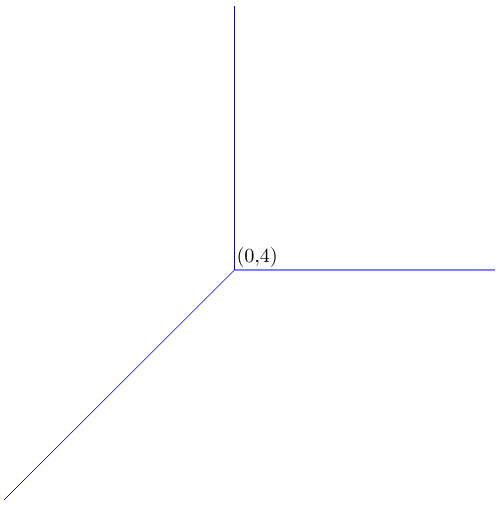


Figura 4.5: Tropical hypersurface for f_5, $trop(f_5)$

4.2. Comparison with other software

As mentioned in Chapter 1, other software is capable of computing tropical hypersurfaces. Those are Gfan [5] and Polymake [7]. Since Polymake embeds Gfan computations, the comparison in performance is made with Polymake only.

The specification of the hardware where the algorithms were written was:

- RAM: 16 Gb
- Processor: Intel Core i7-8700K
- Clock Speed: 3.70 GHz
- Processors: 6 physical, 12 logical

We added the following three tropicalized polynomials to the previous five presented previously: $trop(f_6)$, $trop(f_7)$ and $trop(f_8)$, which have degree 4,5 and 6, respectively.

- $trop(f_6) = 6x^4 \oplus 4x^3y \oplus 3x^2y^2 \oplus 4xy^3 \oplus 5y^4 \oplus 2x^3 \oplus x^2y \oplus 1xy^2 \oplus 4y^3 \oplus 2x^2 \oplus xy \oplus 3y^2 \oplus x \oplus 2y \oplus 5$
- $trop(f_7) = 6x^5 \oplus 2x^4y \oplus 4x^3y^2 \oplus x^2y^3 \oplus 3xy^4 \oplus 8y^5 \oplus 6x^4 \oplus 4x^3y \oplus 3x^2y^2 \oplus 4xy^3 \oplus 5y^4 \oplus 2x^3 \oplus x^2y \oplus 1xy^2 \oplus 4y^3 \oplus 2x^2 \oplus xy \oplus 3y^2 \oplus x \oplus 2y \oplus 5$
- $trop(f_8) = 10x^6 \oplus 8x^5y \oplus 6x^4y^2 \oplus 6x^3y^3 \oplus 4x^2y^4 \oplus 6xy^5 \oplus 9y^6 \oplus 6x^5 \oplus 2x^4y \oplus 4x^3y^2 \oplus x^2y^3 \oplus 3xy^4 \oplus 8y^5 \oplus 6x^4 \oplus 4x^3y \oplus 3x^2y^2 \oplus 4xy^3 \oplus 5y^4 \oplus 2x^3 \oplus x^2y \oplus 1xy^2 \oplus 4y^3 \oplus 2x^2 \oplus xy \oplus 3y^2 \oplus x \oplus 2y \oplus 10$

The results are:

Polynomial	Time Polymake (ms)	Our time(ms)
f1	20	0,31
f2	20	0,28
f3	20	1,49
f4	20	0,10
f5	20	0,01
f6	20	4,03
f7	20	7,09
f8	20	17,64

Cuadro 4.6: Comparison between Polymake and our package for polynomials $f_1, ..., f_8$.

