The University of New South Wales
School of Mathematics and Statistics

Planar Universal Geometry

Timothy Blair Leslie
<tim.leslie@gmail.com>

**Masters of Science and Technology
(Mathematics)**

October 2009

Supervisor: A/Prof. Norman J. Wildberger

Abstract

We consider affine universal geometry in the case of a two dimensional vector space with an arbitrary quadratic form over a general field. A series of theorems for lines and conics are presented, which extend previous results from Euclidean rational trigonometry and chromogeometry. A numerical software library, `pygeom`, is also presented. This library uses the results derived here to allow calculations to be performed.

Contents

List of Figures

Chapter 1

Introduction

1.1 Rational Trigonometry

Classical trigonometry, as generally taught to students in high school, defines distances in terms of square roots and angles in terms of the transcendental trigonometric functions. While these ideas have found many applications, both theoretical and practical, they have a number of drawbacks as well. In particular, the need to calculate square roots and trigonometric functions makes it impossible to get exact numerical answers in all but the most trivial of cases.

Rational trigonometry [1] addresses these issues by replacing the notion of distance with that of *quadrance* and the idea of angles with that of *spread*. Quadrance is just the square of the distance while the spread between two lines with an angle θ is equal to $\sin^2\theta$. By working with these new quantities, it is possible to develop many geometrical theorems in purely algebraic terms. This means that all calculations can be made *exactly*, overcoming this particular drawback of classical geometry.

Although the spread is equivalent to $\sin^2\theta$, it can be defined algebraically without reference to the trigonometric functions. This allows trigonometry to be developed without having to first develop a theory of circular functions (i.e. sin, cos, etc). It can be argued that this makes teaching and learning trigonometry simpler, as the theory of triangles is kept separate from the theory of circles [2].

It has been found that rational trigonometry provides a powerful set of theorems[1]. While some of these theorems, such as the triple quad formula [1] have no direct classical analogue, many of the theorems of classical geometry, such as Heron's formula and Pythagoras' theorem, can be derived from rational trigonometry [3].

1.2 Universal Geometry

The ideas of rational trigonometry can be extended from the two dimensional, Euclidean form to a completely general form which works over an n-dimensional vector space with a metric defined by a symmetric bilinear form [4]. This *universal geometry* allows results to be obtained in a general setting. By considering specific bilinear forms one can recover results for both Euclidean and hyperbolic geometry.

The spread between two lines and the quadrance between two points in universal geometry is defined in terms of the bilinear form. As such all constructions can be done in terms of this single object. From this simple foundation many interesting ideas can be recovered, including many results of Euclidean geometry. While some work has been done to develop results in this area [5], there remains a broad scope for further investigations.

1.3 Chromogeometry

One particular area of universal geometry which has been developed is *chromogeometry* [6]. In chromogeometry we consider the case of a two dimensional vector space over a general field. We then take the three different quadratic forms $G_b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $G_r = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ and $G_g = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and get the three geometries known as the *blue*, *red* and *green* geometries respectively. The blue geometry corresponds to the usual Euclidean geometry while the red and the green are both hyperbolic geometries.

Chromogeometry investigates the three-fold symmetries which exist between geometrical objects in each of these geometries. For example, given a parabola, the focus and directrix will be different in each of the different geometries. An interesting result arises in that the directrices from the red and green

[1]For a series of video tutorials on Rational Trigonometry see the "WildTrig" series at http://www.youtube.com/view_play_list?p=3C58498718451C47&search_query=wildtrig

geometries intersect at the blue focus, the red and blue directrices intersect at the green focus and the blue and green directrices intersect at the red focus [7].

1.4 Planar Universal Geometry

Planar Universal Geometry studies the specific case of universal geometry in a two dimensional plane. This contains the geometries of chromogeometry as a special case, but does not extend to the arbitrary dimensionality of universal geometry. Planar universal geometry has not been systematically studied in detail previously, and so this thesis aims to address this.

Theories developed in this area may serve a number of purposes. Firstly, they will be useful in their own right in making numerical calculations and further developing the field. Secondly they may provide insight into the results which can be expected in the case of n-dimensional universal geometry. Finally, by developing theories in the case of an arbitrary quadratic form, one gets Euclidean geometry and chromogeometry as a special case. This removes the need to prove individual results from first principles in these cases.

1.5 Overview

This thesis comprises of two main sections. In the first, a set of theorems in planar universal geometry are proven. In the second, a software library for doing numerical calculations in planar universal geometry is presented. This software package implements many of the theorems proved in the first section, providing a practical tool to assist in further investigations in the field.

In Chapter 2 we introduce the idea of a geometry with a metric defined by a quadratic form, and develop some results for points and lines in such a geometry. Chapter 3 builds on this, developing a set of theories relating to conics in planar universal geometry. In Chapter 4 a new software library, `pygeom`, is presented. Chapter 5 discusses the test framework which comes with `pygeom` and ensures that it produces correct results in a robust manner.

The source code of the `pygeom` library is attached as a separate appendix. A soft copy of the code is available from the author upon request.

Chapter 2

Planar Universal Geometry

We begin by developing some results about points and lines in the plane when taken with an arbitrary quadratic form as the metric. Some of the results presented below are well known results from Euclidean geometry while others are completely novel. Those non-original results are included for completeness, so that the reader may have a comprehensive set of results presented in a consistent notation. Many of the definitions make reference to a general field $\mathbb{F}$. We will take as given that this field is not of characteristic two, as otherwise many of the definitions would break down.

2.1 Notation and Definitions

Definition 1. Given a field $\mathbb{F}$, a *point* is any element $X \in \mathbb{F} \times \mathbb{F}$ and we write $X = [x, y]$ where $x, y \in \mathbb{F}$.

An alternative representation of a point is as a two element column vector. The vector representation allows us to use standard linear algebra notation when doing calculations.

Definition 2. The vector representation of a point X is

$$\vec{X} = \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.1}$$

Definition 3. Given a field $\mathbb{F}$, a *line* is the locus of points $[x, y] \in \mathbb{F} \times \mathbb{F}$ which satisfy the equation

$$ax + by + c = 0, \tag{2.2}$$

where $a, b, c \in \mathbb{F}$ and a, b are not both zero. The standard representation of this line is $\langle a : b : c \rangle$, which emphasises the fact that what is important is the ratio between the values of a, b and c. The lines $\langle a : b : c \rangle$ and $\langle \lambda a : \lambda b : \lambda c \rangle$, where $\lambda \in \mathbb{F}$ is non-zero, are equivalent, as they represent the same locus of points.

We can also represent a line as a two element column vector. This representation does not completely specify the line and so more care must be taken with its use.

Definition 4. The vector representation of a line l is

$$\vec{l} = \begin{pmatrix} -b \\ a \end{pmatrix}. \tag{2.3}$$

We also note here that the forms $\langle a : b : c \rangle$ and $\langle \lambda a : \lambda b : \lambda c \rangle$ represent the same line, yet have different vector representations. As such when using the vector representation of a line we must be careful not to use the scaling properties of the standard representation, as this will lead to inconsistencies.

To perform any kind of geometrical measurement, we require a way to define the quantities being measured. By changing the way quantities are measured, we can change the properties of the geometry itself. In planar universal geometry measurement is done in terms of a quadratic form, known as the metric.

Definition 5. Given a field $\mathbb{F}$, a *metric* G is a symmetric, non-singular two-by-two matrix over the field, i.e.

$$\begin{aligned} G &= \begin{pmatrix} a & b \\ b & c \end{pmatrix} && (2.4) \\ \Delta_G &= ac - b^2 \\ &\neq 0 && (2.5) \end{aligned}$$

where $a, b, c \in \mathbb{F}$.

To make reading the following proofs simpler, a standard naming convention has been adopted for points, lines and metrics. A points will be denoted as X_n, and it is assumed that its components are $[x_n, y_n]$. A line will be denoted as l_n, and it is assumed that its general form is $\langle a_n : b_n : c_n \rangle$. The metric G will always be assumed to be $\begin{pmatrix} a & b \\ b & c \end{pmatrix}$. Unless otherwise specified it can be assumed that this convention is being followed.

We will often associate a geometry with its metric, and thus the phrases "a geometry with a metric G" and "a geometry G" will be used interchangeably. Planar universal geometry allows many of the ideas of Euclidean geometry to be generalised. The dot product is one such operation.

Definition 6. The *metric dot product* in a geometry with metric G is defined as

$$\vec{v} \cdot_G \vec{u} \;=\; \vec{v}^T G \vec{u} \tag{2.6}$$

where $\vec{v}$ and $\vec{u}$ are vectors which may represent either lines or points.

The definition of a metric dot product leads naturally to the idea of a metric norm.

Definition 7. The *metric norm* of a vector $\vec{q}$ in a geometry with metric G is defined as

$$\|\vec{q}\|_G \;=\; \vec{q}^T G \vec{q}. \tag{2.7}$$

It will often be clear from the context that the metric dot product or metric norm is being used, in which case the $_G$ subscript will be dropped to ease readability.

Definition 8. A line l is a *null line* in the geometry G if the norm of the vector representation of the line is zero, i.e. $\|\vec{l}\|_G = 0$.

Definition 9. A point X is a *null point* in the geometry G if the norm of the vector representation of the point is zero, i.e. $\|\vec{X}\|_G = 0$.

In standard Euclidean geometry, the separation of any two points is given in terms of distance, which is calculated using a metric function such as $d(X_0, X_1) = \sqrt{\|\vec{X_1} - \vec{X_0}\|}$. Since we wish to use only field operations, we replace the concept of distance with that of quadrance.

Definition 10. The *quadrance* between two points X_0 and X_1 in a geometry with metric G is defined as

$$Q(X_0, X_1)_G \;=\; \left\|\vec{X_0} - \vec{X_1}\right\|_G. \tag{2.8}$$

As is the case with distance, the standard measure of separation of lines, the angle between them, cannot be defined in terms of field operations. We replace the notion of angle with that of spread.

