Finding Minimal Enclosing Triangles about Newton Polygons

by

Wesley Chorney

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Honours Bachelors

in the
Department of Mathematics
Faculty of Science

Approval

Name:	**Wesley Chorney**
Degree:	**Honours Bachelors (Mathematics)**
Title:	**Finding Minimal Enclosing Triangles about Newton Polygons**
Examining Committee:	**Nathan Ilten** Supervisor Assistant Professor
	Jonathan Jedwab Internal Examiner Professor
Date Defended:	**December 14, 2018**

Abstract

Knowing a minimal enclosing triangle about a Newton polygon satisfying a certain criterion leads to some interesting algebro-geometric information. We determine which of a large family of Newton polygons support Laurent polynomials vanishing to multiplicity m at $(1,1)$, where m is at least the square root of twice the area of a minimal enclosing triangle about the polygon. To date, the computational methods we use have not been implemented on large families of Newton polygons. We use these methods in conjunction with a new dataset that classifies a large number of Newton polygons. By combining existing algorithms and using the dataset, we classify which polygons have the property of interest for genus lesser than or equal to 18. Besides providing a large amount of algebro-geometric information, these results could point to structural properties that will stimulate new research in algebraic geometry.

Keywords: Newton Polygon, Minimal Enclosing Triangle, Weighted Projective Plane, Negative Curve

The things I know, anyone can know — but my heart is mine and mine alone.
Werther, *The Sorrows of Young Werther*

Acknowledgements

The help, support, and enriching conversations from family and friends have all been a great motivator in completing this project. I would especially like to express gratitude to my parents for their continued support.

This project could not have been completed (or indeed, begun) without the aegis of my supervisor, Nathan Ilten, whose guidance throughout the entire process has been of key importance; to him I am very grateful.

Finally, I would like to thank Jonathan Jedwab, whose careful eye and insightful comments helped shape this document into what it is.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Computation and Data in Mathematics

The advent of computation has served to advance many areas of mathematics. In the past, the generation or processing of large amounts of data were computationally infeasible, yet with automatic processes, more data can be generated on mathematically interesting objects than ever before. Computers, coupled with datasets and instructions on how to process them, can efficiently produce important data for a variety of mathematical fields. Even areas as seemingly far removed from computational methods as algebraic geometry can benefit greatly from the efficient generation of data. For example, Gröbner bases transformed some algebro-geometric questions, such as ideal membership and automatic geometric theorem proving, into algorithmic processes [3].

One specific example where computational methods can be applied productively to algebraic geometry is the problem of finding blowups of weighted projective planes that contain a certain class of negative curves. By automating a procedure derived from [6], we are able to determine interesting algebro-geometric information by purely computational methods. Note that this is simply motivation for the work to follow; while these notions will be clarified in Chapter 5, it is not necessary to understand them in order to follow the rest of this document.

The given procedure transforms a complicated question into one that can be easily understood, and which lends itself well to automation. Our problem is to classify Newton polygons supporting a polynomial vanishing to multiplicity m at the point $(1,1)$, where m is at least the square root of twice the area of the minimal enclosing triangle of the polygon.

We will explain this problem in more detail in what follows. See Figure 1.1 for an example of a minimal enclosing triangle about a polygon.

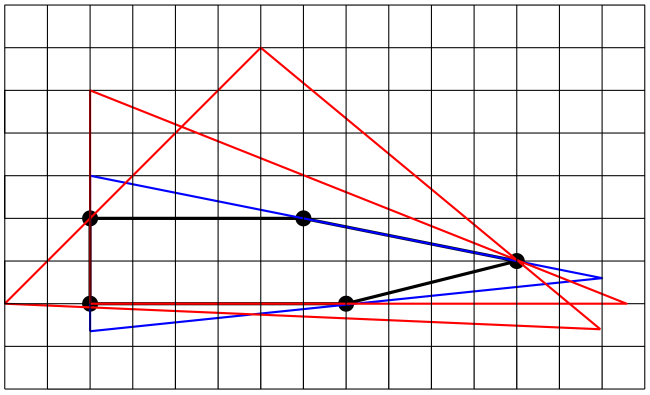


Figure 1.1: Many triangles enclose the given polygon, but the blue one has minimal area.

1.2 The Central Algorithm

From a very high-level perspective, we implement two filters which narrow down a dataset. We encourage the reader to refer to Figure 1.2 and keep it in mind while reading about the details of our process.

We give some intuitive definitions in what follows, so that the process we use can be easily understood. See Section 2.1 for formal definitions.

Let $f = \sum_{i,j} a_{i,j} x^i y^j$ be a bivariate polynomial. The *Newton polygon* of f can be thought of as being obtained by taking all points (i, j) in the plane corresponding to nonzero monomials in f, and then taking the outline that these points form (and everything enclosed therein). Notice that different polynomials can give the same polygon — the constants, as well as some monomials, can vary and still produce the same Newton polygon. For instance, the polynomials

$$f = 1 + x^2y + xy^2$$
$$g = 3 + 5xy - 7x^2y + xy^2$$

both have the same Newton polygon, which is depicted in Figure 1.3.

The *multiplicity* of a polynomial f at $(0, 0)$ is the total degree of its smallest monomial term (ordered with respect to degree). For instance, if $f = x+y+y^2$, then f has multiplicity 1 at $(0, 0)$. To obtain the multiplicity of f at a point (a, b) we set $g = f(x - a, y - b)$, and calculate the multiplicity of g at $(0, 0)$.

For computational simplicity, we may set $g(x, y) = f(x + a, y + b)$, bringing us to the point $(-a, -b)$. Then, the multiplicity of $h(x, y) = g(-x, -y)$ at $(0, 0)$ is the multiplicity of f at (a, b). Note that g and h have the same Newton polygon, and the same multiplicity at $(0, 0)$, so that without loss of generality, we may work with $f(x + a, y + b)$ instead.

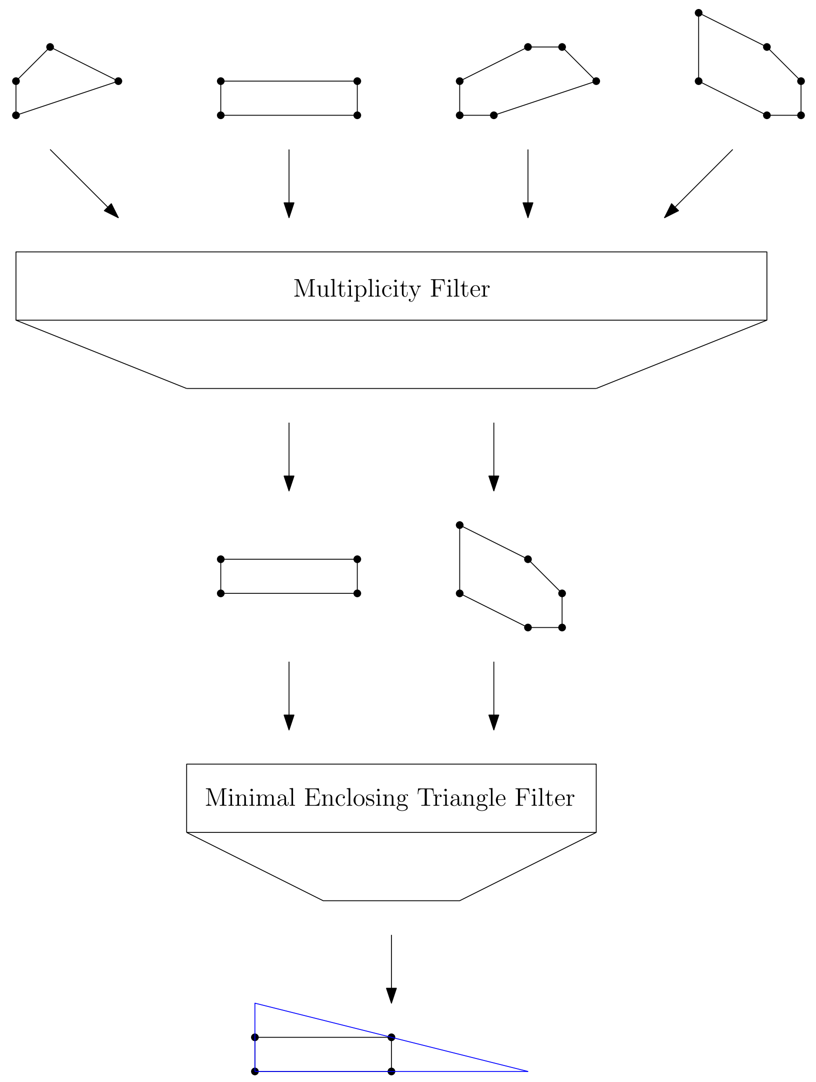


Figure 1.2: A visual depiction of the procedure that forms the cornerstone of this project.

Throughout the rest of this document, when we talk about calculating the multiplicity of $f(x, y)$ at (a, b), we will really be computing at the point $(-a, -b)$.

By *minimal enclosing triangle* about a Newton polygon, we mean a triangle, minimal with respect to area, completely containing the Newton polygon.

We now know enough to understand the process that forms the core of this document. We start with a polygon P whose vertices and interior lattice points are $v_1, \ldots, v_N$. Let $v_i = (u_i, w_i)$. We want to consider all possible polynomials that P *supports* — by which we mean we want to consider all polynomials that have P as their Newton polygon. We can write all of these as

$$f = a_{u_1,w_1} x^{u_1} y^{w_1} + \cdots + a_{u_N,w_N} x^{u_N} y^{w_N}.$$

where coefficients in front of monomials corresponding to vertices of P must be nonzero.

Next, we shift f to the point $(1,1)$ — that is, we set $g = f(x+1, y+1)$. Denote by $A(P)$ the area of P. We want to find the maximum m such that $\sqrt{2A(P)} \leq m \leq \deg(f)$, while the Newton polygon of g is identical to that of f. If we write $g = \sum_{i,j} b_{i,j}x^iy^j$, then this amounts to requiring $b_{i,j} = 0$ whenever $i+j < m$, and at least one $b_{i,j} \neq 0$ when $i+j = m$. We are interested in the cases where, while the required $b_{i,j}$ are equal to zero, the Newton polygon of g is identical to P.

For each polygon P satisfying the above criterion, we want to find a triangle Δ of smallest area, with rational coordinates, such that Δ encloses P. If $\sqrt{2A(\Delta)} \leq m$, then this implies the existence of a certain class of negative curve on a blowup of a weighted projective plane — see Chapter 5 for more details.