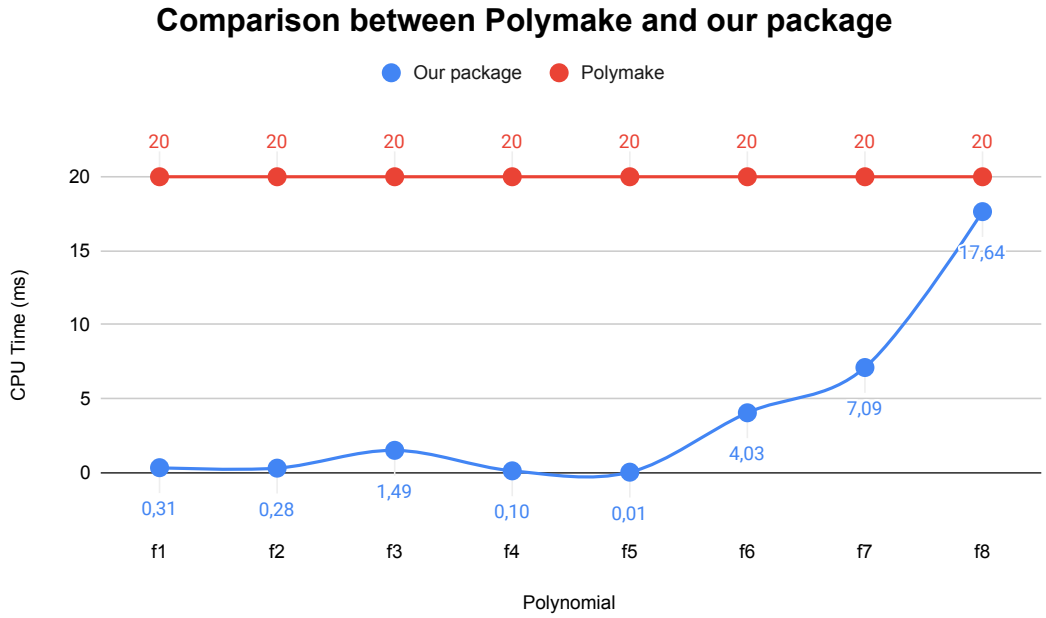


Figura 4.6: Comparison between Polymake and our package for polynomials $f_1, ..., f_8$

These benchmarks were performed with polynomials in two variables since we have not generalized the algorithm for polynomials of an arbitrary number of variables.

Capítulo 5

Discussion and Conclusions

5.1. Results analysis

We have tested the algorithm for five polynomials first and got their respective tropical curves. We notice that the way we store the hypersurfaces is proper for plotting. As the results are correct, we can say that the algorithm is working well. It is worth to say that we have worked only with smooth polynomials (polynomials that produce a unimodular triangulation). Thus, the behavior of the algorithm with non-smooth polynomials is not known at this point.

In terms of speed, we can see in fig. 4.6 that our algorithm is faster than the one in Polymake with polynomials with degree less or equal than 6. Notice that our computations of the polynomial f_8 are still faster than Polymake. For a polynomial of a higher degree, our algorithm becomes slower. However, in some instances, polynomials with higher degrees could be faster in our implementation. For that to happen, the Newton polytope of a polynomial f should not use all the coordinates given by the monomials of f. That is, some points should be inside the convex hull. That means that the computation of the convex hull is reduced to fewer points, which makes the whole algorithm faster.

A possible explanation of the results could be that our package resolves the hypersurface problem with only two variables, whereas Polymake works with n-dimensional polynomials. That means that even for simple cases, Polymake could perform unnecessary computations. However, when the problem gets bigger, Polymake turns to be faster, and those unnecessary computations are no longer so. However, this is only a conjecture since we do not know how exactly the Polymake or Gfan algorithm works.

5.2. Conclusion

The main objective of this package is to compute tropical hypersurfaces of polynomials in two variables by computing its Newton polytope and subdividing it according to the valuation of the coefficients. The use of Haskell as a programming language eased the coding at the implementation of data types and functions in a recursive way.

However, many other features can be used from this package. One of these is the usage of the tropical matrices to solve optimization problems such as shortest-path or the assignment problem. They correspond to the tropical matrix product and tropical determinant, respectively.