Definition 11. The *spread* between two non-null lines l_1 and l_2 in a geometry with metric G is defined as

$$s = 1 - \frac{\left(\vec{l_1} \cdot_G \vec{l_2}\right)^2}{\|\vec{l_1}\|_G \|\vec{l_2}\|_G}. \tag{2.9}$$

If either l_1 or l_2 are null lines then the spread between these lines is not defined. The definition above is essentially identical to that for spread in universal geometry. We can use the fact that we are working in two dimensions to simplify this further.

Theorem 1. The spread between the non-null lines $l_1 = \langle a_1 : b_1 : c_1 \rangle$ and $l_2 = \langle a_2 : b_2 : c_2 \rangle$ in a geometry G is

$$s = \Delta_G \frac{(a_1b_2 - a_2b_1)^2}{\|\vec{l_1}\|\|\vec{l_2}\|}. \tag{2.10}$$

Proof. We first evaluate the denominator in the spread formula to get

$$\begin{aligned}
\|\vec{l_1}\|\|\vec{l_2}\| &= (ab_1^2 - 2ba_1b_1 + ca_1^2)(ab_2^2 - 2ba_2b_2 + ca_2^2) \\
&= a^2b_1^2b_2^2 + 4b^2a_1a_2b_1b_2 + c^2a_1^2a_2^2 - 2abb_1b_2(b_1a_2 + a_1b_2) + \\
&\quad ac(a_2^2b_1^2 + a_1^2b_2^2) - 2bca_1a_2(a_2b_1 + a_1b_2).
\end{aligned}$$

Now, the numerator is

$$\begin{aligned}
\left(\vec{l_1} \cdot \vec{l_2}\right)^2 &= (b_1(ab_2 - ba_2) - a_1(bb_2 - ca_2))^2 \\
&= (ab_1b_2 - b(a_1b_2 + a_2b_1) + ca_1a_2)^2 \\
&= a^2b_1^2b_2^2 + b^2(a_1b_2 + a_2b_1)^2 + c^2a_1^2a_2^2 - 2abb_1b_2(a_1b_2 + a_2b_1) + \\
&\quad 2aca_1a_2b_1b_2 - 2bca_1a_2(a_1b_2 + a_2b_1).
\end{aligned}$$

Using the result found for the denominator we can write this as

$$\begin{aligned}
\left(\vec{l_1} \cdot \vec{l_2}\right)^2 &= \|\vec{l_1}\|\|\vec{l_2}\| + b^2(a_1b_2 + a_2b_1)^2 + 2aca_1a_2b_1b_2 - 4b^2a_1a_2b_1b_2 - \\
&\quad ac(a_1^2b_2^2 + a_2^2b_1^2) \\
&= \|\vec{l_1}\|\|\vec{l_2}\| + (b^2 - ac)(a_1b_2 - a_2b_1)^2 \\
&= \|\vec{l_1}\|\|\vec{l_2}\| - \Delta_G(a_1b_2 - a_2b_1)^2.
\end{aligned}$$

The spread formula now becomes

$$\begin{aligned}
s &= 1 - \frac{\|\vec{l_1}\|\|\vec{l_2}\| - \Delta_G(a_1b_2 - a_2b_1)^2}{\|\vec{l_1}\|\|\vec{l_2}\|} \\
&= \Delta_G \frac{(a_1b_2 - a_2b_1)^2}{\|\vec{l_1}\|\|\vec{l_2}\|}.
\end{aligned}$$

□

Definition 12 (Archimedes function)**.** The function $A : \mathbb{F}\times\mathbb{F}\times\mathbb{F} \to \mathbb{F}$ defined as

$$a, b, c \quad \mapsto \quad (a+b+c)^2 - 2(a^2+b^2+c^2) \tag{2.11}$$

is called *Archimedes function* [1].

2.2 Properties of Lines

Many of the properties of lines in rational trigonometry extend naturally to planar universal geometry. In the following definitions we do not use the metric of the geometry explicitly and so equivalent results can be found in the study of rational trigonometry [1].

Definition 13. Two lines are *perpendicular* when the spread between them is equal to one.

Corollary 2. If two lines are perpendicular then their metric dot product is zero.

Proof. This follows directly from the definition of the spread between two lines. □

Definition 14. Two lines l_1 and l_2 are *parallel* if $a_1b_2 - a_2b_1 = 0$.

Corollary 3. Two non-null lines l_1 and l_2 are parallel if and only if the spread between them is zero.

Proof. This follows directly from Theorem 1. □

Definition 15. A *point of intersection* of two lines is a point which lies on both lines.

Theorem 4. If two lines l_1 and l_2 are not parallel then they intersect at the point $X = \left[\frac{b_1c_2-b_2c_1}{a_1b_2-a_2b_1}, \frac{a_2c_1-a_1c_2}{a_1b_2-a_2b_1}\right]$.

Proof. We need to solve the following two equations for x and y:

$$\begin{aligned} a_1x + b_1y + c_1 &= 0 \\ a_2x + b_2y + c_2 &= 0. \end{aligned}$$

Writing these in matrix form we get

$$\begin{aligned}
\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} &= -\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \\
\begin{pmatrix} x \\ y \end{pmatrix} &= -\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix}^{-1} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \\
&= \frac{-1}{a_1 b_2 - a_2 b_1} \begin{pmatrix} b_2 & -b_1 \\ -a_2 & a_1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}
\end{aligned}$$

□

Definition 16. Given a point X_0 and a line l_1, the *residue* of the point with respect to the line is

$$l_1(X_0) = a_1 x_0 + b_1 y_0 + c_1. \tag{2.12}$$

Corollary 5. If the point X lies on the line l then $l(X) = 0$.

Proof. This is an immediate consequence of the definition of a line as the locus of points satisfying the equation $ax + by + c = 0$. □

2.3 Properties of Points

Given two distinct points, the line which passes through them is defined independently of the geometry. As such the first theorem regarding points is a well known result of Euclidean geometry.

Theorem 6. The line which passes through the points X_0 and X_1 is $\langle a_1 : b_1 : c_1 \rangle$ where

$$\begin{aligned}
a_1 &= y_1 - y_0 && (2.13) \\
b_1 &= x_0 - x_1 && (2.14) \\
c_1 &= x_1 y_0 - x_0 y_1. && (2.15)
\end{aligned}$$

Proof. If we assume the line has the form given above and then take the residues of X_0 and X_1 we get

$$\begin{aligned}
l_1(X_0) &= (y_1 - y_0)x_0 + (x_0 - x_1)y_0 + (x_1 y_0 - x_0 y_1) \\
&= 0 \\
l_1(X_1) &= (y_1 - y_0)x_1 + (x_0 - x_1)y_1 + (x_1 y_0 - x_0 y_1) \\
&= 0.
\end{aligned}$$

Since the residue of both points is zero they both lie on the line, as required. □

The quadrance between two points is defined in terms of the metric of the geometry. As such we expect that the set of points which are equiquadrance from two given points to depend on the metric. As is the case with Euclidean geometry, this set of points turns out to always be a line.

Definition 17. The *equiquadrance line* in a geometry G between two points X_0 and X_1 is the locus of points such that $Q(X, X_0) = Q(X, X_1)$.

Theorem 7. Given two points X_0, X_1 in a geometry G, the equation of the equiquadrance line is $\langle a_1 : b_1 : c_1 \rangle$ where

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} = 2G(\vec{X_0} - \vec{X_1}) \quad (2.16)$$

$$c_1 = \vec{X_1}^2 - \vec{X_0}^2. \quad (2.17)$$

Proof. From the definition we have

$$\begin{aligned} Q(X, X_0) &= Q(X, X_1) \\ \vec{X_0}^2 - 2\vec{X_0} \cdot \vec{X} + \vec{X}^2 &= \vec{X_1}^2 - 2\vec{X_1} \cdot \vec{X} + \vec{X}^2 \\ 2(\vec{X_0} - \vec{X_1}) \cdot \vec{X} + \vec{X_1}^2 - \vec{X_0}^2 &= 0. \end{aligned}$$

□

We can now obtain a result which is inspired by Euclidean geometry. In Euclidean geometry, the points on the perpendicular bisector of two points are equidistant from the two points. In planar universal geometry we have the following generalised version of this theorem.

Theorem 8. Given two points X_0 and X_1, the equiquadrance line is perpendicular to the line between the two points.

Proof. If l_1 is the line through X_0 and X_1, and l_2 is their perpendicular bisector then the metric dot product of the two lines is

$$\begin{aligned} \vec{l_1} \cdot \vec{l_2} &= \begin{pmatrix} -b_2 \\ a_2 \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} -b_1 \\ a_1 \end{pmatrix} \\ &= 2 \begin{pmatrix} -(b(x_0 - x_1) + c(y_0 - y_1)) \\ a(x_0 - x_1) + b(y_0 - y_1) \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix} \\ &= 2 \begin{pmatrix} -(b(x_0 - x_1) + c(y_0 - y_1)) \\ a(x_0 - x_1) + b(y_0 - y_1) \end{pmatrix} \begin{pmatrix} a(x_1 - x_0) + b(y_1 - y_0) \\ b(x_1 - x_0) + c(y_1 - y_0) \end{pmatrix} \\ &= 0. \end{aligned}$$

Since their dot product is zero, the lines are perpendicular. □

2.4 Altitudes

Another idea we can borrow from Euclidean geometry is that of an altitude.

Definition 18. In a geometry with metric G, given a point X_0 and a line l_1, an *altitude* is a line which passes through X_0 and is perpendicular to l_1 with respect to the metric.

While the definition of an altitude in planar universal geometry is equivalent to that in Euclidean geometry, the equation of the altitude line and its foot depend on the metric.

Theorem 9. Given a point X_0 and a line l_1 in a geometry with metric G, the altitude l_a has the form $\langle ba_1 - ab_1 : ca_1 - bb_1 : (ab_1 - ba_1)x_0 + (bb_1 - ca_1)y_0\rangle$.