Example 1. *Consider the polygon described by $(0,0), (2,1), (1,2)$, shown in Figure 1.3. It supports polynomials of the form*

$$f = a_{0,0} + a_{1,1}xy + a_{2,1}x^2y + a_{1,2}xy^2$$

We set

$$\begin{aligned} g &= f(x+1, y+1) \\ &= (a_{0,0} + a_{1,1} + a_{1,2} + a_{2,1}) + (2a_{2,1} + a_{1,1} + a_{1,2})x + (a_{2,1} + a_{1,1} + 2a_{1,2})y \\ &+ (2a_{2,1} + a_{1,1} + 2a_{1,2})xy + a_{2,1}x^2 + a_{1,2}y^2 + a_{2,1}x^2y + a_{1,2}xy^2 \end{aligned}$$

Now, we want to check if the Newton polygon of g is equal to P when we force g to have certain multiplicities. In particular, we are interested in multiplicities m such that $\sqrt{2A(P)} = \sqrt{3} \leq m \leq 3$. Thus, we are interested in the cases $m = 2, 3$. When $m = 2$, we must have that

$$\begin{aligned} a_{0,0} + a_{1,1} + a_{2,1} + a_{1,2} &= 0 \\ 2a_{2,1} + a_{1,1} + a_{1,2} &= 0 \\ a_{2,1} + a_{1,1} + 2a_{1,2} &= 0 \end{aligned}$$

One can verify (for example, by row-reducing a coefficient matrix) that none of the $a_{i,j}$ are forced to be zero, so there exist choices of $a_{i,j}$ for which the Newton polygon of g is equal to P when g has multiplicity 2. Now, since P is itself a triangle, $\Delta = P$, and we are done. When $m = 3$, we have more than four equations, four of which are linearly independent, so $a_{i,j} = 0$ for each i, j; so in this case we do not need to move on to the second step of the process and find the smallest triangle enclosing P.

In order to automate this procedure, we require a few steps. First, we need a sufficiently large list of Newton polygons. Fortunately, Castryck in [2] classifies Newton polygons of

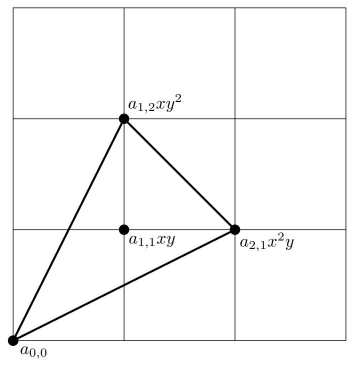


Figure 1.3: Newton polygon in Example 1.

genus lesser than or equal to 30 up to integer linear transformation. Here, genus refers to the number of interior points. This is a list of roughly 4,800,000 Newton polygons — not all of which were able to be processed for this project, but which can, in principle, be processed using the methods and code described herein.

Next, it is necessary to obtain the general polynomial f supported by a polygon P, and restrict it to have multiplicity m at (1,1). Then we must check if the polygon that supports this polynomial is equal to P. This is achieved in *Macaulay2*; see [7] for more details on the software.

Finally, for suitable polygons P, we find a minimal triangle Δ enclosing P, and verify that its area satisfies $\sqrt{2A(\Delta)} \leq m$. Finding the minimal enclosing triangle about a polygon P is a topic of interest in robotics, and so this has been researched extensively; see for instance [11]. In fact, an algorithm is known for finding a minimal enclosing triangle, which has been shown to be optimal with respect to runtime. It has been implemented in the publicly available OpenCV library, and so we use this to obtain our Δ [1].

Underlying all these steps is the pre-processing, intermediate processing, and final processing of the initial dataset. All of this is done in *R*. The *R* script used at each step of the process can be found in the appendices, including that used to write the final dataset.

The structure of the rest of this document is as follows. In Chapter 2, we give a mathematical explanation of our problem, and explain why our computational procedure performs the necessary mathematical operations. Then, in Chapter 3, we discuss the genus zero case (genus zero polygons must be treated separately, as they are not included in our initial dataset). Next, we give some results on computing the minimal enclosing triangle about a polygon, and see how these results impact the genus zero case. In Chapter 4, we review the code used to automate our procedure for genus greater than zero, and explain how it works in conjunction with our data. In Chapter 5, we discuss the algebro-geometric interpretation of our results, and finally, we conclude by discussing our results and questions that remain to be answered.

Chapter 2

A Mathematical Justification for the Computations

2.1 Preliminary Definitions

Herein we make precise every notion intuitively defined in the introduction. Next, we give a step-by-step description of the process we automated, and explain why, at each step, it is correct. Unless otherwise stated, $\mathbb{K}$ is a field.

Definition 2. *A* ***Laurent polynomial*** *is a polynomial in* $\mathbb{K}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$*. This is to say that it is a linear combination of positive and negative powers of variables* $x_1, \ldots, x_n$ *with coefficients* $a_i \in \mathbb{K}$*.*

Laurent polynomials are ubiquitous in complex analysis, and are commonplace in many other areas of mathematics. We can represent some information about bivariate polynomials geometrically by associating to them what is known as a Newton polygon. However, we first need the notion of a convex hull.

Definition 3. *A* ***convex set*** S *is a set of points such that, given any two points* a, b *in that set, the line joining them lies entirely within* S*.*

Definition 4. *Let* S *be a set of points in* $\mathbb{K}^n$*. The* ***convex hull*** *of* S *is the smallest convex set containing* S*. It is denoted* $conv(S)$*.*

In two dimensions, the convex hull can be pictured quite intuitively. Given a set S of points in the plane, each point can be thought of as a peg. Then, if one were to stretch an elastic band about all the pegs, the elastic band and its interior would be the convex hull of S.

Definition 5. *Let* f *be a Laurent polynomial in* $\mathbb{K}[x, x^{-1}, y, y^{-1}]$*. The* ***Newton polygon*** *of* f *is*

$$N(f) = conv(\{(i,j) \mid monomial\ a_{i,j}x^iy^j\, appears\ in\ f,\ \ and\ a_{i,j} \neq 0\})$$

This is to say that the Newton polygon of f is the convex hull of the set of points in $\mathbb{Z}^2$ corresponding to the powers of x and y in nonzero monomial terms in f. For example, if $f = 1 + x + y$, $N(f)$ is the triangle with vertices $(0,0), (1,0), (0,1)$. For more examples, see Figure 2.1. If f has Newton polygon P, we say that P *supports* f. While Newton polygons are independently interesting, they are but one part of the process described in [1].

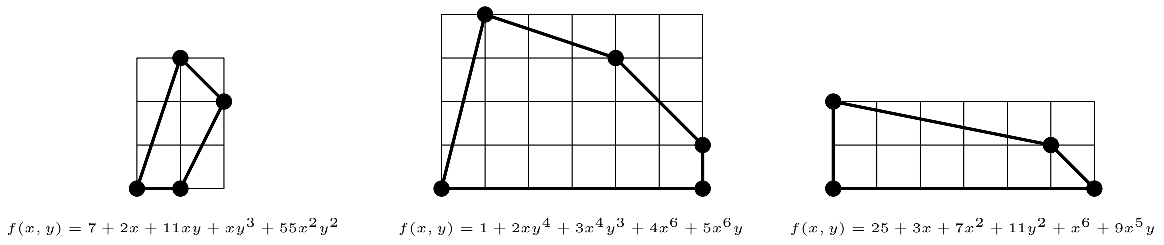

$f(x, y) = 7 + 2x + 11xy + xy^3 + 55x^2y^2$
$f(x, y) = 1 + 2xy^4 + 3x^4y^3 + 4x^6 + 5x^6y$
$f(x, y) = 25 + 3x + 7x^2 + 11y^2 + x^6 + 9x^5y$

Figure 2.1: Some polynomials and their Newton polygons.

Definition 6. *Let f be a Laurent polynomial in $\mathbb{K}[x, x^{-1}, y, y^{-1}]$. Write $f = F_r + \cdots + F_s$, where each F_i term consists of all monomials in f of degree i, and $r < \cdots < s$. The* ***multiplicity*** *of f at $(0,0)$ is $m(f) = r$. The multiplicity of f at a point (a,b) is the multiplicity of $g = f(x-a, y-b)$ at $(0,0)$.*

Again, recall that throughout the remainder of this document, we will use $g = f(x + a, y + b)$ to calculate the multiplicity of f at (a,b). The definition of multiplicity coincides with the notion of the degree to which a root vanishes at a point in the one-dimensional case — see Figure 2.2.

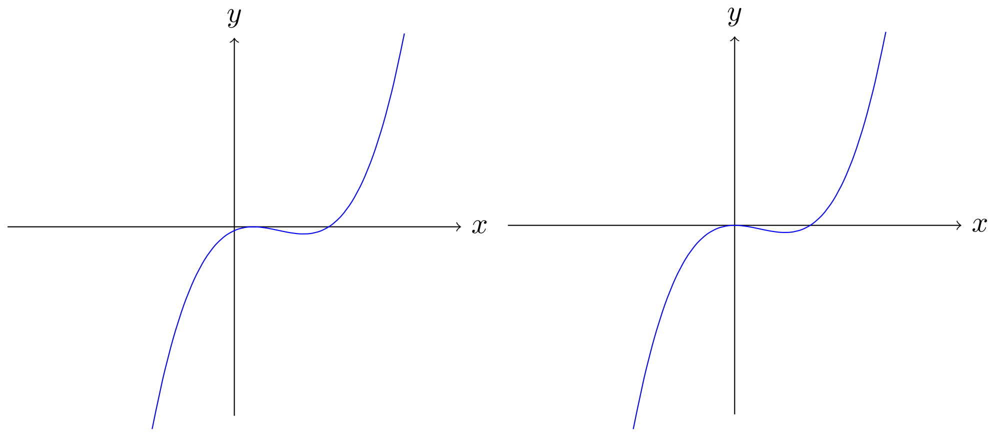


Figure 2.2: The left graph is $f(x) = (x - 1/5)^2(x-1)$, which should have multiplicity 2 at $-1/5$. Indeed, $f(x + 1/5) = x^3 + (4/5)x^2$, which is the right graph, has multiplicity 2 at 0.

2.2 The Algorithm — Mathematically

Recall that we want to find a minimal triangle Δ enclosing those Newton polygons that support a Laurent polynomial vanishing to multiplicity m at $(1,1)$, and check if $\sqrt{2(A(\Delta)} \leq m$.

We begin with a Newton polygon P. From the definition, each point $(i,j) \in \mathbb{Z}^2 \cap P$ corresponds to a monomial term in a polynomial f which P supports. The term (i,j) corresponds to a term that must be nonzero if it is the intersection of two of the edges of P. We will use $\mathcal{S}$ to denote the set of such terms. We can represent terms generally in f as $a_{i,j}x^iy^j$, where $a_{i,j} \in \mathbb{K}$ (unless $(i,j) \in \mathcal{S}$, in which case $a_{i,j} \in \mathbb{K}\backslash 0$). Thus, in general, P supports Laurent polynomials of the form

$$f = \sum_{((i,j)\in\mathbb{Z}^2\cap P)\backslash\mathcal{S}} a_{i,j}x^iy^j + \sum_{(r,s)\in\mathcal{S}} a_{r,s}x^ry^s$$

However, letting c (respectively d) denote the largest x (respectively y) coordinate P contains, it will be better to think of f as having coefficients in the ring $\mathbb{K}[a_{0,0},\ldots,a_{c,d}]$ — since we do not restrict any $a_{i,j}$ to a specific value, it is best to think of them as indeterminates. With this in mind, we may simply write $f = \sum_{(i,j)\in\mathbb{Z}^2\cap P} a_{i,j}x^iy^j$.