Furthermore, all the auxiliary functions used to implement the convex hull algorithm can be used for geometrical purposes. There are functions to check coplanarity and colinearity, to compute line segments, triangles, and tetrahedrons from a list of points. Indeed, the convex hull algorithm can be used for other geometrical purposes like the Delaunay triangulation.

The crucial fact about all the written algorithms is that they support arbitrary precision, which is a Haskell feature. It means that the propagation of errors caused by rounding rational numbers is avoided. Then, the code is more robust in that aspect, only depending on the capacity of the hardware.

Another advantage of Haskell is the existence of algebraic data types, which allows defining data types more succinctly and precisely. The main example of this was the definition of tropical numbers: `data Tropical a = Tropical a | Inf`, which emulates the field extension $\mathbb{R} \cup \{\infty\}$. Dependent typing was also a feature that we took advantage of by using it to ensure that polynomials of different arities do not ever operate between them. Another advantage, in terms of code readability, is the fact that Haskell has pattern matching, which allows avoiding excessive `if` or `switch` statements that would be inevitable in an imperative language.

This package offers a first glance at the combination of functional programming and tropical geometry. We expect that this package can be maintained over time and serve as a tool for scientists who need to compute tropical stuff and have or need to do it functionally.

5.3. Further work

Generalizing the algorithm for non-smooth tropical polynomials should be the next step. For this, it must be noticed that any polygons, not only triangles, may form subdivision. Then, the computation of the vertex and its emanating rays would change.

In terms of functionality, the package has some limitations. The main limitation is

related to working with polynomials in higher dimensions. Remember that to compute the tropical hypersurface of a polynomial requires to go through its Newton polytope. However, computing Newton polytopes of a polynomial of n variables requires the computation of a convex hull in $n + 1$ variables. Then, as the package can compute convex hull up to 3 dimensions, we can only work with polynomials up to 2 variables. Further research and development should focus on writing the algorithm for n-dimensional convex hull. In this way, the package will work with polynomials in any number of variables.

As well as n-dimensional convex hull, further research and development should also focus on the computation of tropical varieties. That topic requires to check concepts like tropical ideals, tropical basis, and monomial ideals. We can be tempted to think that the variety can be computed by intersecting tropical curves, but the thing is that the intersection of hypersurfaces does not always give a variety but a prevariety. This is shown in Lemma 3.7 in [28].

Further versions of the package may include computations that may need a higher amount of resources like memory or processing capacity. In this case, parallelization will be required, and we can take advantage of another great feature of functional languages. Since functions are independent of each other, parallelization is relatively easy to perform. That means that we do not have to take into account problems like shared variables. Following this idea, for parallelization, we do not have to take care of changes in the state of variables because variables are immutable.

Appendices

Appendix A: Code

First of all, we will introduce briefly the Haskell color coding:

- Color for defining data types
- Color for naming data types and type classes
- Color for declaring functions
- Color for operators
- Color for comments

Here we show the main parts of the written code. We will show data types first and then the most important functions

Algebraic data types[1]

We start with the data type for tropical numbers:

```haskell
-- Defintion of the tropical number data type
data Tropical a = Tropical {value :: a} | Inf


-- Definition of tropical sum betwwen numbers
(.+.) :: (Ord a) => Tropical a -> Tropical a -> Tropical a
(.+.) Inf t = t
(.+.) t Inf = t
(.+.) t1 t2 = Tropical $ (min `on` value) t1 t2


-- Definition of tropical product between numbers
(.*.) :: (Num a) => Tropical a -> Tropical a -> Tropical a
(.*.) t Inf = Inf
(.*.) Inf t = Inf
(.*.) t1 t2 = Tropical $ ((Prelude.+) `on` value) t1 t2
```

[1]Here, algebraic data types do not refer to the fact that data types are algebraic, but refer to data types that represent algebraic objects.