Proof. We require l_1 and l_a to be perpendicular, which means we must have $\vec{l_1} \cdot_G \vec{l_a} = 0$. This leads to the equation

$$-(ba_1 - ab_1)b_a + (ca_1 - bb_1)a_a = 0.$$

A solution to this equation is $\vec{l_a} = (bb_1 - ca_1, ba_1 - ab_1)^T$, giving $l_a = \langle ba_1 - ab_1 : ca_1 - bb_1 : c_a\rangle$. We require the altitude l_a to pass through X_0 and so

$$\begin{aligned} l_a(X_0) &= 0 \\ (ba_1 - ab_1)x_0 + (ca_1 - bb_1)y_0 + c_1 &= 0 \\ c_1 &= (ab_1 - ba_1)x_0 + (bb_1 - ca_1)y_0. \end{aligned}$$

□

Definition 19. In a geometry with metric G, given a line, a point and the altitude from the point to the line, the *altitude foot* is defined as the point where the altitude intersects the line.

Theorem 10. Given a point X_0 and a line l_1 in a geometry with metric G, the foot of the altitude is $F = [x_F, y_F]$ where

$$F_x = -\frac{b_1x_0(ba_1 - ab_1) + (ca_1 - bb_1)(c_1 + b_1y_0)}{\|\vec{l_1}\|} \tag{2.18}$$

$$F_y = \frac{a_1y_0(ca_1 - bb_1) + (ba_1 - ab_1)(c_1 + a_1x_0)}{\|\vec{l_1}\|}. \tag{2.19}$$

Proof. For a given l_1 and X_0, we wish to find the foot of the altitude, $F = [x_F, y_F]$, which is the point where l_1 intersects the altitude l_a. From Theorem 4, which gives the intersection of two lines, we find that

$$\begin{aligned}
F_x &= \frac{b_1 c_a - b_a c_1}{a_1 b_a - a_a b_1} \\
&= \frac{-b_1((ba_1 - ab_1)x_0 + (ca_1 - bb_1)y_0) - (ca_1 - bb_1)c_1}{a_1(ca_1 - bb_1) - (ba_1 - ab_1)b_1} \\
&= \frac{-b_1 x_0(ba_1 - ab_1) - (ca_1 - bb_1)(c_1 + b_1 y_0)}{\|\vec{l_1}\|} \\
F_y &= \frac{a_a c_1 - a_1 c_a}{a_1 b_a - a_a b_1} \\
&= \frac{c_1(ba_1 - ab_1) - a_1((ab_1 - ba_1)x_0 + (bb_1 - ca_1)y_0)}{\|\vec{l_1}\|} \\
&= \frac{(ba_1 - ab_1)(c_1 + a_1 x_0) + (ca_1 - bb_1)a_1 y_0}{\|\vec{l_1}\|}.
\end{aligned}$$

□

While the equation for the foot of the altitude is reasonably complex, the quadrance between the foot and the original point is relatively simple.

Definition 20. The *point-line quadrance* between a point and a line is defined as the quadrance between the point and the foot of the altitude formed with the line.

Theorem 11. Given a point X_0 and a line l_1 in a geometry with metric G, the point-line quadrance is

$$Q(X_0, l_1) = \frac{l_1(X_0)^2}{\|\vec{l_1}\|}\Delta_G. \tag{2.20}$$

Proof. If $X_0 = [x_0, y_0]$ and the foot of the altitude is $F = [x_F, y_F]$ then from the definition we have

$$\begin{aligned}
Q(X_0, l_1) &= Q(X_0, F) \\
&= a(x_F - x_0)^2 + 2b(x_F - x_0)(y_F - y_0) + c(y_F - y_0)^2.
\end{aligned} \tag{2.21}$$

From Theorem 10 we have

$$\begin{aligned}
x_F - x_0 &= \frac{-b_1x_0(ba_1 - ab_1) - (ca_1 - bb_1)(c_1 + b_1y_0) - (ab_1^2 - 2ba_1b_1 + ca_1^2)x_0}{ab_1^2 - 2ba_1b_1 + ca_1^2} \\
&= \frac{ba_1b_1x_0 - (ca_1 - bb_1)(c_1 + b_1y_0) - ca_1^2x_0}{\|\vec{l_1}\|} \\
&= \frac{(bb_1 - ca_1)(a_1x_0 + b_1y_0 + c_1)}{\|\vec{l_1}\|} \\
&= \frac{l_1(X_0)}{\|\vec{l_1}\|}(bb_1 - ca_1) \\
y_F - y_0 &= \frac{(ba_1 - ab_1)(c_1 + a_1x_0) + (ca_1 - bb_1)a_1y_0 - (ab_1^2 - 2ba_1b_1 + ca_1^2)y_0}{ab_1^2 - 2ba_1b_1 + ca_1^2} \\
&= \frac{(ba_1 - ab_1)(c_1 + a_1x_0) + ba_1b_1y_0 - ab_1^2y_0}{\|\vec{l_1}\|} \\
&= \frac{(ba_1 - ab_1)(a_1x_0 + b_1y_0 + c_1)}{\|\vec{l_1}\|} \\
&= \frac{l_1(X_0)}{\|\vec{l_1}\|}(ba_1 - ab_1).
\end{aligned}$$

Combining these expressions with equation 2.21 we finally find

$$\begin{aligned}
Q(X_0, l_1) &= \left(\frac{l_1(X_0)}{\|\vec{l_1}\|}\right)^2 \left(a(bb_1 - ca_1)^2 + 2b(bb_1 - ca_1)(ba_1 - ab_1) + c(ba_1 - ab_1)^2\right) \\
&= \left(\frac{l_1(X_0)}{\|\vec{l_1}\|}\right)^2 \left((ac^2 - b^2c)a_1^2 - 2(abc - b^3)a_1b_1 + (a^2c - ab^2)b_1^2\right) \\
&= \left(\frac{l_1(X_0)}{\|\vec{l_1}\|}\right)^2 (ac - b^2)(ca_1^2 - 2ba_1b_1 + ab_1^2) \\
&= \frac{l_1(X_0)^2}{\|\vec{l_1}\|}\Delta_G. \qquad (2.22)
\end{aligned}$$

□

The results in this chapter show how ideas from Euclidean geometry can be expressed in planar universal geometry. In the following chapter we will use these ideas to build up a theory of conics.

Chapter 3

Conics

3.1 Introduction

Having developed a set of theorems for points and lines, the next most complex geometrical objects to consider are conics. In this chapter we will present the core theory of conics in planar universal geometry, drawing inspiration from previous work done on conics in chromogeometry [7]. Throughout this chapter we assume that all values used are taken from a given field $\mathbb{F}$ not of characteristic two.

We begin by establishing some basic definitions and notation.

Definition 21. Given numbers A, B, C, D, E, F, a *conic* is defined as the locus of points $[x, y]$ such that $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$.

We note that a conic is defined independently of any geometry. When associated with a particular geometry, a conic may take on particular interesting properties, which will be discussed below.

To assist in working with conics we introduce two notations for their representation.

Definition 22. The conic which satisfies $Ax^2+Bxy+Cy^2+Dx+Ey+F = 0$ is represented by the notations $\langle A\!:\!B\!:\!C\!:\!D\!:\!E\!:\!F\rangle$ or $\langle P\!:\!\vec{q}\!:\!r\rangle$ where

$$\begin{aligned} P &= \begin{pmatrix} A & B/2 \\ B/2 & C \end{pmatrix} \\ \vec{q} &= \begin{pmatrix} D \\ E \end{pmatrix} \\ r &= F. \end{aligned}$$

Throughout this chapter we will use the convention that the conic $\langle P : \vec{q} : r\rangle$ corresponds to $\langle A : B : C : D : E : F\rangle$ without explicitly stating this relation.

Definition 23. Two conics, $\langle P : \vec{q} : r\rangle$ and $\langle P' : \vec{q}' : r'\rangle$ are defined to be equivalent if

$$\begin{aligned} P &= \lambda P' && (3.1)\\ \vec{q} &= \lambda \vec{q}' && (3.2)\\ r &= \lambda r' && (3.3) \end{aligned}$$

for some $\lambda \neq 0$.

We also introduce a function $M : \mathbb{F} \times \mathbb{F} \to M_2(\mathbb{F})$ defined by

$$M(a, b) \mapsto \begin{pmatrix} a^2 & ab \\ ab & b^2 \end{pmatrix}. \qquad (3.4)$$

This function arises naturally in a number of places and has some useful properties.

- $\det(M(a, b)) = 0$.
- $M(a, b) = 0$ iff $a = b = 0$.
- If exactly one of a and b are zero then $M(a, b)$ has one non-zero entry.
- If both a and b are non-zero then $M(a, b)$ has no zero entries.

3.2 Tangents, Poles and Polars

Tangents, while being a geometrical construct, are often considered in the study of analysis, where they are defined in terms of infinitesimal limits. As has been shown for Euclidean geometry, it is possible to define tangents to conics in a purely algebraic manner [1]. Indeed the definition of a tangent does not depend on a metric, and so the results below serve to express results known from Euclidean geometry in the notation of this thesis.

Definition 24. A line is a *tangent* to a conic if it intersects that conic at exactly one point.