Now, we want to check if a minimal enclosing triange of the Newton polygon of f, Δ, satisfies $\sqrt{2A(\Delta)} \leq m$, where f vanishes to multiplicity m at $(1,1)$. Since Δ encloses P, we know that $A(P) \leq A(\Delta)$. Thus, we can force f to vanish at multiplicities ranging from $\left\lceil \sqrt{2A(P)} \right\rceil$ to $\deg(f)$ at $(1,1)$.

We set $g = f(x+1, y+1)$. Now we need to force g to vanish to multiplicity m at $(0,0)$. Each monomial term x^ky^l in g has as its coefficient a linear sum of $a_{i,j}$, for every (i,j) with both $k \leq i$ and $l \leq j$. That is,

$$g = \sum_{(k,l)\in\mathbb{Z}^2\cap P} \left(\sum_{(i,j)\geq(k,l)} \binom{i}{k}\binom{j}{l} a_{i,j} \right) x^ky^l$$

where we treat the $a_{i,j}$ as formal variables. The binomial coefficients $\binom{i}{k}\binom{j}{l}$ arise since when $i \geq k$ and $j \geq l$, we have i copies of the $(x+1)$ term from $a_{i,j}(x+1)^i(y+1)^j$ and we must choose k of them from which we take an x, and similar for y.

Clearly, we could force g to vanish with multiplicity m at $(0,0)$ by setting $a_{i,j} = 0$ for every i and j such that $i + j < m$. However, notice that the zero-degree term of g is simply $\sum_{i,j} a_{i,j}$. Thus, we would set every coefficient in f to be zero, rendering f identically zero. In this case, f is not supported by P. We instead seek non-trivial solutions for the $a_{i,j}$.

We can obtain such solutions with some straightforward algebra. For each monomial x^iy^j such that $i + j < m$, add its coefficient to an ideal I. By doing so, we are setting sums of $a_{k,l}$ to be zero, not necessarily forcing each individual coefficient to be zero. Next, for each $(r,s) \in \mathcal{S}$, we check whether $a_{r,s}$ modulo I is nonzero. That is, we check if we can force each monomial term in g of degree less than m to vanish, without forcing any of the $a_{r,s}$ to be zero (since if any of these were zero, f would no longer be supported by P).

Alternatively, for each monomial x^iy^j in g with $i+j<m$, we can think of its coefficient as a vector in $\mathbb{K}$. For instance, if g has term $(a_{1,0}+2a_{0,1}+6a_{1,1}+a_{2,1})x^2y$ in the ring $\mathbb{K}[a_{0,0},a_{1,0},a_{0,1},a_{1,1},a_{2,1}]$, then we obtain the vector $\langle 0,1,2,6,1\rangle$. Obtaining a vector for each monomial x^iy^j in g with $i+j<m$, we may form a matrix M, which we put in row-reduced echelon form. If a row of M has a 1 in any (r,s) position for $(r,s)\in\mathcal{S}$ and zeroes everywhere else, then $a_{r,s}=0$ and P does not support f. Otherwise, P supports f, which vanishes to the required multiplicity at $(1,1)$.

Example 7. *We revisit the example of Chapter 1, but we use the methods described above. Recall, we set*

$$\begin{aligned} g &= f(x+1,y+1)\\ &= (a_{0,0}+a_{1,1}+a_{1,2}+a_{2,1})+(2a_{2,1}+a_{1,1}+a_{1,2})x+(a_{2,1}+a_{1,1}+2a_{1,2})y\\ &+(2a_{2,1}+a_{1,1}+2a_{1,2})xy+a_{2,1}x^2+a_{1,2}y^2+a_{2,1}x^2y+a_{1,2}xy^2\end{aligned}$$

which, when $m=2$, gave the equations

$$\begin{aligned} a_{0,0}+a_{1,1}+a_{2,1}+a_{1,2}&=0\\ 2a_{2,1}+a_{1,1}+a_{1,2}&=0\\ a_{2,1}+a_{1,1}+2a_{1,2}&=0\end{aligned}$$

We can represent each of these as vectors, with the same coefficient order as above.

$$\begin{gathered}\langle 1,1,1,1\rangle\\ \langle 0,2,1,1\rangle\\ \langle 0,1,1,2\rangle\end{gathered}$$

We put these vectors into a matrix and row-reduce it, obtaining

$$\begin{pmatrix} 1 & 0 & 0 & -1\\ 0 & 1 & 0 & -1\\ 0 & 0 & 1 & 3\end{pmatrix}$$

We can see that there exist choices for $a_{i,j}$ such that no coefficient is forced to be zero, so we can force f to have multiplicity 2 at $(1,1)$ while retaining the same Newton polygon. Alternatively, we create the ideal

$$I=(a_{0,0}+a_{1,1}+a_{2,1}+a_{1,2},2a_{2,1}+a_{1,1}+a_{1,2},a_{2,1}+a_{1,1}+2a_{1,2})$$

and check for $a_{0,0}, a_{2,1}, a_{1,2}$ *that none of these are equivalent to 0 modulo* I. *From the matrix above, we see that*

$$I = (a_{0,0} - a_{1,2}, a_{2,1} - a_{1,2}, a_{1,1} + 3a_{1,2})$$

so in particular, no coefficient will be zero modulo I.

Now, for each $\left\lceil \sqrt{2A(P)} \right\rceil \leq m \leq \deg(f)$, we check whether f vanishes to multiplicity m at (1,1). If this holds for any m, we find the minimal triangle enclosing P. Fortunately, plenty of work has been done on this topic, and it exists as a publicly-available algorithm in the OpenCV library — see for instance [11]. Next, we use a computer to check that the area restriction is met. If so, we are done.

Chapter 3

Starting from the Bottom: Genus Zero

3.1 Introduction

In [2], Castryck classifies Newton polygons of genus $1 \leq g \leq 30$, up to integer linear transformation, and gives an algorithm for classifying those with higher genus. However, we are also interested in the genus zero case — fortunately, these are also relatively straightforward. Up to integer linear transformation, they are of the form pictured in Figure 3.1. Ignoring the triangle (described by the points $(0,0), (2,0), (0,2)$), we may describe them up to equivalence with four vertices, and without loss of generality, we may assume they are of the form $(0,0), (0,1), (k,0), (l,1)$, where $k, l \geq 1$ and also $k \geq l$ (since this is always achievable with an integer linear transformation) — see Proposition 8.

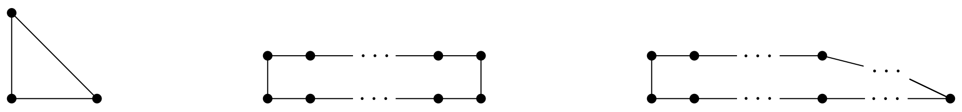


Figure 3.1: Genus zero Newton polygons.

Proposition 8. *Up to integer linear transformation, genus zero polygons may be described with the points* $(0,0), (0,1), (k,0), (l,1)$ *where* $k, l \geq 1$ *and* $k \geq l$*; or as a triangle with vertices* $(0,0), (2,0), (0,2)$.

Proof. Let P be a genus zero polygon. By E we will denote the edge of P with longest lattice length. After integer linear transformation, we may always assume this is the line segment connecting the points $(0,0), (k,0)$, and we may assume further that P lies in the upper half-plane.

For each $i > 0$, let l_i denote the length of the line segment $y = i$ intersected with P (see Figure 3.2). Let m be the maximal index such that l_m is nonzero. If any $l_i > 1$ when $i \neq 0, m$, then P must have an interior lattice point. By convexity, if $k > 1$ and $m > 2$, P

will have an interior lattice point (this forces $l_1 > 1$). Similarly, if we have $k > 1$ and $m = 2$, while either $l_m > 0$ or $k > 2$, we again have an interior lattice point, since $l_1 > 1$.

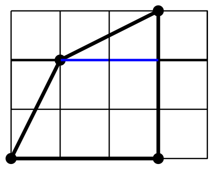


Figure 3.2: The blue line is l_2.

It follows that we must have $k = 2$ with $l_m = 0$ and $m = 2$ and $l_1 = 1$, or $m = 1$, or $k = 1$.

In the first case, P is a triangle. Up to integer linear transformation, there are two possibilities — one is the triangle described by $(0,0),(2,0),(0,2)$, and the other has an interior lattice point.

If $m = 1$, then, up to integer linear transformation, P is of the form $(0,0),(0,1),(k,0),(l,1)$, where $k,l \geq 1$, and $k \geq l$.

Finally, if $k = 1$ and $m > 1$, let v be any vertex of P with height m. Consider the triangle with vertices $(0,0),(1,0),v$. This triangle is contained in P, and its edges must have no interior lattice points, since otherwise P would have an interior lattice point or an edge longer than E. If this is the case, then by Pick's theorem, the area of this triangle is $1/2$ — but we required that $m > 1$, which is a contradiction. $\square$

Since we know what a minimal enclosing triangle about a triangle is, we may immediately deal with one genus zero case.

Proposition 9. *Let P be the genus zero polygon described by the points $(0,0),(2,0),(0,2)$, and T its minimal enclosing triangle. Then P supports a Laurent polynomial vanishing to multiplicity 2 at $(1,1)$ with $\sqrt{2A(T)} \leq 2$.*

Proof. Note that $A(P) = 2$, so $\sqrt{2A(P)} = 2$. Therefore, $m \geq 2$. However, P supports polynomials of the form

$$f(x,y) = a_{0,0} + a_{1,0}x + a_{2,0}x^2 + a_{0,1}y + a_{1,1}xy + a_{0,2}y^2$$

so since $m \leq \deg(f)$, we require $m = 2$.

Setting $g = f(x+1, y+1)$ and requiring coefficients in front of terms with degree less than 2 gives us the following system of equations:

$$\begin{aligned} a_{0,0} + a_{1,0} + a_{2,0} + a_{0,1} + a_{1,1} + a_{0,2} &= 0 \\ a_{1,0} + 2a_{2,0} + a_{1,1} &= 0 \\ a_{0,1} + a_{1,1} + 2a_{0,2} &= 0 \end{aligned}$$

In row-reduced matrix form, we have

$$\begin{pmatrix} 1 & 0 & -1 & 0 & -1 & -1 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 2 \end{pmatrix}$$

Since there exist choices for the $a_{i,j}$ such that no $a_{r,s}$ corresponding to the vertices of P are zero, P supports f vanishing to multiplicity 2 at $(1,1)$, and its minimal enclosing triangle T satisfies $\sqrt{2A(T)} \leq 2$. $\square$

3.2 Admissible Parameters

We may immediately dispense with some cases, as the following proposition demonstrates.