Now, data type for monomials:

```
type SNat (n :: Nat) = Sing n
type Sized' n a = DS.Sized Seq.Seq n a
type Mon n = Sized' n Int

-- Monomial is defined as an array of exponents but the array length
-- is treated as a type. That is the reason we need the three previous
-- lines.
newtype Monomial ord n = Monomial {getMonomial :: Mon n}
```

And for polynomials:

```
newtype Polynomial k ord n = Polynomial {
 getTerms :: MS.Map (Monomial ord n) k
 } deriving(Eq)
```

Geometric data types

Here we show the data types that are used in the geometric aspect of the algorithm. We have points in 2D and 3D, vertices, edges, facets and the data for the conflict graph.

```haskell
type Point2D = (Int, Int)
type Point3D = (Int, Int, Int)

newtype Vertex = Vertex {coordinates :: Point3D}

-- Edge as a pair of vertices. When constructing the initial
-- tetrahedron, the order of the vertices must be counterclockwise.
newtype Edge = Edge {vertices :: (Vertex, Vertex)}

-- Facet in this case is a 2-face. It is stored as a collection of
-- edges.
newtype Facet = Facet {edges :: [Edge]}

-- The convex hull of a set is the collection of all the facets
-- that conform that polyhedron.
newtype ConvexHull = ConvexHull {facets :: [Facet]}

-- | The conflict graph is a data structure that stores for each
-- point the list of facets that points views. And for every facet
-- the list of points that facet views.
data ConflictGraph = ConflictGraph {
 verticesF :: MS.Map Vertex [Facet],
 facetsV :: MS.Map Facet [Vertex]
 }
```

Functions

Function to build an initial tetrahedron

```haskell
computeTetrahedron :: [Point3D] -> Maybe [Point3D]
computeTetrahedron [] = Nothing
computeTetrahedron points
 | isNothing triangle = Nothing
 | isNothing nicePoint = Nothing
 | otherwise = (++) <$> triangle <*> fmap return nicePoint
 where
 triangle = computeTriangle points
 nicePoint = find (not . isCoplanar (fromJust triangle)) points
```

Function to compute the convex hull in 3D:

```haskell
-- | Assume every point is different
convexHull3 :: [Point3D] -> Maybe ConvexHull
convexHull3 points
| length points < 4 = Just $ convexHull2In3 points
| isNothing tetraHedron = Just $ convexHull2In3 points
| otherwise = Just convexHull
 where
 convexHull = addPoints initialCH afterTetrahedron conflictGraph
 conflictGraph = startConflictGraph initialCH afterTetrahedron
 initialCH = (initializeCH . fromJust) tetraHedron
 tetraHedron = computeTetrahedron points
 afterTetrahedron = points \\ fromJust tetraHedron
```

Function to compute the projection of the convex hull

```haskell
-- Computes the projection R^3 -> R^2 of a convex hull. The input is
-- a convex hull in R^3 and the output will be a regular subdivision
-- of the convex hull in R^2

projectionToR2 :: ConvexHull -> [[Point2D]]
projectionToR2 convexHull =
 let lengthConvexHull = length $ facets convexHull in
 if lengthConvexHull == 1 then
 projected.triangles.facetsInPoints $ facets convexHull
 else
 projected.triangles.facetsInPoints $ lowerFaces
 where
 lowerFaces = filter isLowerFace $ facets convexHull
 facetsInPoints = map fromFacet
 triangles = filter (\points -> length points == 3)
 projected = map (map project3To2)
```

Function to compute the sudivision of a regular triangulation. It is performed as a composition of the convexhull function and the projection function:

```haskell
subdivision :: (Integral k) => Polynomial k ord n -> [[Point2D]]
subdivision poly = (projectionToR2.fromJust.convexHull3) points
 where
 terms = MS.toList $ getTerms poly
 monExps = DS.toList . getMonomial
 toPoints (mon, coef) = let [a,b] = monExps mon
 in (a, b, fromIntegral coef)
 points = map toPoints terms
```