Theorem 12 (Tangent condition)**.** The line $\langle a\!:\!b\!:\!c\rangle$ is tangent to the conic $\langle P\!:\!\vec{q}\!:\!r\rangle$ if and only if

$$\begin{aligned}(Bc + Db + Ea)^2 - 4(ACc^2 + AFb^2 + CFa^2) + \\ 4\left((AE - BD)bc + (CD - BE)ac + (BF - DE)ab\right) &= 0. \qquad (3.5)\end{aligned}$$

Proof. If the line is a tangent then we need to solve the simultaneous equations

$$\begin{aligned}Ax^2 + Bxy + Cy^2 + Dx + Ey + F &= 0\\ ax + by + c &= 0\end{aligned}$$

to find the unique point $X = [x, y]$. From the equation of the line we get $x = -\frac{c+by}{a}$, which, when substituted into the conic equation, gives

$$\begin{aligned}0 &= A\left(\frac{c+by}{a}\right)^2 - B\left(\frac{c+by}{a}\right)y + Cy^2 - D\left(\frac{c+by}{a}\right) + Ey + F\\ &= A(c+by)^2 - Ba(c+by)y + Ca^2y^2 - Da(c+by) + Ea^2y + Fa^2\\ &= (Ab^2 - Bab + Ca^2)y^2 + (2Abc - Bac - Dab + Ea^2)y + (Ac^2 - Dac + Fa^2).\end{aligned}$$

This equation is quadratic in y, so for there to be a unique solution we require the determinant in the quadratic equation to be zero. This leads to the condition

$$(2Abc - Bac - Dab + Ea^2)^2 = 4(Ab^2 - Bab + Ca^2)(Ac^2 - Dac + Fa^2).$$

Algebraic manipulation (omitted here for brevity) leads to the desired result. □

Theorem 13 (Tangent through a point)**.** Given a point X_0 which lies on the conic $\langle P\!:\!\vec{q}\!:\!r\rangle$, the tangent to the conic through the point is $\langle a\!:\!b\!:\!c\rangle$ where

$$\begin{aligned}\begin{pmatrix}a\\ b\end{pmatrix} &= \vec{q} + 2P\vec{X}_0 \qquad (3.6)\\ c &= \vec{q}\cdot\vec{X}_0 + 2r. \qquad (3.7)\end{aligned}$$

Proof. The relations above can be shown to satisfy equation (3.5). The algebraic manipulations required to verify this are best attempted in a computer algebra system since they result in up to 48 terms and as such are omitted here. □

Another metric-free notion which we can borrow from Euclidean geometry is that of poles and polars.

Definition 25. Given a point X_0 and a conic $\langle P : \vec{q} : r\rangle$, we can generally construct two tangents to the conic which pass through X_0. If we denote the points where these tangents meet the conic as X_1 and X_2 then the line which passes through X_1 and X_2 is defined as the *polar* of the point X_0, the *pole.*

Theorem 14 (Polar from pole)**.** Given a point X_0 and a conic $\langle P:\vec{q}:r\rangle$, the polar of the pole X_0, with respect to the conic, is $\langle a:b:c\rangle$ where

$$\begin{pmatrix} a \\ b \end{pmatrix} = \vec{q} + 2P\vec{X_0} \tag{3.8}$$

$$c = \vec{q}\cdot\vec{X_0} + 2r. \tag{3.9}$$

Proof. From Theorem 13 we know that the tangent line which passes through through X_0 (the pole) and X_1, the tangent point on the conic, must satisfy the equation

$$(\vec{q} + 2P\vec{X_1})\cdot\vec{X_0} + \vec{q}\cdot\vec{X_1} + 2r = 0.$$

Likewise, the tangent through X_2 and X_0 gives

$$(\vec{q} + 2P\vec{X_2})\cdot\vec{X_0} + \vec{q}\cdot\vec{X_2} + 2r = 0.$$

Adding these two equations we get

$$\begin{aligned} 2\vec{q}\cdot\vec{X_0} + 2(\vec{X_1}+\vec{X_2})P\vec{X_0} + \vec{q}\cdot(\vec{X_1}+\vec{X_2}) + 4r &= 0 \\ (\vec{q}+2P\vec{X_0})\cdot(\vec{X_1}+\vec{X_2}) + 2(\vec{q}\cdot\vec{X_0}+2r) &= 0. \end{aligned}$$

As such the line which passes through X_1 and X_2 is

$$(\vec{q}+2P\vec{X_0})\cdot\vec{X} + \vec{q}\cdot\vec{X_0} + 2r = 0.$$

□

Corollary 15. The tangent through a point on a conic is also the polar of that point.

Proof. This follows directly from the equations in Theorems 13 and 14. □

Theorem 16 (Pole from polar)**.** Given a conic $\langle P:\vec{q}:r\rangle$ and a polar $\langle a:b:c\rangle$, the corresponding pole is

$$\vec{X_0} = \frac{1}{2}P^{-1}\begin{pmatrix} \lambda a - D \\ \lambda b - E \end{pmatrix} \tag{3.10}$$

where

$$\lambda = \frac{D(CD - BE/2) + E(AE - BD/2) - 4\Delta_P F}{D(Ca - Bb/2) + E(Ab - Aa/2) - 2\Delta_{PC}}. \tag{3.11}$$

Proof. Equation (3.8) gives the vector form of the polar, up to a scale factor. Rearranging this equation we get

$$\begin{aligned}\vec{X}_0 &= \frac{P^{-1}}{2}\left(\lambda\begin{pmatrix}a\\b\end{pmatrix} - \vec{q}\right)\\ &= \frac{1}{2\Delta_P}\begin{pmatrix}C & -B/2\\ -B/2 & A\end{pmatrix}\begin{pmatrix}\lambda a - D\\ \lambda b - E\end{pmatrix}.\end{aligned}$$

We can find the scale factor λ by combining this result with equation (3.9) to give

$$\begin{aligned}\lambda c &= \vec{q}\cdot\vec{X}_0 + 2r\\ &= \frac{1}{2\Delta_P}D\left(C(\lambda a - D) - B(\lambda b - E)/2\right) +\\ &\quad \frac{1}{2\Delta_P}E(-B(\lambda a - D)/2 + A(\lambda b - E)) + 2F\\ \lambda &= \frac{\frac{1}{2\Delta_P}(D(CD - BE/2) + E(AE - BD/2) - 2F}{\frac{1}{2\Delta_P}(D(Ca - Bb/2) + E(Ab - Aa/2)) - c}\\ &= \frac{D(CD - BE/2) + E(AE - BD/2) - 4\Delta_P F}{D(Ca - Bb/2) + E(Ab - Aa/2) - 2\Delta_P c}.\end{aligned}$$

□

3.3 General Conics

Having established a notation and some metric-free results about conics, we now look at ways of constructing conics based on structures such as points and lines. These constructions will depend intrinsically on the metric of the geometry. The results found here will generalise certain definitions and results known from chromogeometry. We begin with the most general construction.

Definition 26. Given a point X_0 and a line l_1 a *general conic* is defined as the locus of points which have a constant ratio of quadrance to X_0 and quadrance to l_1. The ratio, K, cannot be zero. The point X_0 is called the *focus* and the line l_1 is the *directrix.*

The points on the conic satisfy the equation

$$Q(X, X_0) = KQ(X, l_1). \tag{3.12}$$

Theorem 17 (General form of a conic)**.** *The general form of a conic with focus X_0, directrix l_1 and constant K is $\langle P : \vec{q} : r \rangle$ where*

$$\begin{aligned} P &= G - K\alpha M(a_1, b_1) && (3.13)\\ \vec{q} &= -2\left(G\vec{X}_0 + c_1 K\alpha \begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right) && (3.14)\\ r &= \vec{X}_0^{\;2} - K\alpha c_1^2 && (3.15) \end{aligned}$$

and $\alpha = \Delta_G / \|\vec{l}_1\|$.

Proof. Expanding equation (3.12) we get

$$\begin{aligned} Q(X, X_0) &= KQ(X, l_1)\\ \vec{X}\cdot\vec{X} - 2\vec{X}\cdot\vec{X}_0 + \vec{X}_0^{\;2} &= K\frac{l_1(X)^2}{\|\vec{l}_1\|}\Delta_G\\ &= K\alpha l_1(X)^2\\ &= K\alpha(a_1x + b_1y + c_1)^2\\ &= K\alpha((a_1^2x^2 + 2a_1b_1xy + b_1^2y^2) + 2c_1(a_1x + b_1y) + c_1^2). \end{aligned}$$

□

The quadratic term P in any conic can tell us a lot about the nature of the conic, as we see here and below when we consider circles and parabolas.

Lemma 18. *A general conic $\langle P : \vec{q} : r \rangle$ satisfies $P \neq G$.*

Proof. If $P = G$ then from (3.13) we require $K\alpha M(a_1, b_1) = 0$ for some $l_1 = \langle a_1 : b_1 : c_1 \rangle$. From the definition of a line we must have either $a_1 \neq 0$ or $b_1 \neq 0$ so we require either $K = 0$ or $\alpha = 0$. We know that $\alpha \neq 0$ since $\Delta_G \neq 0$. We also know that $K \neq 0$ by definition and therefore $P \neq G$. □

Lemma 19. *A general conic $\langle P : \vec{q} : r \rangle$ satisfies $G \neq \lambda P$ for $\lambda \neq 0$.*

Proof. Assume that $G = \lambda P$. From Lemma 18 we can take $\lambda \neq 1$. From equation (3.13) we have

$$\begin{aligned} P &= G - K\alpha M(a_1, b_1)\\ \left(1 - \frac{1}{\lambda}\right)G &= K\alpha M(a_1, b_1). \end{aligned}$$

Taking the determinant of each side we get

$$\begin{aligned}\left(1-\frac{1}{\lambda}\right)\Delta_G &= K\alpha\det(M(a_1,b_1))\\ &= 0\end{aligned}$$

which is a contradiction, since by definition $\Delta_G \neq 0$. □

Some theorems below will require that the matrix $G-P$, for some conic $\langle P:\vec{q}:r\rangle$ in a geometry G, has no zero entries. Pre-empting this, we present the following two results.

Lemma 20. For a general conic $\langle P:\vec{q}:r\rangle$ the matrix $G-P$ has either 1 or 4 non-zero entries.

Proof. Since $G-P=K\alpha M(a_1,b_1)$ and at least one of a_1, b_1 are non-zero the result follows directly from the properties of the function M. □

Lemma 21. For a general conic $\langle P:\vec{q}:r\rangle$, if the matrix $G-P$ has 1 non-zero entry, there exists an equivalent conic $\langle P':\vec{q}':r'\rangle$ such that $G-P'$ has 4 non-zero entries.