Proposition 10. *Let P be a genus zero Newton polygon described as above, which supports f. Suppose $k > l^2 + l + 1$. Then for any $\left\lceil \sqrt{2A(P)} \right\rceil \leq m \leq \deg(f)$, the Newton polygon supporting f vanishing to multiplicity m at $(1,1)$ has zero area.*

Proof. Suppose $m > l+1$. In this case, the coefficient in front of the term $x^l y$ in $f(x+1, y+1)$ must be zero. However, notice that after expanding $f(x+1, y+1)$, the coefficient in front of $x^l y$ is $a_{l,1}$, which must be zero. Furthermore, the coefficient in front of $x^{l-1}y$ in $f(x+1, y+1)$ is $(a_{l-1,1} + \binom{l}{l-1}a_{l,1})$, and this must be zero as well. Since $a_{l,1} = 0$, we have that $a_{l-1,1} = 0$. In general, the coefficient in front of the term $x^{l-i}y$ will be $\sum_{j=l-i}^{l} \binom{j}{l-i} a_{j,1}$; and via the same reasoning as above, we will have that $a_{l,1} = a_{l-1,1} = \cdots = a_{l-i+1,1} = 0$, and so we conclude that $a_{l-i,1} = 0$ as well, for $0 \leq i \leq l$.

Therefore, we care only about the cases where $m \leq l + 1$. We also must have that $2A(P) \leq m^2$; and from Pick's theorem, $2A(P) = k + l$.

Thus, for non-triviality, we require $k + l \leq (l+1)^2$, or that $k \leq l^2 + l + 1$. $\square$

3.3 Results on Minimal Enclosing Triangles and Implications

We would now like to know something about the triangles of minimal area which enclose genus zero polygons, to see if we can determine for which multiplicities the bound is satisfied. To do so, we state the following (reworded) result due to O'Rourke et al., which simplifies the search for minimal enclosing triangles [10]:

Theorem 11. *If a triangle T is a local minimum (with respect to area) among triangles enclosing a polygon P, then there exists a triangle T' of equal area to T, also enclosing P, such that at least two edges of T' coincide with P, and the midpoint of the third edge of T' touches P.*

This is the result upon which the algorithm of [10] is built. We may create equivalence classes of triangles enclosing P based on having equal area. The above result guarantees that the representatives of all classes of area $A < \alpha$ (for sufficiently small α) are triangles with two edges coinciding with P, and with the midpoint of the third edge touching P. To find the global minimum, we simply select a representative of the equivalence class corresponding to smallest area.

In fact, this renders the genus zero case nearly trivial. Since we are dealing with polygons that have four edges (and in one case, a great deal of symmetry), we have but a few cases to investigate. In fact, the following lemmata settle the genus zero case entirely.

Lemma 12. *Let P be a genus zero Newton polygon that supports f vanishing to multiplicity m at $(1,1)$, described by the points $(0,0),(0,1),(l,0),(l,1)$; and T its minimal enclosing triangle. If $\lceil 2\sqrt{l}\rceil \leq m \leq \deg(f)$, then P supports a Laurent polynomial vanishing to multiplicity m at $(1,1)$ satisfying the area restriction; that is, $\sqrt{2A(T)} \leq m$.*

Proof. From Theorem 11, the minimal enclosing triangle T of P has two edges flush with those of P. Without loss of generality, we may take these to be the edges between $(0,0),(0,1)$, and $(0,0),(l,0)$. The third edge of T must have midpoint touching P, and since it must form a triangle, it will touch P at $(l,1)$, and meet the other lines of T at $(0,2)$, and $(2l,0)$ — see for an example Figure 3.3. Now, this triangle has area $2l$, and we require $\sqrt{2A(T)} \leq m$, and so the result follows. □

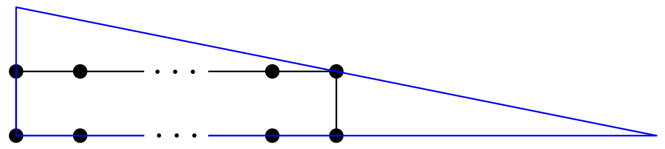


Figure 3.3: The triangle of minimal area enclosing P in Lemma 12.

Lemma 13. *Let P be a genus zero Newton polygon that supports f vanishing to multiplicity m at $(1,1)$, described by the points $(0,0),(0,1),(k,0),(l,1)$, where $l < k$, and T its minimal enclosing triangle. If $\min\left(\left\lceil \frac{k}{\sqrt{k-l}}\right\rceil, \left\lceil 2\sqrt{l}\right\rceil\right) \leq m \leq \deg(f)$, then P supports a Laurent polynomial vanishing to multiplicity m at $(1,1)$ satisfying the area restriction; that is, $\sqrt{2A(T)} \leq m$.*

Proof. We need to find the area of triangles T with edges flush with at least two the edges of P. Since P has four edges, it is sufficient to check six cases.

Throughout, we will assume labelling for P as in Figure 3.4.

The first case is where T has edges flush with edges 1 and 3 of P. However, we also require that the third edge of T touches P at its midpoint. This can only happen if it is flush with edge 4. In this case, T has points $(0,0),(k,0),(0,\frac{k}{k-l})$, since the third point lies on the line $y = \frac{1}{l-k}(x-k)$ and must intersect meet the y-axis. Thus, $A(T) = \frac{k^2}{2(k-l)}$, and we want $\sqrt{2A(T)} = \frac{k}{\sqrt{k-l}} \leq m$.

The second case is where T has edges flush with edges 1 and 4 of P. If $l \leq \frac{1}{2}k$, then we are back in the first case, so we assume $l > \frac{1}{2}k$. In this case, the third edge must touch the point $(l, 1)$ at its midpoint. Since one point on this line must be of the form $(a, 0)$ (because it intersects the line flush with $(0, 0)$ and $(k, 0)$), and the other $(0, b)$ (for similar reasons), we see that $a = 2l$ and $b = 2$. Thus, $A(T) = 2l$, and we want $\sqrt{2A(T)} = 2\sqrt{l} \leq m$.

The third case is where T has edges flush with edges 1 and 2 of P. This is almost identical to case 2, except that instead of $2l$, we will have $A(T) = 2k$. However, $k > l$, so this triangle will never be minimal with respect to area.

The fourth case is where T has edges flush with edges 2 and 3 of P. In this case, the third edge of T must touch the point $(0, 0)$ at its midpoint. Two points on the third line will be of the form $p_1 = (c, 1)$ and $p_2 = (d, \frac{d-k}{l-k})$. The form of the first point is because it must lie on the line of T flush with edge 2 of P, and the formula for the second point, because it must lie on the line flush with edge 3 of P. This line is $y = \frac{x-k}{l-k}$. Now, for the edge between these two points to have midpoint $(0, 0)$, we must have $c + d = 0$, or that $d = -c$. In particular, the y-coordinate of p_2 is necessarily negative, so $\frac{d-k}{l-k} = \frac{c+k}{k-l} < 0$, and since $k > l$, it must be the case that $|c| > k$. Since p_1 has a y-coordinate of 1, and the midpoint has y-coordinate 0, $\frac{d-k}{l-k} = -1$. Thus the resulting triangle has a height of 2, and base greater than $2k$, so it will not be minimal among all enclosing triangles.

The fifth case is where T has edges flush with edges 2 and 4 of P. In this case, we have two parallel lines, so we cannot form a triangle.

The sixth and final case is where T has edges flush with edges 3 and 4 of P. If $l \leq \frac{1}{2}k$, we are in case 1, so suppose $l > \frac{1}{2}k$. The third line must have midpoint $(0, 1)$. In this case, the third line must be between points of the form $p_1 = (2l - k, 2)$ and $p_2 = (e, 0)$. This is due to the fact that p_1 lies on the line $y = \frac{x-k}{l-k}$ and must have y-coordinate equal to 2 to achieve midpoint $(0, 1)$. Now, to achieve an x-coordinate of 0, it must be the case that $e = k - 2l$. In this case, T has base $2l$ and height 2, so $A(T) = 2l$, as in case 2. $\square$

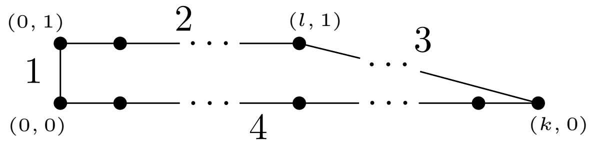


Figure 3.4: P as in the proof of Lemma 13.

Now that we have a better understanding of what the genus zero case looks like, we can move on to the cases for Newton polygons with genus greater than zero. In these cases, it is not so simple, since there is much more variation in terms of the possible polygons we can obtain. It is at this point that we must turn to computation.

Chapter 4

Code and Computation: Genus Greater than Zero

4.1 Introduction

Newton polygons with genus greater than zero cannot be classified as simply as those with genus zero. Fortunately, a classification (up to integer linear transformation) does exist, due to Castryck in [2]. Our first step is to change the way this data is presented, so that it is easily readable by the computer algebra software *Macaulay2*. Next, we automate the process described in Section 2 using *Macaulay2*. From this, we obtain a list L_1 of polygons which support a Laurent polynomial vanishing to multiplicity $\left\lceil \sqrt{2A(P)} \right\rceil$ at $(1,1)$. We use the *minEnclosingTriangle* algorithm in the OpenCV library to obtain vertices of the minimal enclosing triangle T of each polygon. For each polygon in L_1, we then check for which multiplicities $\left\lceil \sqrt{2A(P)} \right\rceil \leq m \leq \deg(f)$ the polygon supports a Laurent polynomial f vanishing to m at $(1,1)$, obtaining a list L_2 of polygons and the multiplicities for which this holds. Finally, we verify that $\sqrt{2A(T)} \leq m$ for each polygon in L_2, and create a final list L_3 containing the polygons, their genera, the largest multiplicity m for which $\sqrt{2A(T)} \leq m$, and whether equality is achieved in the latter.

We now review the code and describe what it is doing.