When we have the subdivision, we compute its dual and store as an array of pairs. Each pair has a vertex and the direction of the rays emanating from them. Here is the function:

```haskell
verticesNormals :: (IsMonomialOrder ord, Ord k, Integral k) =>
 Polynomial k ord n -> MS.Map Point2D Normals
verticesNormals poly = MS.fromList $
 map (findFanNVertex polyMap) triangles
 where
 polyMap = mapTermPoint poly
 triangles = subdivision poly
```

However, for plotting purposes, we need the information stored as an array of line segments, then we transform the vertices-normals structure in an array of line segments:

```haskell
hypersurface :: (IsMonomialOrder ord, Ord k, Integral k) =>
 Polynomial k ord n -> [(Point2D, Point2D)]
hypersurface poly = nub $ computeEdges
 (convertMap neighbors pointTriangles) pointNormals
 where
 pointNormals = verticesNormals poly
 neighbors = neighborTriangles
 (map sort $ subdivision poly) MS.empty
 pointTriangles = pointsWithTriangles poly
```

Appendix B: Polymake script used for comparison

```
use Benchmark qw(:all);
use application 'tropical';



sub s_f1{
 my $f1 = toTropicalPolynomial("min(2x+1, x+y, 2y+1, x+z, y+z, 2z+2)",
 qw(z x y));
 my $H1 = new Hypersurface<Min>(POLYNOMIAL=>$f1);
 $H1->VERTICES;
}

sub s_f2{
 my $f2 = toTropicalPolynomial("min(2x+3, x+y, 2y+3, x+1+z, y+1+z,2z)",
 qw(z x y));
 my $H2 = new Hypersurface<Min>(POLYNOMIAL=>$f2);
 $H2->VERTICES;
}
sub s_f3{
 my $f3 = toTropicalPolynomial("min(3x+3, 2x+y+1, x+2y+1, 3y+3, 2x+1+z,
 x+y+z, 2y+1+z, x+1+2z, y+1+2z, 3z+3 )", qw(z x y));
 my $H3 = new Hypersurface<Min>(POLYNOMIAL=>$f3);
 $H3->VERTICES;
}
sub s_f4{
 my $f4 = toTropicalPolynomial("min(3x, 2x+y, x+2y, 3y, 2x+z, x+y+z,
 2y+z, x+2z, y+2z, 3z)", qw(z x y));
 my $H4 = new Hypersurface<Min>(POLYNOMIAL=>$f4);
 $H4->VERTICES;
}
```

```
sub s_f5{
 my $f5 = toTropicalPolynomial("min(x-y+2, -y+2+z, -2)", qw(z x y));
 my $H5 = new Hypersurface<Min>(POLYNOMIAL=>$f5);
 $H5->VERTICES;
}
sub s_f6{
 my $f6 = toTropicalPolynomial("min(4x+6, 3x+y+4, 2x+2y+3, x+3y+4, 4y+5,
 3x+2+z, 2x+y+z, x+2y+1+z, 3y+4+z, 2x+2+2z, x+y+2z, 2y+3+2z, x+3z, y+2+3z, 4z+5)", q
 my $H6 = new Hypersurface<Min>(POLYNOMIAL=>$f6);
 $H6->VERTICES;
}
sub s_f7{
 my $f7 = toTropicalPolynomial("min(5x+6, 4x+y+2, 3x+2y+4, 2x+3y,
 x+4y+3, 5y+8, 4x+6+z, 3x+y+4+z, 2x+2y+3+z, x+3y+4+z, 4y+5+z,
 3x+2+2z, 2x+y+2z, x+2y+1+2z, 3y+4+2z, 2x+2+3z, x+y+3z,
 2y+3+3z, x+4z, y+2+4z, 5z+5)", qw(z x y));
 my $H7 = new Hypersurface<Min>(POLYNOMIAL=>$f7);
 $H7->VERTICES;
}
sub s_f8{
 my $f8 = toTropicalPolynomial("min(6x+10, 5x+y+8, 4x+2y+6, 3x+3y+6,
 2x+4y+4, x+5y+6, 6y+9, 5x+6+z, 4x+y+2+z, 3x+2y+4+z, 2x+3y+z,
 x+4y+3+z, 5y+8+z, 4x+6+2z, 3x+y+4+2z, 2x+2y+3+2z, x+3y+4+2z,
 4y+5+2z, 3x+2+3z, 2x+y+3z, x+2y+1+3z, 3y+4+3z, 2x+2+4z,
 x+y+4z, 2y+3+4z, x+5z, y+2+5z, 5z+10)", qw(z x y));
 my $H8 = new Hypersurface<Min>(POLYNOMIAL=>$f8);
 $H8->VERTICES;
}

sub myBenchmark{
 my $t1=Benchmark::timeit(1,"s_f1");
```