Proof. If we pick $\lambda \notin \{0, c/C, a/A\}$ and let $P'=\lambda P$ then

$$G-P' = \begin{pmatrix} a-\lambda A & b-\lambda B/2\\ b-\lambda B/2 & c-\lambda C\end{pmatrix}.$$

From our choice of λ we know that $a-\lambda A\neq 0$ and $c-\lambda C\neq 0$, which means $G-P'$ has at least two non-zero entries. Since $G-P'$ must have either one or four non-zero entries, it must have four non-zero entries. □

Having found a way to construct conics from a focus and a directrix, we would like to be able to, given a conic, recover the focus and directrix used to construct it. The remainder of this section addresses this problem.

Definition 27. If l_1 is a directrix of a conic then $\vec{l_1}$ is a *directrix vector* of the conic.

Theorem 22 (Direction of directrices)**.** The conic $\langle P:\vec{q}:r\rangle$ has directrix vectors $\vec{l_1}=\begin{pmatrix}-b_1\\ a_1\end{pmatrix}$ and $\vec{l_2}=\begin{pmatrix}-b_2\\ a_2\end{pmatrix}$ where

$$\begin{aligned}\begin{pmatrix} a_1 & a_2\\ b_1 & b_2\end{pmatrix} &= G-P &(3.16)\\ &= K\alpha M(a_1,b_1). &(3.17)\end{aligned}$$

Proof. We wish to solve equation (3.13) for a_1 and b_1. Without loss of generality we can assume that $G - P$ has no zero entries, which means that a_1 are b_1 are both non-zero. This lets us find an expression for $K\alpha$ by considering the off-diagonal elements of our matrix equation.

$$\begin{aligned} P &= G - K\alpha M(a_1, b_1) \\ \begin{pmatrix} a - A & b - B/2 \\ b - B/2 & c - C \end{pmatrix} &= K\alpha \begin{pmatrix} a_1^2 & a_1 b_1 \\ a_1 b_1 & b_1^2 \end{pmatrix} \\ K\alpha &= \frac{b - B/2}{a_1 b_1}. \qquad (3.18) \end{aligned}$$

We can now use this value to equate the diagonal elements of the matrices above, giving

$$\begin{aligned} a_1(b - B/2) &= b_1(a - A) \\ b_1(b - B/2) &= a_1(c - C). \end{aligned}$$

There are two possible solutions to these equations. $a_1 = a - A$, $b_1 = b - B/2$ or $a_1 = b - B/2$, $b_1 = c - C$. Putting these results into matrix form gives the desired result. □

Lemma 23. The two directrix vectors of a conic are parallel.

Proof. Since $G - P = K\alpha M(a_1, b_1)$ and $\det(M(a_1, b_1)) = 0$, we must have $\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} = a_1 b_2 - a_2 b_1 = 0$ and therefore the directrix vectors are parallel. □

Although we have not yet found the focus and directrix, we are now in a position to determine the constant K.

Theorem 24 (Constant of focus/directrix pairs)**.** The constant K associated with the focus/directrix pairs of $\langle P : \vec{q} : r \rangle$ is

$$K = \frac{a(b - B/2)^2 - 2b(a - A)(b - B/2) + c(a - A)^2}{(a - A)\Delta_G}. \qquad (3.19)$$

Proof. From Theorem 22 and equation (3.18) we get

$$\begin{aligned} K &= \frac{b - B/2}{\alpha a_1 b_1} \\ &= \frac{b - B/2}{\alpha(a - A)(b - B/2)} \\ &= \frac{\|\vec{l_1}\|}{\alpha(a - A)\Delta_G} \\ &= \frac{a(b - B/2)^2 - 2b(a - A)(b - B/2) + c(a - A)^2}{(a - A)\Delta_G}. \end{aligned}$$

□

We have shown that the two directrix vectors are parallel, so in actual fact we only have a single direction for our directrices. Although in Euclidean geometry we expect a conic to have two focus and directrix pairs, to prove this in planar universal geometry we need the following result.

Theorem 25. Given a conic $\langle P : \vec{q} : r\rangle$ with directrix vector $\vec{l_1}$ and constant K, its two directrices are $l_1 = \langle a_1 : b_1 : c_{11}\rangle$ and $l_2 = \langle a_1 : b_1 : c_{12}\rangle$ where c_{11} and c_{12} are the roots of the equation

$$0 = 4K\alpha(\|\vec{l_1}\|K\alpha - \Delta_G)c_1^2 + 4K\alpha(\vec{Q}\cdot\vec{l_1})c_1 + (\vec{Q}\cdot\vec{Q} - 4\Delta_G F) \quad (3.20)$$

where

$$\vec{Q} = \begin{pmatrix} -E \\ D \end{pmatrix} \quad (3.21)$$

$$\alpha = \frac{\Delta_G}{\|\vec{l_1}\|}. \quad (3.22)$$

Proof. We need to solve equations (3.14) and (3.15) for c_1. We begin by finding an expression for X_0.

$$\begin{aligned}
\vec{q} &= -2\left(G\vec{X_0} + c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right) \\
2G\vec{X_0} &= -\vec{q} - 2c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \\
2\vec{X_0} &= -G^{-1}\left(\vec{q} + 2c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right). \quad (3.23)
\end{aligned}$$

We need to find $\vec{X_0}^2$ so we take the metric dot product of each side of the above equation with itself and obtain

$$\begin{aligned}
(2\vec{X_0})^T(2G\vec{X_0}) &= \left(G^{-1}\left(\vec{q} + 2c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right)\right)^T\left(\vec{q} + 2c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right) \\
4\vec{X_0}^2 &= \frac{1}{\Delta_G}\begin{pmatrix} c(D + 2a_1c_1K\alpha) - b(E + 2b_1c_1K\alpha) \\ -b(D + 2a_1c_1K\alpha) + a(E + 2b_1c_1K\alpha) \end{pmatrix}\begin{pmatrix} D + 2a_1c_1K\alpha \\ E + 2b_1c_1K\alpha \end{pmatrix} \\
4\Delta_G\vec{X_0}^2 &= c(D + 2a_1c_1K\alpha)^2 - 2b(D + 2a_1c_1K\alpha)(E + 2b_1c_1K\alpha) + \\
&\quad a(E + 2b_1c_1K\alpha)^2.
\end{aligned}$$

We are now able to solve equation (3.15) in terms of c_1.

$$\begin{aligned}
r &= \vec{X_0}^2 - K\alpha c_1^2 \\
4\Delta_G(F + c_1^2 K\alpha) &= c(D + 2a_1c_1K\alpha)^2 - 2b(D + 2a_1c_1K\alpha)(E + 2b_1c_1K\alpha) + \\
&\quad a(E + 2b_1c_1K\alpha)^2 \\
0 &= (4ca_1^2K^2\alpha^2 - 8ba_1b_1K^2\alpha^2 + 4ab_1K^2\alpha^2 - 4\Delta_G K\alpha)c_1^2 \\
&\quad +(4cDa_1K\alpha - 2b(2Db_1K\alpha + 2Ea_1K\alpha) + 4aEb_1K\alpha)c_1 \\
&\quad +(cD^2 - 2bDE + aE^2 - 4\Delta_G F) \\
&= 4K\alpha(\|\vec{l_1}\|K\alpha - \Delta_G)c_1^2 + 4K\alpha(\vec{Q}\cdot\vec{l_1})c_1 + (\vec{Q}\cdot\vec{Q} - 4\Delta_G F).
\end{aligned}$$

□

Having found that there are indeed two directrices, finding their associated foci is relatively simple.

Theorem 26 (Focus of a directrix)**.** Given a conic $\langle P : \vec{q} : r\rangle$ with directrix $l_1 = \langle a_1 : b_1 : c_1\rangle$ and constant K, the associated focus is

$$\vec{X_0} = -\frac{1}{2}G^{-1}\left(\vec{q} + 2c_1K\alpha\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}\right). \tag{3.24}$$

Proof. This result follows directly from (3.23). □

3.4 Circles

In the construction of conics from focus and directrix we found that we were unable to construct conics such that $G = \lambda P$. We now investigate another way of constructing conics which complements the focus/directrix construction.

Definition 28. Given a geometry, G, a *circle* is defined as the locus of points which are quadrance K from a fixed point X_0, i.e. those points which satisfy

$$Q(X, X_0) = K. \tag{3.25}$$

The point X_0 is the *centre* of the circle and K is its *quadrance.*

Theorem 27 (General form of a circle)**.** The general form of a circle with radius K and centre X_0 is $\langle P : \vec{q} : r\rangle$ where

$$\begin{aligned}
P &= G && (3.26)\\
\vec{q} &= -2G\vec{X_0} && (3.27)\\
r &= \vec{X_0}^2 - K. && (3.28)
\end{aligned}$$

Proof. Expanding (3.25) we get

$$\begin{aligned} Q(X, X_0) &= K \\ (\vec{X} - \vec{X_0}) \cdot (\vec{X} - \vec{X_0}) &= K \\ \vec{X} \cdot \vec{X} - 2\vec{X} \cdot \vec{X_0} + \vec{X_0}^2 - K &= 0. \end{aligned}$$

□

Lemma 28. A conic $\langle P : \vec{q} : r \rangle$ is not a circle in the geometry G if $G \neq \lambda P$ for some $\lambda \neq 0$.

Proof. If $\langle P : \vec{q} : r \rangle$ is a circle, then from the general form given above there exists an equivalent conic $\langle P' : \vec{q}' : r' \rangle$ such that $P' = \lambda P = G$ for $\lambda \neq 0$. If no such P' can be found then the conic is not a circle. □

Theorem 29. If a conic $\langle P : \vec{q} : r \rangle$ in the geometry G satisfies $\lambda P = G$ with $\lambda \neq 0$ then it is a circle in G with centre and quadrance given by

$$\vec{X_0} = -\frac{\lambda}{2} G^{-1} \vec{q} \tag{3.29}$$

$$K = \vec{X_0}^2 - \lambda r. \tag{3.30}$$

Proof. These results follow directly from (3.27) and (3.28). □

Corollary 30. A conic $\langle P : \vec{q} : r \rangle$ is a circle in a geometry G if and only if $G = \lambda P$ with $\lambda \neq 0$.