4.2 Data Preprocessing

The following code preprocesses the dataset from [2]. It assumes only that this dataset has been saved as a text file (originally it is an m-file) and that it is in the present working directory.

```
createList <- function() {
        #We assume the pre-processed polygons1to30.txt
        #file is in the pwd. We also include the
        #required packages.
```

```
library(gdata)
t <- scan("./Polygons1to30.txt", what="character", fill=T
    ↪ , sep="\n")

#Every line corresponding to a valid polygon
# begins with "{ {", we search for these.
polyt <- grepl("{␣{", t, fixed=TRUE)

#Create new list containing only Newton
#Polygons
t2 <- t[polyt]

t2 <- gsub("[", "{", t2, fixed=T)
t2 <- gsub("]", "}", t2, fixed=T)

#We also need to eliminate commas from the end
#of any line, lest Macaulay2 believes it to be
#a sequence.
for(i in seq_along(t2)) {
if(grepl("}␣},", t2[i])==TRUE) {
t2[i] <- substr(t2[i], 1, nchar(t2[i])-1)
}
}

#Convert this list into a matrix for write.fwf
t3 <- matrix(t2)

#Now, we write this to a fwf file for ease of
#use with Macaulay2's readIn function.
write.fwf(t3, file="PolygonData.txt", width="128")

}
```

The above code addresses four matters. Firstly, the original dataset organizes polygons by genus and, as a result, has some lines that are unnecessary for our purposes. We use the *grepl* function to filter the data so we get lines corresponding only to polygons, and save this to *t*2.

Secondly, *Macaulay2* begins and ends lists with curly braces, whereas the original file begins and ends lists with square braces. Thus, in our updated dataset *t*2, we search for opening and closing square braces, and replace them with opening and closing square braces, respectively.

Next, some lines end with commas. This is due to how the original dataset was organized; however, this will be an issue when read into *Macaulay2*, thus we search for lines ending with commas and remove them.

Finally, *Macaulay2*'s *read* function takes two arguments: the file to read, and the number of bytes to read. Thus, we write our final dataset as a fixed-width file. Each line is 128 bytes in length (shorter lines are filled with spaces) — we chose the 128 based on the length of the longest line, which can be obtained by running the command *nchar(t2[which.max(nchar(t2))])* in the *R* console.

4.3 First Filter: The Smallest Multiplicity

Now that we have our dataset in a format *Macaulay2* can easily understand, we begin to filter our data. Our first step is in verifying which polygons support a polynomial vanishing to multiplicity $\lceil\sqrt{2A(P)}\rceil$ (the smallest possible multiplicity). Note that the multiplicity determines how many sums of coefficients we add to our ideal, as described in Section 2. In particular, if a polygon supports a polynomial vanishing to multiplicity m at $(1,1)$ for $m > \lceil\sqrt{2A(P)}\rceil$, it will also support this polynomial when $m = \lceil\sqrt{2A(P)}\rceil$. Thus, this is a valid first step in diminishing the size of our data.

```
checkPoly=L->(
        c:=min(apply(L,a->a_0));
        d:=min(apply(L,a->a_1));
        LL:=apply(L,i->{i_0-c,i_1-d});
        P:=convexHull transpose matrix LL; -- create a polygon
            ↪ from list of vertices
        lp:=apply(latticePoints P,i->flatten entries i); -- list
            ↪ of all lattice points in P
        R:=QQ[apply(lp,i->a_i)]; -- ring with variables indexed
            ↪ by lattice points of P
         S:=R[x_0,x_1,Join=>false]; -- ring for our polynomial f
         f:=sum apply(lp,i->(a_i)*(x_0^(i_0))*(x_1^(i_1))); --
             ↪ create a general polynomial with P as newton
             ↪ polygon
         C:=coefficients sub(f,{x_0=>x_0+1,x_1=>x_1+1}); -- do
             ↪ change of variables and make coefficient matrix
         m:=ceiling sqrt (2*(volume P)); -- we need the vanishing
             ↪  multiplicity to be at least m
        I:=sub(ideal (flatten entries C_1)_(positions(flatten
            ↪ entries C_0,i->(degree i)_0<m)),R); -- conditions
            ↪ imposed on the coefficients by the multiplicity
            ↪ requirement
        not any(LL,i->R_(position(lp,j->j==i))%I==0) -- checks if
            ↪ any of the coefficients for a vertex vanishes
        )

polyList = L -> (
        myFile := openIn("./Documents/PolygonData.txt");
        currentList := read(myFile, 2);
        while 1==1 do (
                outPut := openOutAppend("./Documents/minimize2.
                    ↪ txt");
                currentList = value(read(myFile, 130));
                outPut << checkPoly(currentList) << endl;
                close(outPut);
                );
        )
```

The first function, *checkPoly*, takes as input a list of vertices that correspond to a Newton polygon. It shifts the vertices so that the smallest vertex is at $(0,0)$, after which it creates the Newton polygon from these vertices, and obtains the list of all lattice points of the polygon. Next, we specify in which ring we are working. We set $R = \mathbb{Q}[a_{\{i,j\}}]$, where $\{i,j\}$ are elements of the list of lattice points. We work over the ring $R[x_0, x_1]$ (it is this ring to which f, the general form of the polynomial which P supports, belongs).

Next, we form f. For each $\{i,j\}$ in our list of lattice points, we add as a summand in f the term $a_{\{i,j\}}x_0^i x_1^j$. Then, we calculate $f(x_0+1, y_0+1)$ and create the coefficient matrix of this polynomial. Next, we set I to be the ideal generated by the coefficients of monomial terms with degree less than m, which is $\left\lceil \sqrt{2A(P)} \right\rceil$. Finally, we check that for each vertex $a_{i,j}$ of P, $a_{i,j} \neq 0 \bmod I$. *checkPoly* thus returns a list of true/false, signifying whether we need to find the minimal enclosing triangle about the polygons in our preprocessed dataset.

The second function, *polyList*, takes as input the polygon dataset we preprocessed for ease of use with *Macaulay2*. Since R assigns a default column name of "$V1$" to our polygon data, we read past this before calling the function *checkPoly*. We also open a file to write to, which will contain those polygons supporting a Laurent polynomial vanishing to multiplicity $\left\lceil \sqrt{2A(P)} \right\rceil$ at $(1,1)$. Recall that we wrote our polygon dataset to have a width of 128 bytes per line, but we call the *read* function with 130 bytes. This is because we have to account for the \n signifying a line break at the end of each line. Finally, note that the *while* loop would run endlessly, but *Macaulay2* will throw an error when it cannot read another line of the file.

Now that we have an indicator telling us for which polygons we want to find the minimal enclosing triangle, we need to do a bit more data processing to obtain something we can feed into the minimal enclosing triangle algorithm in OpenCV. We relegate this piece of code to the appendix, since it is very similar to the first piece of code we presented.

4.4 Second Filter: The Minimal Enclosing Triangle

We now use the *minEnclosingTriangle* algorithm in the OpenCV library to find minimal enclosing triangles about the polygons we have selected in the previous step. Note that this is implemented in *C++*.

```
#include "stdafx.h"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <fstream>
#include <sstream>
#include <iostream>
using namespace cv;
using namespace std;

bool getPoint(string &full, Point &point)
{
if (full == "")
{
return false;
}

int closingBraceIndex = full.find("}");
int commaIndex = full.find(",");

point.x = stoi(full.substr(1, commaIndex - 1));
point.y = stoi(full.substr(commaIndex + 1, closingBraceIndex - (
   ↪ commaIndex + 1)));

int restIndex = closingBraceIndex + 3;
if (restIndex < full.size())
{
full = full.substr(restIndex, full.size() - restIndex);
```

```
} else {
full = full.substr(0, 0);
}

return true;
}

int main()
{

ifstream file("toMinimize.txt");

if (!file) {
cout << "unable␣to␣open␣input␣file";
exit(1);
}

ofstream outputFile;

if (!outputFile) {
cout << "unable␣to␣open␣output␣file";
exit(1);
}

string s;
while (getline(file, s))
{
outputFile.open("minTris2.txt", ios_base::app);
int startIndex = s.find("{") + 2;
int endIndex = s.rfind("}") - 1;

s = s.substr(startIndex, endIndex - startIndex);
vector<Point> points;

Point point;
while (getPoint(s, point))
{
points.push_back(point);
};
```

```
vector<Point2f> triangle;
minEnclosingTriangle(points, triangle);

outputFile << "c(" << triangle.at(0).x << ",␣" << triangle.at(0).
    ↪ y << "," << triangle.at(1).x << "," << triangle.at(1).y << "
    ↪ ," << triangle.at(2).x << "," << triangle.at(2).y << ")" <<
    ↪ endl;

outputFile.close();
}

return 0;
}
```

A large portion of this code is parsing through the input to obtain points. Recall that our polygons are presented in a way that *Macaulay2* can understand; however, this makes it more difficult to process in *C++*. The first function, *getPoint*, parses through its input to determine the x- and y-coordinates for a vertex of a Newton polygon. We then create a vector of points (which are the vertices for a Newton polygon) to use as an argument for *minEnclosingTriangle*. Finally, we write the output of *minEnclosingTriangle* into a format *R* can understand — a vector containing six points, the x-coordinate followed by y-coordinate, then likewise for the next two points.

We now use *R* to determine the area of the given triangles. We store them in a list so that we may compare them with the area of their respective polygons.

```
retArea <- function(x) {
abs((x[1]*(x[4]-x[6]) + x[3]*(x[6]-x[2]) + x[5]*(x[2]-x[4]))/2)
}



triArea <- function() {
#Read list of triangle coordinates.
#Read as R code.

tris <- scan("./minTris2.txt", what="character", sep="\n")

for(i in seq_along(tris)) {
tris[i] <- retArea(eval(parse(text=tris[i])))
}
tris
}
```

The function *retArea* takes as input three points in $\mathbb{R}^2$ and returns the area of the triangle they form. Note that this is essentially just the well-known Heron's formula.

The function *triArea* then calls *retArea* on every triangle in the list that our *C++* code produced. Note that since the *C++* algorithm is numeric in nature, the triangle areas are, at times, approximate.

The next step is to determine for which multiplicities m our polygons support a Laurent polynomial vanishing to m at $(1,1)$. Since this constitutes a one-line modification to the first piece of *Macaulay2* code we used, we include this code in the appendix.

Now, that we have a list of polygons supporting Laurent polynomials to multiplicity m at $(1,1)$, as well as a corresponding list of minimal enclosing triangles about these polygons, we want to check, for each polygon, at which multiplicities m (if any) the inequality $\sqrt{2A(\Delta)} \leq m$ holds. The following *R* code accomplishes this.

```
createList3 <- function() {

#We include USRADP3, a necessary file for this to work.
source("USRADP3.R")

#We read in the list of multiplicities returned by the second
    ↪ pass into M2.
mData <- scan("Multiplicities2.txt", what="character", sep="\n")

#We want this to be in a format R understands
for(i in seq_along(mData)) {
mData[i] <- gsub("{", "c(", mData[i], fixed=TRUE)
mData[i] <- gsub("}", ")", mData[i], fixed=TRUE)
}

#Get the area of min triangle for each polygon
triAreas <- triArea()

#Initialize vector of acceptable multiplicities at which the
    ↪ polygon supports polynomial vanishing...
goodMults <- NULL

#We want to compare multipicity to area, to check if it meets the
    ↪  criterion
for(i in seq_along(mData)) {
satisfies <- NULL
mults <- eval(parse(text=mData[i]))
for(j in seq_along(mults)) {
if(sqrt(2*as.numeric(triAreas[i])) <= mults[j]) {
satisfies <- mults[j]
break
}
}
if(length(satisfies)==0) {satisfies <-FALSE}
goodMults <- c(goodMults, satisfies)
}

return(goodMults)
}
```

This code first reads in the list of multiplicities returned by *Macaulay2*, then translates it into a format that R can easily understand. Next, it obtains the list of triangle areas. For each polygon, it then checks whether there is a multiplicity m with $\sqrt{2A(\Delta)} \leq m$ — if so, we retain only the largest multiplicity m for which this is true.