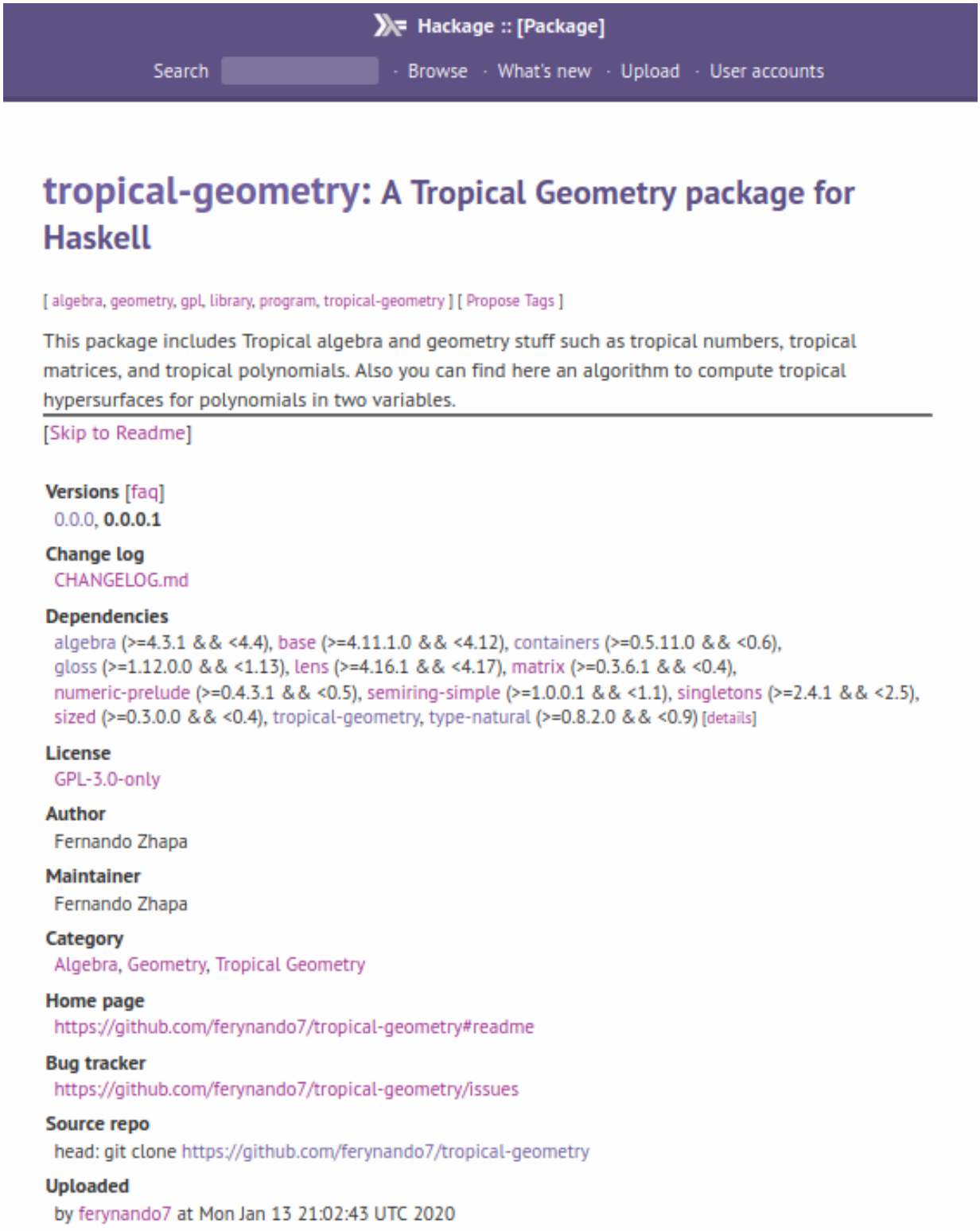
```
 print timestr($t1) . "\n";
 my $t2=Benchmark::timeit(1,"s_f2");
 print timestr($t2) . "\n";
 my $t3=Benchmark::timeit(1,"s_f3");
 print timestr($t3) . "\n";
 my $t4=Benchmark::timeit(1,"s_f4");
 print timestr($t4) . "\n";
 my $t5=Benchmark::timeit(1,"s_f5");
 print timestr($t5) . "\n";
 my $t6=Benchmark::timeit(1,"s_f6");
 print timestr($t6) . "\n";
 my $t7=Benchmark::timeit(1,"s_f7");
 print timestr($t7) . "\n";
 my $t8=Benchmark::timeit(1,"s_f8");
 print timestr($t8) . "\n";
}
```