Proof. This is essentially a restatement of Lemma 28 and Theorem 29 combined. □

Corollary 31. Every conic $\langle P : \vec{q} : r \rangle$ where $\det(P) \neq 0$ is a circle in some geometry G.

Proof. If we let $G = P$ then G is a valid geometry, since $\det(G) \neq 0$ and P is a circle in G. □

A result of Euclidean geometry which is taught to all high school students is that the radius of a circle to a point on the circle is perpendicular to the tangent at that point. In planar universal geometry we can obtain a more general result, which contains the equivalent of the radius-tangent rule as a special case.

Theorem 32 (Pole to a circle)**.** Given a point X_1 and a circle with centre X_0, the polar of X_1 is l_1 where

$$\begin{aligned} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} &= 2G(\vec{X_1} - \vec{X_0}) && (3.31) \\ c_1 &= 2\vec{X_0}G(\vec{X_0} - \vec{X_1}) - 2K. && (3.32) \end{aligned}$$

Proof. Combining the equations for the polar from Theorem 14 and the general form of a circle we get

$$\begin{aligned} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} &= \vec{q} + 2P\vec{X_1} \\ &= -2G\vec{X_0} + 2G\vec{X_1} \\ &= 2G(\vec{X_1} - \vec{X_0}) \\ c_1 &= \vec{q} \cdot \vec{X_1} + 2r \\ &= (-2G\vec{X_0}) \cdot \vec{X_1} + 2(\vec{X_0}^2 - K) \\ &= 2\vec{X_0}G(\vec{X_0} - \vec{X_1}) - 2K. \end{aligned}$$

□

Theorem 33. Given a point X_1 and a circle with centre X_0, the line which passes through the centre of the circle and X_1 is perpendicular to the polar of X_1.

Proof. We let $r = \langle y_0 - y_1 : x_1 - x_0 : x_0 y_1 - x_1 y_0 \rangle$ be the line through the centre, X_0 and the point X_1. If we denote the tangent vector as $\vec{t}$ then from the previous theorem we have

$$\begin{aligned} \vec{t} &= 2\begin{pmatrix} -b(x_0 - x_1) - c(y_0 - y_1) \\ a(x_0 - x_1) + b(y_0 - y_1) \end{pmatrix} \\ \vec{r} &= \begin{pmatrix} x_0 - x_1 \\ y_0 - y_1 \end{pmatrix} \\ \vec{t} \cdot \vec{r} &= 2\begin{pmatrix} -b(x_0 - x_1) - c(y_0 - y_1) \\ a(x_0 - x_1) + b(y_0 - y_1) \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x_0 - x_1 \\ y_0 - y_1 \end{pmatrix} \\ &= 2\begin{pmatrix} -b(x_0 - x_1) - c(y_0 - y_1) \\ a(x_0 - x_1) + b(y_0 - y_1) \end{pmatrix} \begin{pmatrix} a(x_0 - x_1) + b(y_0 - y_1) \\ b(x_0 - x_1) + c(y_0 - y_1) \end{pmatrix} \\ &= 0. \end{aligned}$$

□

Corollary 34. Given a point X_1 on a circle, the tangent passing through this point is perpendicular to the line through X_1 and the centre of the circle.

Proof. Since the tangent through X_1 is also the polar of X_1 this follows directly from Theorem 33. □

3.5 Parabola

Probably the first quadratic equation a maths student will meet is $y = x^2$, which describes a parabola in Euclidean geometry. As one might expect, we can construct such objects in planar universal geometry.

Definition 29. A *parabola* is the locus of points whose quadrance to a point X_0 is equal to its quadrance to a line l_1. A parabola is essentially a general conic with $K = 1$.

Theorem 35. The general form of a parabola with focus X_0, directrix l_1 is $\langle P\!:\!\vec{q}\!:\!r\rangle$ where

$$\begin{aligned} P &= G - \alpha M(a_1, b_1) & (3.33)\\ \vec{q} &= -2(GX_0 + c_1\alpha(a_1, b_2)) & (3.34)\\ r &= X_0^2 - \alpha c_1^2. & (3.35) \end{aligned}$$

Proof. This result follows directly from letting $K = 1$ in equations (3.13)-(3.15). □

Theorem 36. A conic $\langle P\!:\!\vec{q}\!:\!r\rangle$ is a parabola if and only if $\det(P) = 0$.

Proof. Taking determinants of each side in equation (3.33) yields

$$\begin{aligned} |P| &= |G - \alpha M(a_1, b_1)|\\ &= (a - \alpha a_1^2)(c - \alpha b_1^2) - (b - \alpha a_1 b_1)^2\\ &= ac - b^2 - \alpha(ca_1^2 - 2ba_1b_1 + ab_1^2) + \alpha^2(a_1^2b_1^2 - a_1^2b_1^2)\\ &= \Delta_G - \frac{\Delta_G}{\|\vec{l_1}\|}\|\vec{l_1}\|\\ &= 0. \end{aligned}$$

□

Corollary 37. If a conic is a parabola in any geometry then it is a parabola in all geometries.

Proof. This result follows direction from the fact that the conditions in Theorem 36 are independent of the geometry. □

Theorem 38. A parabola has only a single directrix/focus pair.

Proof. To find the directrices of a conic we need to solve equation (3.20), which is

$$0 = 4K\alpha(\|\vec{l_1}\|K\alpha - \Delta_G)c_1^2 + 4K\alpha(\vec{Q}\cdot\vec{l_1})c_1 + (\vec{Q}\cdot\vec{Q} - 4\Delta_G F).$$

For a parabola with $K = 1$ we have $\|\vec{l_1}\|K\alpha - \Delta_G = \|\vec{l_1}\|\frac{\Delta_G}{\|\vec{l_1}\|} - \Delta_G = 0$, which reduces the quadratic equation to a linear equation, which will only have one root. □

Theorem 39. The focus and directrix of a parabola $\langle P : \vec{q} : r\rangle$ are l_1 and X_0 where

$$a_1 = a - A \tag{3.36}$$

$$b_1 = b - B/2 \tag{3.37}$$

$$c_1 = \frac{4\Delta_G F - \vec{Q}\cdot\vec{Q}}{4\alpha(\vec{Q}\cdot\vec{l_1})} \tag{3.38}$$

$$\vec{X_0} = -\frac{1}{2}G^{-1}\left(\vec{q} + 2c_1\alpha\begin{pmatrix}a_1\\ b_1\end{pmatrix}\right) \tag{3.39}$$

and

$$\vec{Q} = \begin{pmatrix}-E\\ D\end{pmatrix} \tag{3.40}$$

$$\alpha = \frac{\Delta_G}{\|\vec{l_1}\|}. \tag{3.41}$$

Proof. The results for a_1, b_1 and X_0 come directly from Theorems 22 and 26 for a general conic. From the proof of Theorem 38 we have seen that c_1 is the solution of the equation

$$0 = 4K\alpha(\vec{Q}\cdot\vec{l_1})c_1 + (\vec{Q}\cdot\vec{Q} - 4\Delta_G F)$$

from which the result directly follows. □

3.6 Grammolas

Conic sections in Euclidean geometry are generally described as parabolas, hyperbolas and ellipses. While planar universal geometry has parabolas as a fundamental object, the notion of hyperbolas and ellipses do not manifest themselves as distinct objects. The simplest example to demonstrate this is the unit circle in the green geometry of chromogeometry. This object has the equation $xy - 1 = 0$, and is, by definition, a circle in the green geometry. The same equation considered in the blue geometry (i.e. regular Euclidean geometry) is not a circle, but a hyperbola.

In place of hyperbolas and ellipses, we have objects known as grammolas and quadrolas. These objects have been studied in the context of chromogeometry [7], and are presented here in the context of planar universal geometry.

Definition 30. A *grammola* is defined as the locus of points such that the sum of quadrances from each point to two given lines is a constant. Given two lines d_1 and d_2, the *diagonals*, and a constant K, the points on a grammola satisfy the equation

$$Q(X, d_1) + Q(X, d_2) = K. \tag{3.42}$$

Theorem 40. Given non-null diagonals $d_1 = \langle a_1 : b_1 : c_1 \rangle$, $d_2 = \langle a_2 : b_2 : c_2 \rangle$ and constant K, the general form of the grammola satisfying $Q(X, d_1)+Q(X, d_2) = K$ is $\langle P : \vec{q} : r \rangle$ where

$$\begin{aligned} P &= \|\vec{d_1}\| M(a_2, b_2) + \|\vec{d_2}\| M(a_1, b_1) && (3.43) \\ \vec{q} &= 2 \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} \begin{pmatrix} \|\vec{d_2}\| c_1 \\ \|\vec{d_1}\| c_2 \end{pmatrix} && (3.44) \\ r &= c_1^2 \|\vec{d_2}\| + c_2^2 \|\vec{d_1}\| - \frac{K \|\vec{d_1}\| \|\vec{d_2}\|}{\Delta_G} && (3.45) \end{aligned}$$

Proof. Starting from the definition and using equation (2.20) we find

$$\begin{aligned} 0 &= Q(X, d_1) + Q(X, d_2) - K \\ &= \frac{d_1(X)^2}{\|\vec{d_1}\|} \Delta_G + \frac{d_2(X)^2}{\|\vec{d_2}\|} \Delta_G - K \\ &= \|\vec{d_2}\| (a_1 x + b_1 y + c_1)^2 + \|\vec{d_1}\| (a_2 x + b_2 y + c_2)^2 - \frac{K \|\vec{d_1}\| \|\vec{d_2}\|}{\Delta_G}. \end{aligned}$$

From this expression the result follows immediately. □

As with the general conics before, we would like to be able to take a grammola and find its diagonals.