4.5 The Final Product: Putting Everything Together

Up to this point, we have checked which Newton polygons P support a Laurent polynomial vanishing to multiplicity m at $(1,1)$, where $\sqrt{2A(\Delta)} \leq m$, where Δ is the minimal enclosing triangle of P. The only task that remains is to write a final dataset for future use. While we have already processed most of the data we want to include, we would like to include the genus of each polygon. The code to do this can be found in the appendix — it contains no new or interesting commands or ideas.

We may now write the dataset. We do this in R, as per the following code.

```
createFinalTable <- function() {
source("USRADP3.R")
source("USRADP5.R")

#Read in list of polygons and their genuses
polygons <- scan("PolyList2.txt", what="character", sep="\n")
g <- scan("Genuses.txt", what="character", sep="\n")
triAreas <- triArea()

goodMults <- createList3()

#Create an indicator to tell when polys do not satisfy criterion
indicator <- goodMults == 0

g <- g[!indicator]
polygons <- polygons[!indicator]
triAreas <- triAreas[!indicator]
goodMults <- goodMults[!indicator]

equality <- sqrt(2*as.numeric(triAreas)) == goodMults

#create a matrix containing the polygons, their genus, and
    ↪ multiplicities at which
#criterion is satisfied, and write to a table.
finalPolys <- cbind(polygons, g, goodMults, equality)

write.table(finalPolys, file="FinalPolygons.txt", row.names=FALSE
    ↪ , col.names=FALSE)
}
```

This code begins by retrieving the list of polygons for which we found a minimal enclosing triangle. It then reads in the corresponding lists of genera and areas of minimal enclosing triangles. Next, an indicator is created, showing when, for a polygon P, $\sqrt{2A(\Delta)} > m$. We select all those polygons that do satisfy $\sqrt{2A(\Delta)} \leq m$, as well as the data regarding their genus, the largest multiplicity m for which the polygon supports a Laurent polynomial vanishing to multiplicity m at $(1,1)$, and the area of a minimal enclosing triangle area about the polygon.

There is one more piece of data that we include, which is whether or not the inequality is an equality; that is, we note whether $\sqrt{2A(\Delta)} = m$. This is the variable *equality* in the code above, which is the final column of our dataset. We reproduce the first few lines of our data, tidied up for legibility, in what follows.

Polygon	Genus	Max Multiplicity	Triangle Area	Equality
$\{\{0,0\},\{2,1\},\{1,2\}\}$	1	2	1.5	FALSE
$\{\{0,0\},\{2,0\},\{1,2\}\}$	1	2	2	TRUE
$\{\{0,1\},\{1,0\},\{2,0\},\{2,1\},\{1,2\},\{0,2\}\}$	1	3	4.5	TRUE
$\{\{0,0\},\{2,0\},\{2,1\},\{1,2\},\{0,2\}\}$	1	3	4.5	TRUE
$\{\{0,0\},\{3,0\},\{1,2\},\{0,2\}\}$	1	3	4.5	TRUE
$\{\{0,0\},\{2,0\},\{2,2\},\{0,2\}\}$	1	4	8	TRUE
$\{\{0,0\},\{3,0\},\{0,3\}\}$	1	3	4.5	TRUE
$\{\{0,-1\},\{1,-1\},\{3,0\},\{2,1\},\{1,1\}\}$	2	3	4.167	FALSE
$\{\{0,-1\},\{3,-1\},\{2,1\},\{1,1\}\}$	2	3	4.5	TRUE
$\{\{0,-1\},\{1,-1\},\{3,0\},\{1,1\},\{0,1\}\}$	2	3	4.5	TRUE
$\{\{0,-1\},\{2,-1\},\{3,0\},\{3,1\},\{1,1\},\{0,0\}\}$	2	4	8	TRUE
$\{\{0,-1\},\{3,-1\},\{3,0\},\{2,1\},\{0,1\}\}$	2	4	8	TRUE
$\{\{0,-1\},\{3,-1\},\{3,1\},\{0,1\}\}$	2	5	12	FALSE
$\{\{0,-1\},\{4,-1\},\{2,1\},\{0,1\}\}$	2	4	8	TRUE
$\{\{-1,0\},\{1,-1\},\{2,-1\},\{0,2\}\}$	3	3	4.375	FALSE
$\{\{-1,0\},\{3,-1\},\{-1,2\}\}$	3	3	4	FALSE
$\{\{-1,-1\},\{2,-1\},\{0,2\}\}$	3	3	4.5	TRUE
$\{\{0,-1\},\{2,-1\},\{4,0\},\{3,1\},\{1,1\}\}$	3	4	8	TRUE
$\{\{0,-1\},\{4,-1\},\{3,1\},\{1,1\}\}$	3	4	8	TRUE
$\{\{0,-1\},\{2,-1\},\{4,0\},\{2,1\},\{0,1\}\}$	3	4	8	TRUE
$\{\{-1,0\},\{0,-1\},\{2,-1\},\{2,0\},\{0,2\},\{-1,2\}\}$	3	4	8	TRUE
$\{\{0,-1\},\{3,-1\},\{4,0\},\{4,1\},\{1,1\},\{0,0\}\}$	3	5	12	FALSE
$\{\{-1,-1\},\{2,-1\},\{2,0\},\{0,2\},\{-1,2\}\}$	3	4	8	TRUE
$\{\{0,-1\},\{4,-1\},\{4,0\},\{3,1\},\{0,1\}\}$	3	5	12	FALSE
$\{\{-1,-1\},\{3,-1\},\{0,2\},\{-1,2\}\}$	3	4	8	TRUE
$\{\{0,-1\},\{4,-1\},\{4,1\},\{0,1\}\}$	3	6	16	FALSE
$\{\{0,-1\},\{5,-1\},\{3,1\},\{0,1\}\}$	3	5	12	FALSE
$\{\{-1,-1\},\{3,-1\},\{-1,3\}\}$	3	4	8	TRUE
$\{\{-1,-1\},\{3,1\},\{1,3\}\}$	4	4	6	FALSE
$\{\{-1,-1\},\{0,-1\},\{2,0\},\{2,1\},\{0,2\},\{-1,2\}\}$	4	4	8	TRUE
$\{\{0,-1\},\{3,-1\},\{5,0\},\{4,1\},\{1,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{5,-1\},\{4,1\},\{1,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{4,-1\},\{5,0\},\{4,1\},\{1,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{3,-1\},\{5,0\},\{3,1\},\{0,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{3,-1\},\{5,0\},\{4,1\},\{1,1\},\{0,0\}\}$	4	5	12.25	FALSE
$\{\{0,-1\},\{3,-1\},\{5,0\},\{5,1\},\{2,1\},\{0,0\}\}$	4	5	12.25	FALSE
$\{\{-1,-1\},\{1,-1\},\{2,0\},\{2,2\},\{0,2\},\{-1,1\}\}$	4	5	12.5	TRUE
$\{\{0,-1\},\{5,-1\},\{5,0\},\{4,1\},\{1,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{4,-1\},\{5,0\},\{3,1\},\{0,1\}\}$	4	5	12.25	FALSE
$\{\{0,-1\},\{4,-1\},\{5,0\},\{4,1\},\{1,1\},\{0,0\}\}$	4	5	12.25	FALSE
$\{\{-1,-1\},\{2,-1\},\{2,2\},\{0,2\},\{-1,1\}\}$	4	5	12.5	TRUE
$\{\{0,-1\},\{6,-1\},\{4,1\},\{1,1\}\}$	4	5	12	FALSE
$\{\{0,-1\},\{5,-1\},\{5,0\},\{3,1\},\{0,1\}\}$	4	5	12.25	FALSE
$\{\{0,-1\},\{4,-1\},\{5,0\},\{5,1\},\{1,1\},\{0,0\}\}$	4	6	16	FALSE

Chapter 5

An Algebro-Geometric Point of View

5.1 Introduction

Earlier, reference was made to such entities as negative curves and weighted projective planes. While no knowledge of algebraic geometry is required to understand what was done in this project, its motivation is rooted in algebraic geometry. Thus, we now define the objects with which our work is entwined, and give an idea how exactly they are connected, as well as the motivation for the project.

5.2 Background and Definitions

The main objects of study in algebraic geometry are varieties. Specifically, we are interested in studying what are known as projective varieties. Throughout, we use $\mathbb{K}$ to denote a field, and $\mathbb{K}^*$ its units.

Definition 14. *Define the equivalence relation $\sim$ by $(x_0, \ldots, x_n) \sim (y_0, \ldots, y_n)$ if and only if $(x_0, \ldots, x_n) = \lambda(y_0, \ldots, y_n)$ for some $\lambda \in \mathbb{K}^* \backslash \{0\}$.*

*Then **projective n-space**, denoted as $\mathbb{P}^n(\mathbb{K})$ or $\mathbb{P}^n$, is*

$$(\mathbb{K}^{n+1} \backslash \{0\}) / \sim .$$

One can think of projective n-space as the set of lines through the origin of an $(n+1)$-dimensional vector space.

Definition 15. *Let V be a subset of projective n-space, and $S = \{f_1, \ldots, f_k\}$ a set of homogeneous polynomials in $\mathbb{K}[x_0, x_1, \ldots, x_n]$ that generate a prime ideal. If V consists of exactly the common zeroes of the polynomials in S, then V is a **projective variety**.*

A **Mori Dream Space** is a particularly nice class of projective variety, with many convenient properties. While its definition is out of the scope of this project, we encourage the interested reader to learn more about them in [9]. There is a part of algebraic geometry called the *Mori Program* that seeks to understand in a precise way all varieties that are birational to a given variety V. Mori Dream Spaces are varieties for which all parts of the Mori Program work as one would wish. Our project is related to determining when, for a variety V which is the blowup of a weighted projective plane at a general point, V is a Mori Dream Space.

Definition 16. *Let $a = (a_0, a_1, a_2)$, $a_i \in \mathbb{N}$. $\mathbb{K}^*$ acts on $\mathbb{K}^3 \backslash \{0\}$ via*

$$\lambda(x_0, x_1, x_2) = (\lambda^{a_0} x_0, \lambda^{a_1} x_1, \lambda^{a_2} x_2)$$

*The a-**weighted projective plane** is*

$$\mathbb{P}(a_0, a_1, a_2) = (\mathbb{K}^3 \backslash \{0\})/\mathbb{K}^*$$

We will shorten *weighted projective plane* to WPP for the rest of this chapter.