Appendix C: Upload to Hackage

The current package has already been put into the Hackage repository. This means that any project in Haskell can import the `tropical-geometry` package in a standard way. The link for getting to it is http://hackage.haskell.org/package/tropical-geometry.

We present in the following figure a screenshot of the front page of the package in Hackage. This frontpage includes the following information:

- A description of the package
- A list of all the uploaded versions.
- A list of the dependencies, each one with the bounds on their version.
- It is not seen in the figure but the front page also includes the documentation generated for the package.



Hackage :: [Package]

Search · Browse · What's new · Upload · User accounts

tropical-geometry: A Tropical Geometry package for Haskell

[algebra, geometry, gpl, library, program, tropical-geometry] [Propose Tags]

This package includes Tropical algebra and geometry stuff such as tropical numbers, tropical matrices, and tropical polynomials. Also you can find here an algorithm to compute tropical hypersurfaces for polynomials in two variables.

[Skip to Readme]

Versions [faq]
0.0.0, **0.0.0.1**

Change log
CHANGELOG.md

Dependencies
algebra (>=4.3.1 && <4.4), base (>=4.11.1.0 && <4.12), containers (>=0.5.11.0 && <0.6), gloss (>=1.12.0.0 && <1.13), lens (>=4.16.1 && <4.17), matrix (>=0.3.6.1 && <0.4), numeric-prelude (>=0.4.3.1 && <0.5), semiring-simple (>=1.0.0.1 && <1.1), singletons (>=2.4.1 && <2.5), sized (>=0.3.0.0 && <0.4), tropical-geometry, type-natural (>=0.8.2.0 && <0.9) [details]

License
GPL-3.0-only

Author
Fernando Zhapa

Maintainer
Fernando Zhapa

Category
Algebra, Geometry, Tropical Geometry

Home page
https://github.com/ferynando7/tropical-geometry#readme

Bug tracker
https://github.com/ferynando7/tropical-geometry/issues

Source repo
head: git clone https://github.com/ferynando7/tropical-geometry

Uploaded
by ferynando7 at Mon Jan 13 21:02:43 UTC 2020

Bibliografía

[1] D. Maclagan and B. Sturmfels, *Introduction to Tropical Geometry (Graduate Studies in Mathematics)*. American Mathematical Society, 2015.