Definition 31. A vector $\vec{d_1} = \begin{pmatrix} -b_1 \\ a_1 \end{pmatrix}$ is a *diagonal vector* of the conic $\langle P : \vec{q} : r \rangle$ if and only if

$$\begin{aligned} \frac{2A\|\vec{d_1}\| - a_1^2(Ac - Bb + Ca)}{2\|\vec{d_1}\|^2} &= \lambda^2 & (3.46) \\ \frac{2C\|\vec{d_1}\| + b_1^2(Ac - Bb + Ca)}{2\|\vec{d_1}\|^2} &= \mu^2 & (3.47) \end{aligned}$$

for some $\lambda, \mu \in \mathbb{F}$. The motivation for this definition will become clear below.

Definition 32. If d_1 and d_2 are the diagonals of a conic then d_1 is the *co-diagonal* of d_2 and vice-versa. Likewise, $\vec{d_1}$ is the *co-diagonal vector* of $\vec{d_2}$ and vice-versa.

Theorem 41. If $\vec{d_1}$ is a diagonal vector of the conic $\langle P : \vec{q} : r \rangle$ then its co-diagonal vector is $\vec{d_2}$ where

$$\begin{aligned} a_2^2 &= \frac{2A\|\vec{d_1}\| - a_1^2(Ac - Bb + Ca)}{2\|\vec{d_1}\|^2} & (3.48) \\ b_2^2 &= \frac{2C\|\vec{d_1}\| + b_1^2(Ac - Bb + Ca)}{2\|\vec{d_1}\|^2} & (3.49) \\ \frac{a_2}{b_2} &= \frac{2A\|\vec{d_1}\| - a_1^2(Ac - Bb + Ca)}{B\|\vec{d_1}\| - a_1 b_1(Ac - Bb + Ca)}. & (3.50) \end{aligned}$$

Proof. To complete this proof we need to solve the system of equations represented in equation (3.43). We start by finding expressions for a_2^2, b_2^2 and

a_2b_2.

$$\begin{aligned}
A &= \|\vec{d_1}\|a_2^2 + \|\vec{d_2}\|a_1^2 \\
&= \|\vec{d_1}\|a_2^2 + (ab_2^2 - 2ba_2b_2 + ca_2^2)a_1^2 \\
B &= 2(\|\vec{d_1}\|a_2b_2 + \|\vec{d_2}\|a_1b_1) \\
&= 2\|\vec{d_1}\|a_2b_2 + 2(ab_2^2 - 2ba_2b_2 + ca_2^2)a_1b_1 \\
C &= \|\vec{d_1}\|b_2^2 + \|\vec{d_2}\|b_1^2 \\
&= \|\vec{d_1}\|b_2^2 + (ab_2^2 - 2ba_2b_2 + ca_2^2)b_1^2 \\
a_2^2 &= \frac{A - aa_1^2b_2^2 + 2ba_1^2a_2b_2}{\|\vec{d_1}\| + ca_1^2} \\
a_2b_2 &= \frac{B - 2aa_1b_1b_2^2 - 2ca_1a_2^2b_1}{2\|\vec{d_1}\| - 4ba_1b_1} \\
b_2^2 &= \frac{C - ca_2^2b_1^2 + 2ba_2b_1^2b_2}{\|\vec{d_1}\| + ab_1^2}.
\end{aligned}$$

At this stage we introduce a new symbol, $w = \|\vec{d_1}\| + ab_1^2$ to ease the algebra. Eliminating b_2^2 we get

$$\begin{aligned}
a_2^2 &= \frac{A - aa_1^2\left(\frac{C - ca_2^2b_1^2 + 2ba_2b_1^2b_2}{\|\vec{d_1}\| + ab_1^2}\right) + 2ba_1^2a_2b_2}{\|\vec{d_1}\| + ca_1^2} \\
(\|\vec{d_1}\| + ab_1^2)(\|\vec{d_1}\| + ca_1^2)a_2^2 &= Aw - aa_1^2(C - ca_2^2b_1^2 + 2ba_2b_1^2b_2) + \\
&\quad 2ba_1^2a_2b_2w \\
((\|\vec{d_1}\| + ab_1^2)(\|\vec{d_1}\| + ca_1^2) - aca_1^2b_1^2)a_2^2 &= Aw - Caa_1^2 + 2ba_1^2a_2b_2(w - ab_1^2) \\
\|\vec{d_1}\|(w + ca_1^2)a_2^2 &= Aw - Caa_1^2 + 2ba_1^2a_2b_2\|\vec{d_1}\|
\end{aligned}$$

and

$$\begin{aligned}
a_2b_2 &= \frac{B - 2aa_1b_1\left(\frac{C - ca_2^2b_1^2 + 2ba_2b_1^2b_2}{\|\vec{d_1}\| + ab_1^2}\right) - 2ca_1a_2^2b_1}{2\|\vec{d_1}\| - 4ba_1b_1} \\
(\|\vec{d_1}\| + ab_1^2)(2\|\vec{d_1}\| - 4ba_1b_1)a_2b_2 &= Bw - 2aa_1b_1(C - ca_2^2b_1^2 + 2ba_2b_1^2b_2) - \\
&\quad 2ca_1a_2^2b_1w \\
((\|\vec{d_1}\| + ab_1^2)(2\|\vec{d_1}\| - 4ba_1b_1) + & \\
4aba_1b_1^3)a_2b_2 &= Bw - 2Caa_1b_1 - 2ca_1a_2^2b_1(w - ab_1^2) \\
2\|\vec{d_1}\|(w - 2ba_1b_1)a_2b_2 &= Bw - 2Caa_1b_1 - 2ca_1a_2^2b_1\|\vec{d_1}\| \\
a_2b_2 &= \frac{Bw - 2Caa_1b_1 - 2ca_1a_2^2b_1\|\vec{d_1}\|}{2\|\vec{d_1}\|(w - 2ba_1b_1)}.
\end{aligned}$$

We introduce a new variable $z = Ac - Bb + Ca$ to further simplify the algebra. Substituting the previous expression into our equation for a_2^2 we get

$$\begin{aligned}
\|\vec{d_1}\|(w + ca_1^2)a_2^2 &= (Aw - Caa_1^2) + ba_1^2 \left(\frac{Bw - 2Caa_1b_1 - 2ca_1a_2^2b_1\|\vec{d_1}\|}{(w - 2ba_1b_1)} \right) \\
\|\vec{d_1}\|(w - 2ba_1b_1)(w + ca_1^2)a_2^2 &= (w - 2ba_1b_1)(Aw - Caa_1^2) + ba_1^2(Bw - 2Caa_1b_1) \\
&\quad -2bca_1^3a_2^2b_1\|\vec{d_1}\| \\
\|\vec{d_1}\|((w - 2ba_1b_1)(w + ca_1^2)+ & \\
2bca_1^3b_1)a_2^2 &= Aw^2 - 2Aba_1b_1w - Caa_1^2w + ba_1^2Bw \\
\|\vec{d_1}\|w(w - 2ba_1b_1 + ca_1^2)a_2^2 &= Aw^2 - 2Aba_1b_1w - Caa_1^2w + ba_1^2Bw \\
2\|\vec{d_1}\|^2a_2^2 &= A(w - 2ba_1b_1) - Caa_1^2 + ba_1^2B \\
2\|\vec{d_1}\|^2a_2^2 &= 2A\|\vec{d_1}\| - Aca_1^2 - Caa_1^2 + ba_1^2B \\
2\|\vec{d_1}\|^2a_2^2 &= 2A\|\vec{d_1}\| - a_1^2z \\
a_2^2 &= \frac{2A\|\vec{d_1}\| - a_1^2z}{2\|\vec{d_1}\|^2}.
\end{aligned}$$

We can now use this expression to find a_2b_2.

$$\begin{aligned}
a_2b_2 &= \frac{Bw - 2Caa_1b_1 - 2ca_1b_1\left(\frac{2A\|\vec{d_1}\| - a_1^2z}{2\|\vec{d_1}\|^2}\right)\|\vec{d_1}\|}{2\|\vec{d_1}\|(w - 2ba_1b_1)} \\
&= \frac{\|\vec{d_1}\|(Bw - 2Caa_1b_1) - ca_1b_1(2A\|\vec{d_1}\| - a_1^2z)}{2\|\vec{d_1}\|^2(2\|\vec{d_1}\| - ca_1^2)} \\
&= \frac{\|\vec{d_1}\|(B(2ab_1^2 - 2ba_1b_1 + ca_1^2) - 2Caa_1b_1 - 2Aca_1b_1) + ca_1^3b_1z}{2\|\vec{d_1}\|^2(2\|\vec{d_1}\| - ca_1^2)} \\
&= \frac{\|\vec{d_1}\|(B(2ab_1^2 - 4ba_1b_1 + ca_1^2) - 2a_1b_1z) + ca_1^3b_1z}{2\|\vec{d_1}\|^2(2\|\vec{d_1}\| - ca_1^2)} \\
&= \frac{\|\vec{d_1}\|B(2ab_1^2 - 4ba_1b_1 + 2ca_1^2 - ca_1^2) + a_1b_1(ca_1^2 - 2\|\vec{d_1}\|)z}{2\|\vec{d_1}\|^2(2\|\vec{d_1}\| - ca_1^2)} \\
&= \frac{\|\vec{d_1}\|B(2\|\vec{d_1}\| - ca_1^2) + a_1b_1(ca_1^2 - 2\|\vec{d_1}\|)z}{2\|\vec{d_1}\|^2(2\|\vec{d_1}\| - ca_1^2)} \\
&= \frac{\|\vec{d_1}\|B - a_1b_1z}{2\|\vec{d_1}\|^2}.
\end{aligned}$$

We can now use the values of a_2b_2 and a_2^2 to find b_2^2.