Next, we want to understand what blowing up a WPP at a general point entails. A WPP has an open subset that looks like $(\mathbb{K}^*)^2$. In fact, we may identify points with all nonzero coordinates to $(\mathbb{K}^*)^2$. When we refer to a general point, we refer to a point in the WPP corresponding to a point in $(\mathbb{K}^*)^2$. However, any point in $(\mathbb{K}^*)^2$ which we choose to blowup will result in an isomorphic variety, so, without loss of generality, we consider the point $(1:1:1)$ — this is the point $(1,1)$ in $(\mathbb{K}^*)^2$.

Definition 17. *The **blowup** of a point in a plane replaces the point with its projectivized tangent space at that point. For a point $(a:b:c)$ with $c \neq 0$, write x, y for its coordinates on $(\mathbb{K}^*)^2$ with $c \neq 0$. Then its blowup is the variety*

$$V = \{(x, y), [z:w] \mid xz + yw = 0\} \subset \mathbb{K}^2 \times \mathbb{P}^1$$

Blowing up this point in $\mathbb{K}^2$, translating it to the torus, and shifting it will then give us a local picture of the blowup of $(1:1:1)$ on our WPP. In particular, the map from our blowup to $(\mathbb{K}^*)^2$ is an isomorphism everywhere except at $(1,1)$, which is replaced with the blowup.

For two curves on a surface, it is possible to define the notion of **intersection number**. If the curves are well-behaved, their intersection number is intuitive — it is simply a count of how many times they intersect. For example, the curves $y - x = 0$ and $x^2 + y^2 - 1 = 0$ have intersection number two.

However, defining this notion algebraically gives rise to some interesting possibilities. For instance, we can consider the intersection of a curve with itself, and we can get negative intersection number. In fact, a curve which has negative intersection number with itself is a **negative curve**. Roughly speaking, negative curves are irreducible algebraic curves on a surface, which cannot be deformed into a different irreducible algebraic curve on the same surface.

In [5], it is shown that the variety V, which is the blowup of a WPP at a general point — that is, $V = \mathrm{Bl}_{t_0}\mathbb{P}(a, b, c)$ — is a Mori Dream Space if and only if

1. V contains a negative curve C different from the exceptional curve E.

2. V contains a nonempty curve D disjoint from C.

5.3 The Connection

Now that we understand more about the entities with which our work is connected, we may explain the connection itself. How does finding a triangle about a polygon satisfying some area bound imply the existence of a certain class of negative curve on the blowup of a WPP?

A toric variety is determined by, and determines, a combinatorial object known as a fan. If this fan satisfies certain conditions, then we will obtain a WPP. Specifically, taking the normal fan of the triangles we found about our polygons gives three points p_0, p_1, p_2. If there are $\lambda_i \in \mathbb{K}^*$ such that

1. $\sum_{i=0}^{2} \lambda_i p_i = 0$, and

2. p_i generate the lattice $\mathbb{Z}^2$,

then the fan corresponds to $\mathbb{P}(\lambda_0, \lambda_1, \lambda_2)$. The triangles we find about the polygons lead us to the λ_i, which give us the WPPs. We encourage the interested reader to learn more on these topics in [4].

The bivariate Laurent polynomials that our Newton polygons support can be thought of as functions on $(\mathbb{K}^*)^2$. Thus, we may consider a Laurent polynomial f as a curve by looking at its variety $V(f)$. We take the closure of $V(f)$ in our WPP blown up at a general point, and take its strict transform — roughly speaking, this is removing the point $(1, 1)$ in the closure of $V(f)$, taking the preimage under our blowup map, and taking the closure in the blown up WPP. This ensures that the curve we get is not the exceptional curve E (see for instance [8]). In fact, this curve will be a negative curve if $A(\Delta) \leq m^2/2$.

Thus, what we have done is to check the first condition of the criteria established by Cutcosky in [5] for certain blowups of WPPs.

Chapter 6

Conclusion

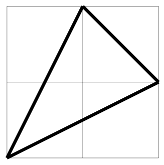
We have translated the process described in [6] into a question of computation. This, in conjunction with the dataset from [2], leads us to a seamless process of determining blowups of weighted projective planes that contain a certain class of negative curves. Our result is a dataset containing Newton polygons, their genus, the maximum multiplicity for which they satisfy the area criterion, as well as the area of their minimal enclosing triangle. Currently, we have processed polygons for genus up to 18. Our data will be made publicly available in a Github repository, at github.com/NewtonPolyProject/NewtonPolyProject.

Further work on this subject could look at the dataset we produced, with the goal of determining patterns or structure within the entries. For instance, is there a recurring family of vertices which seem to always satisfy $\left\lceil \sqrt{2A(\Delta)} \right\rceil \leq m$ for some m and some polygon P, other than those found in [6]?

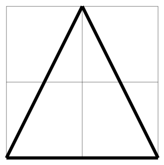
We now give the genus one, genus two, and genus three polygons, as well as their minimal enclosing triangles and information about the maximum multiplicity and minimal enclosing triangle area.

Genus One

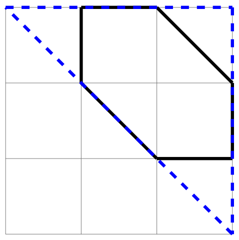
Maximum Multiplicity: 2
Triangle Area: 1.5



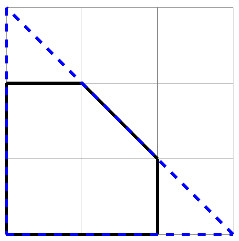
Maximum Multiplicity: 2
Triangle Area: 2



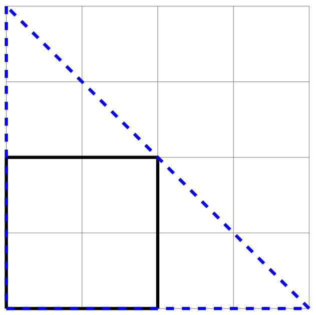
Maximum Multiplicity: 3
Triangle Area: 4.5



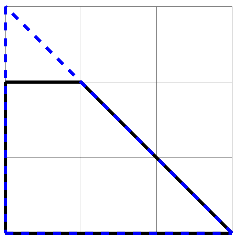
Maximum Multiplicity: 3
Triangle Area: 4.5



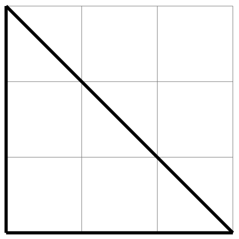
Maximum Multiplicity: 4
Triangle Area: 8



Maximum Multiplicity: 3
Triangle Area: 4.5

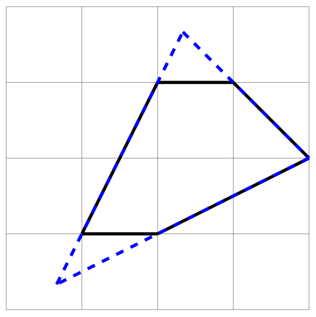


Maximum Multiplicity: 3
Triangle Area: 4.5

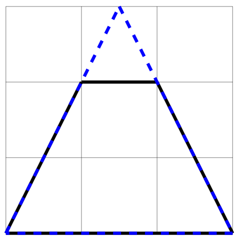


Genus Two

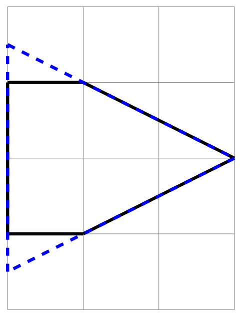
Maximum Multiplicity: 3
Triangle Area ≈ 4.166677500005



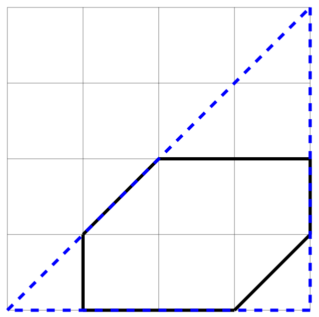
Maximum Multiplicity: 3
Triangle Area: 4.5



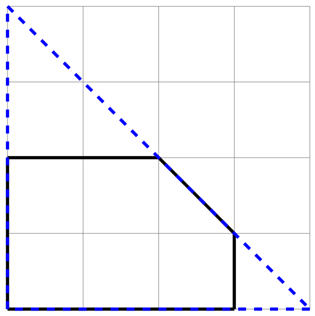
Maximum Multiplicity: 3
Triangle Area: 4.5



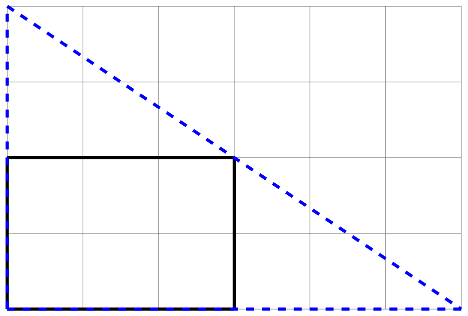
Maximum Multiplicity: 4
Triangle Area: 8



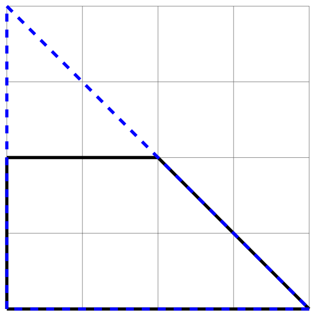
Maximum Multiplicity: 4
Triangle Area: 8



Maximum Multiplicity: 5
Triangle Area: 12



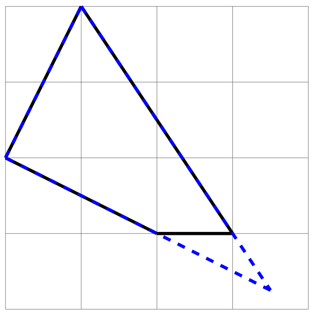
Maximum Multiplicity: 4
Triangle Area: 8