[2] G. Mikhalkin, "Tropical geometry and its applications," 2006.

[3] H. Farooq, H. Z. U. Haq, M. K. Hanif, S. Javaid, and K.-H. Zimmermann, "R: Sparse tropical algebra," R Foundation for Statistical Computing, 2018. [Online]. Available: https://CRAN.R-project.org/package=tropicalSparse

[4] A. Humayun, M. Asif, and M. K. Hanif, "BTAS: A library for tropical algebra," *CoRR*, vol. abs/1701.04733, 2017. [Online]. Available: http://arxiv.org/abs/1701.04733

[5] A. N. Jensen, "Gfan, a software system for Gröbner fans and tropical varieties," Available at http://home.imf.au.dk/jensen/software/gfan/gfan.html.

[6] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 4-1-2 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2019.

[7] E. Gawrilow and M. Joswig, "`polymake`: a framework for analyzing convex polytopes," in *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, ser. DMV Sem. Birkhäuser, Basel, 2000, vol. 29, pp. 43–73.

[8] HaskellWiki, "Haskell in industry," 2019, online; accessed 12-February-2020. [Online]. Available: https://wiki.haskell.org/index.php?title=Haskell_in_industry&oldid=63179

[9] E. A. Kmett. algebra: Constructive abstract algebra. [Online]. Available: http://hackage.haskell.org/package/algebra

[10] H. Ishii, "The computational-algebra package," 2017. [Online]. Available: http://hackage.haskell.org/package/computational-algebra

[11] C. C. Pinter, *A Book of Abstract Algebra: Second Edition (Dover Books on Mathematics)*. Dover Publications, 2010.

[12] J. S. Golan, *Semirings and their Applications*. Springer Netherlands, 1999. [Online]. Available: https://doi.org/10.1007/978-94-015-9333-5

[13] D. A. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*. Springer International Publishing, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-16721-3

[14] N. Bourbaki, *Elements of Mathematics - Theory of Sets*. Addison-Wesley, 1968.

[15] ——, *Elements de Mathematique. Algebre commutative. Chapitres 5 à 7*. Springer, 2006.

[16] D. Maclagan, "Introduction to tropical algebraic geometry," 2012.

[17] G. M. Ziegler, *Lectures on Polytopes*. Springer New York, 1995. [Online]. Available: https://doi.org/10.1007/978-1-4613-8431-1

[18] A. S. Mena, *Beginning Haskell*. Apress, 2014. [Online]. Available: https://doi.org/10.1007/978-1-4302-6251-0

[19] (2018) Haskell: Introduction. [Online]. Available: https://wiki.haskell.org/Introduction

[20] B. Milewski, *Category Theory for Programmers*. Blurb, Incorporated, 2018. [Online]. Available: https://books.google.com.ec/books?id=ZaP-swEACAAJ

[21] D. Astels, *Test-Driven Development: A Practical Guide: A Practical Guide*. Prentice Hall, jul 2003.

[22] K. Beck, *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.

[23] The Haskell Package Repository. [Online]. Available: http://hackage.haskell.org/

[24] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry*. Springer Berlin Heidelberg, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-77974-2

[25] F. P. Preparata and M. I. Shamos, *Computational Geometry*. Springer New York, 1985. [Online]. Available: https://doi.org/10.1007/978-1-4612-1098-6

[26] "The Haskell Cabal: Overview." [Online]. Available: https://www.haskell.org/cabal/

[27] "The Haskell Tool Stack." [Online]. Available: https://docs.haskellstack.org/en/stable/README/

[28] J. Richter-Gebert, B. Sturmfels, and T. Theobald, "First steps in tropical geometry," 2003.