Genus Three

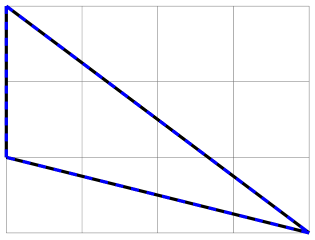
Maximum Multiplicity: 3

Triangle Area: 4.375



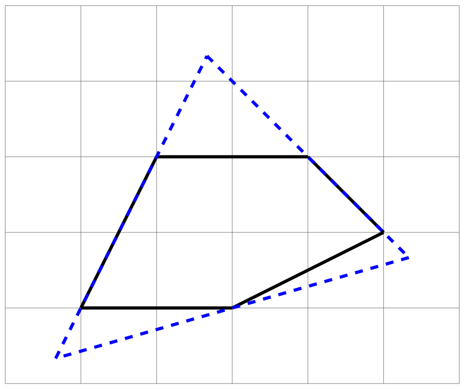
Maximum Multiplicity: 3

Triangle Area: 4



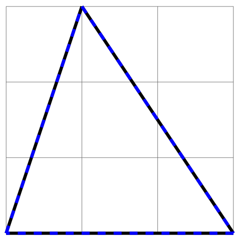
Maximum Multiplicity: 4

Triangle Area: 8



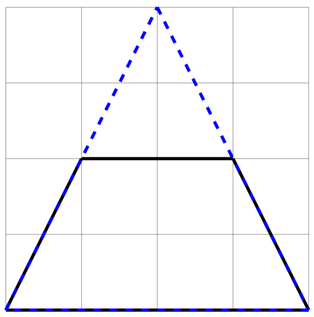
Maximum Multiplicity: 3

Triangle Area: 4.5



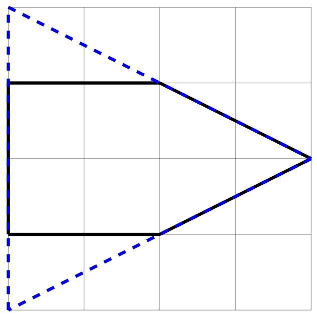
Maximum Multiplicity: 4

Triangle Area: 8

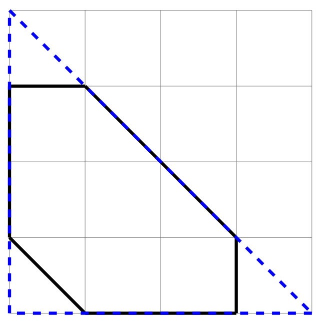


Maximum Multiplicity: 4

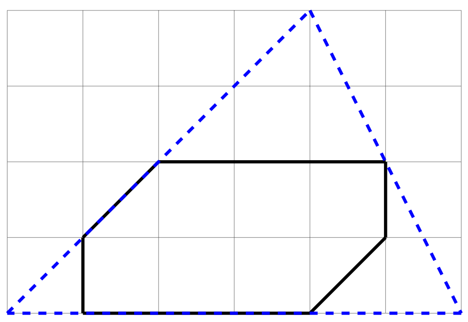
Triangle Area: 8



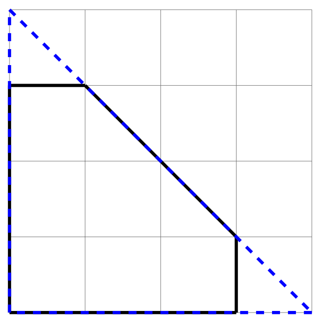
Maximum Multiplicity: 4
Triangle Area: 8



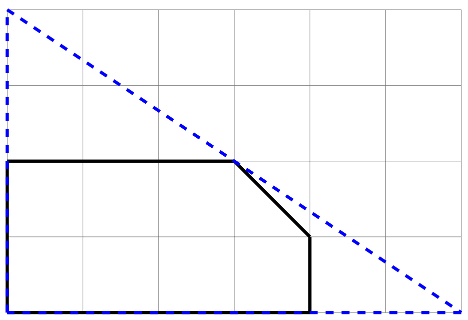
Maximum Multiplicity: 5
Triangle Area: 12



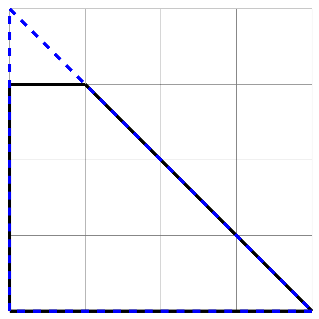
Maximum Multiplicity: 4
Triangle Area: 8



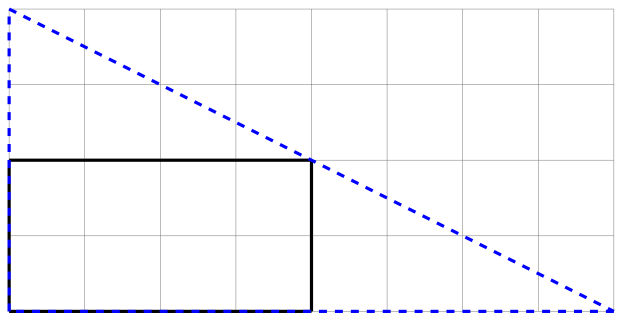
Maximum Multiplicity: 5
Triangle Area: 12



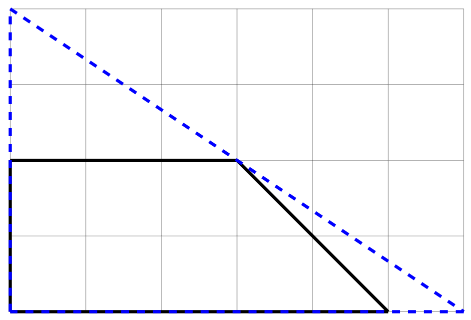
Maximum Multiplicity: 4
Triangle Area: 8



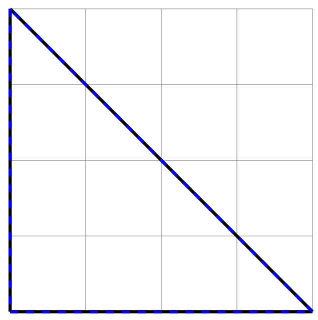
Maximum Multiplicity: 6
Triangle Area: 16



Maximum Multiplicity: 5
Triangle Area: 12



Maximum Multiplicity: 4
Triangle Area: 8



Bibliography

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] W. Castryck. Moving out the edges of a lattice polygon. *Discrete and Computational Geometry*, 47(3):496–518, 2012.

[3] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Undergraduate Texts in Mathematics. Springer, 233 Spring Street, New York, 3 edition, 2007.

[4] D. Cox, J. Little, and H. Schenk. *Toric Varieties.* American Mathematical Society, 201 Charles St, Providence, 2011.

[5] S. Cutkosky. Symbolic algebras of monomial primes. *J. Reine Angew. Math.*, 416(1):71–89, 1991.

[6] J. González-Anaya, J. L. González, and K. Karu. On a family of negative curves. *ArXiv e-prints*, arXiv:1712.04635, December 2017.

[7] D. Grayson and M. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[8] R. Hartshorne. *Algebraic Geometry.* Graduate Texts in Mathematics. Springer, 233 Spring Street, New York, 1 edition, 1977.

[9] Y. Hu and S. Keel. Mori Dream Spaces and GIT. *Michigan Math. J.*, 48(1):331–348, 2000.

[10] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin. An optimal algorithm for finding minimal enclosing triangles. *Journal of Algorithms*, 7(2):258–269, June 1986.

[11] O. Pârvu and D. Gilbert. Implementation of linear minimum area enclosing triangle algorithm. *Computational and Applied Mathematics*, 35(2):423–438, July 2016.

Appendix A

Code

Here, we include pieces of code which strongly resembled previous ones.

Note that the beginning of the following code chunk is similar to our data preprocessing step, with the difference being we read in our indicator list obtained from the previous code as *filterIndex*. By doing so we can easily select only those polygons for which we want to find a minimal enclosing triangle.

```
createList2 <- function() {
#Read in the list which contains true/false
#values for polygons to minimize.

filterIndex <- scan("./minimize2.txt", what="
   ↪ character")
t <- scan("./Polygons1to30.txt", what="character"
   ↪ , fill=T, sep="\n")

#Every line corresponding to a valid polygon
# begins with "{ {", we search for these.
polyt <- grepl("{␣{", t, fixed=TRUE)

#Create new list containing only Newton
#Polygons
t2 <- t[polyt]

#We also need to eliminate commas from the end
#of any line, lest Macaulay2 believes it to be
#a sequence.
for(i in seq_along(t2)) {
if(grepl("}␣},", t2[i])==TRUE) {
t2[i] <- substr(t2[i], 1, nchar(t2[i])-1)
}
}
```

```
                #Set remaining values of filterIndex false by
                #default. Convert filterIndex to logical variable
                  ↪ .
                filterIndex[(length(filterIndex)+1):length(t2)]
                  ↪ <- "false"
                filterIndex <- as.logical(filterIndex)

                #Select those polygons from t2 for minimization.
                t3 <- t2[filterIndex]

                write(t3, file="toMinimize.txt")
                }
```

The next code chunk is similar to the first piece of *Macaulay2* code presented. Note that the only change is the line *"for m from ceiling..."* In particular, this code checks for which multiplicities a polygon P will support a Laurent polynomial vanishing to multiplicity m at $(1, 1)$, and returns each of these m.

```
checkPoly=L->(
c:=min(apply(L,a->a_0));
d:=min(apply(L,a->a_1));
ms:={};
LL:=apply(L,i->{i_0-c,i_1-d});
P:=convexHull transpose matrix LL; -- create a polygon from list
  ↪ of vertices
lp:=apply(latticePoints P,i->flatten entries i); -- list of all
  ↪ lattice points in P
R:=QQ[apply(lp,i->a_i)]; -- ring with variables indexed by
  ↪ lattice points of P
S:=R[x_0,x_1,Join=>false]; -- ring for our polynomial f
f:=sum apply(lp,i->(a_i)*(x_0^(i_0))*(x_1^(i_1))); -- create a
  ↪ general polynomial with P as newton polygon
C:=coefficients sub(f,{x_0=>x_0+1,x_1=>x_1+1}); -- do change of
  ↪ variables and make coefficient matrix
for m from ceiling sqrt (2*(volume P)) to first degree f do ( --
  ↪ we need the vanishing multiplicity to be at least m
I:=sub(ideal (flatten entries C_1)_(positions(flatten entries C_0
  ↪ ,i->(degree i)_0<m)),R); -- conditions imposed on the
  ↪ coefficients by the multiplicity requirement
if not any(LL,i->R_(position(lp,j->j==i))%I==0) then (ms=append(
  ↪ ms,m)); -- checks if any of the coefficients for a vertex
  ↪ vanishes
);
return ms;
)



polyList2=L->(
g := openIn("PolyList2.txt");
while 1 == 1 do (
f := openOutAppend("Multiplicities2.txt");
f << checkPoly(value(read(g,106))) << endl;
close(f);
);
)
```

The final code chunk computes the genus of a Newton polygon.

```
polyGenus=L->(
        length interiorLatticePoints convexHull transpose matrix
           ↪ L
)

genusList=L->(
        g := openIn("PolyList2.txt");
        while 1 == 1 do (
                f := openOutAppend("Genuses.txt");
                f << polyGenus(value(read(g,106))) << endl;
                close(f);
        );
)
```

The function *polyGenus* takes as input a list of vertices corresponding to a Newton polygon. It then creates this Newton polygon and counts the number of interior points. *genusList* calls *polyGenus* for every polygon we selected previously.