$$\begin{aligned}
b_2^2 &= \frac{C - ca_2^2b_1^2 + 2ba_2b_1^2b_2}{\|\vec{d_1}\| + ab_1^2} \\
&= \frac{C - cb_1^2\left(\frac{2A\|\vec{d_1}\| - a_1^2z}{2\|\vec{d_1}\|^2}\right) + 2bb_1^2\left(\frac{\|\vec{d_1}\|B - a_1b_1z}{2\|\vec{d_1}\|^2}\right)}{\|\vec{d_1}\| + ab_1^2} \\
&= \frac{2\|\vec{d_1}\|^2C - cb_1^2(2A\|\vec{d_1}\| - a_1^2z) + 2bb_1^2(\|\vec{d_1}\|B - a_1b_1z)}{2\|\vec{d_1}\|^2(\|\vec{d_1}\| + ab_1^2)} \\
&= \frac{\|\vec{d_1}\|(2\|\vec{d_1}\|C - 2Acb_1^2 + 2bb_1^2B) + a_1b_1^2(ca_1 - 2bb_1)z}{2\|\vec{d_1}\|^2(\|\vec{d_1}\| + ab_1^2)} \\
&= \frac{\|\vec{d_1}\|(2Cab_1^2 - 4Cba_1b_1 + 2Cca_1^2 - 2Acb_1^2 + 2bb_1^2B) + a_1b_1^2(ca_1 - 2bb_1)z}{2\|\vec{d_1}\|^2(\|\vec{d_1}\| + ab_1^2)} \\
&= \frac{\|\vec{d_1}\|(4Cab_1^2 - 4Cba_1b_1 + 2Cca_1^2) + b_1^2(ca_1^2 - 2ba_1b_1 - 2\|\vec{d_1}\|)z}{2\|\vec{d_1}\|^2(\|\vec{d_1}\| + ab_1^2)} \\
&= \frac{2C\|\vec{d_1}\|(\|\vec{d_1}\| + ab_1^2) + b_1^2(\|\vec{d_1}\| + ab_1^2)z}{2\|\vec{d_1}\|^2(\|\vec{d_1}\| + ab_1^2)} \\
&= \frac{2C\|\vec{d_1}\| + b_1^2z}{2\|\vec{d_1}\|^2}.
\end{aligned}$$

We finally take the ratio of a_2^2 and a_2b_2 to establish the appropriate roots to take when calculating values of a_2 and b_2.

$$\begin{aligned}
\frac{a_2}{b_2} &= \frac{a_2^2}{a_2b_2} \\
&= \frac{2A\|\vec{d_1}\| - a_1^2z}{B\|\vec{d_1}\| - a_1b_1z}.
\end{aligned}$$

□

This result motivates our original definition of the the diagonal vectors as we have expressions for the squares of a_2 and b_2.

Theorem 42. Given a conic $\langle P : \vec{q} : r\rangle$ with diagonal vectors $\vec{d_1}$ and $\vec{d_2}$, the diagonals are d_1 and d_2 where

$$c_1 = \frac{b_2D - a_2E}{2\|\vec{d_2}\|(a_1b_2 - a_2b_1)} \tag{3.51}$$

$$c_2 = \frac{a_1E - b_1D}{2\|\vec{d_1}\|(a_1b_2 - a_2b_1)}. \tag{3.52}$$

Proof. Starting from equation (3.44) we find

$$\begin{aligned}
\vec{q} &= 2\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}\begin{pmatrix} \|\vec{d_2}\|c_1 \\ \|\vec{d_1}\|c_2 \end{pmatrix} \\
\begin{pmatrix} \|\vec{d_2}\|c_1 \\ \|\vec{d_1}\|c_2 \end{pmatrix} &= \frac{1}{2}\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}^{-1}\vec{q} \\
&= \frac{1}{2(a_1b_2 - a_2b_1)}\begin{pmatrix} -b_2 & -a_2 \\ -b_1 & a_1 \end{pmatrix}\begin{pmatrix} D \\ E \end{pmatrix} \\
&= \frac{1}{2(a_1b_2 - a_2b_1)}\begin{pmatrix} b_2D - a_2E \\ a_1E - b_1D \end{pmatrix}.
\end{aligned}$$

□

Theorem 43. Given a conic $\langle P : \vec{q} : r\rangle$ with diagonals d_1 and d_2, the grammola constant K is given by

$$K = \Delta_G\left(\frac{c_1^2}{\|\vec{d_1}\|} + \frac{c_2^2}{\|\vec{d_2}\|} - \frac{F}{\|\vec{d_1}\|\|\vec{d_2}\|}\right). \tag{3.53}$$

Proof. This result follows directly from equation (3.45). □

Corollary 44. Given a conic $\langle P : \vec{q} : r\rangle$ and a single diagonal vector $\vec{d_1}$, we can calculate the diagonals d_1 and d_2 as well as the grammola constant K.

Proof. From Theorem 41 we can find the co-diagonal vector $\vec{d_2}$. Theorem 42 then lets us find the diagonals d_1 and d_2 which in turn can be used in Theorem 43 to find the constant K. □

3.7 Quadrolas

While grammolas are defined with respect to two lines, the diagonals, we define their counterpart, the quadrola, with respect to two points.

Definition 33. Given two points, X_1 and X_2, and a constant K, a *quadrola* is defined as the locus of points satisfying the equation $A(Q(X, X_1), Q(X, X_2), K) = 0$.

Theorem 45. Given two points, $X_0 = [x_0, y_0]$ and $X_1 = [x_1, y_1]$, and a constant K, the general form of their quadrola is $\langle P : \vec{q} : r \rangle$ where

$$\begin{aligned} P &= 4M(a\Delta x + b\Delta y, b\Delta x + c\Delta y) - 4KG && (3.54) \\ \vec{q} &= 4(\vec{X_0}^2 - \vec{X_1}^2) G \Delta\vec{X} + 4KG(\vec{X_0} + \vec{X_1}) && (3.55) \\ r &= \left(K - \vec{X_0}^2 - \vec{X_1}^2\right)^2 - 4\vec{X_0}^2\vec{X_1}^2 && (3.56) \end{aligned}$$

and $\Delta\vec{X} = \vec{X_1} - \vec{X_0}$, $\Delta x = x_1 - x_0$ and $\Delta y = y_1 - y_0$.

Proof. If we let $Q_1 = Q(X, X_1)$ and $Q_2 = Q(X, X_2)$, then from the definition and equation (2.11) we have

$$\begin{aligned} 0 &= (Q_1 + Q_2 + K)^2 - 2(Q_1^2 + Q_2^2 + K^2) \\ &= Q_1^2 + Q_2^2 + K^2 - 2(Q_1Q_2 + KQ_1 + KQ_2) \\ &= (Q_1 - Q_2)^2 - 2K(Q_1 + Q_2) + K^2. \end{aligned}$$

Expressing this in terms of X, and using metric dot products we have

$$\begin{aligned} 0 &= \left(\left(\vec{X}^2 - 2(\vec{X}\cdot\vec{X_0}) + \vec{X_0}^2\right) - \left(\vec{X}^2 - 2(\vec{X}\cdot\vec{X_1}) + \vec{X_1}^2\right)\right)^2 \\ &\quad -2K\left(\left(\vec{X}^2 - 2(\vec{X}\cdot\vec{X_0}) + \vec{X_0}^2\right) + \left(\vec{X}^2 - 2(\vec{X}\cdot\vec{X_1}) + \vec{X_1}^2\right)\right) + K^2 \\ &= \left(2\vec{X}\cdot(\vec{X_1} - \vec{X_0}) + (\vec{X_0}^2 - \vec{X_1}^2)\right)^2 \\ &\quad -2K\left(2\vec{X}^2 - 2\vec{X}\cdot(\vec{X_0} + \vec{X_1}) + \vec{X_0}^2 + \vec{X_1}^2\right) + K^2 \\ &= \left(2\vec{X}\cdot(\vec{X_1} - \vec{X_0})\right)^2 + 4(\vec{X_0}^2 - \vec{X_1}^2)\vec{X}\cdot(\vec{X_1} - \vec{X_0}) + \left(\vec{X_0}^2 - \vec{X_1}^2\right)^2 \\ &\quad -2K\left(2\vec{X}^2 - 2\vec{X}\cdot(\vec{X_0} + \vec{X_1}) + \vec{X_0}^2 + \vec{X_1}^2\right) + K^2 \\ &= 4\left(\vec{X}\cdot(\vec{X_1} - \vec{X_0})\right)^2 - 4K\vec{X}^2 \\ &\quad +4(\vec{X_0}^2 - \vec{X_1}^2)\vec{X}\cdot(\vec{X_1} - \vec{X_0}) + 4K\vec{X}\cdot(\vec{X_0} + \vec{X_1}) \\ &\quad -2K(\vec{X_0}^2 + \vec{X_1}^2) + \left(\vec{X_0}^2 - \vec{X_1}^2\right)^2 + K^2 \\ &= 4\left(\left(\vec{X}\cdot(\vec{X_1} - \vec{X_0})\right)^2 - K\vec{X}^2\right) \\ &\quad +4\left((\vec{X_0}^2 - \vec{X_1}^2)\vec{X}\cdot(\vec{X_1} - \vec{X_0}) + K\vec{X}\cdot(\vec{X_0} + \vec{X_1})\right) \\ &\quad +\left(K - \vec{X_0}^2 - \vec{X_1}^2\right)^2 - 4\vec{X_0}^2\vec{X_1}^2. \end{aligned}$$

□

The inverse problem for a quadrola is significantly more difficult than for a grammola, since the equations we need to solve are quartic. As such, the inverse problem is not addressed here, but would be a natural progression in the development of the theory of quadrolas in planar universal geometry.

Chapter 4

Pygeom Library

Having established the core mathematical theories involved in planar universal geometry, it will be helpful to have a tool which allows us to perform the calculations involved in any practical application. In this chapter we present the `pygeom` library, a Python package for performing calculations in planar universal geometry.

The classes, functions and interfaces provided by the library are described here. While full technical detail is avoided, it is assumed that the reader has a basic understanding of the Python language[1].

4.1 Overview

The `pygeom` library contains a number of modules, each of which implements a particular piece of functionality. The modules described here are the `field`, `geometry`, `core` and `pairs` modules, which constitute the library API intended for the user.

These modules form a natural hierarchy, with each building on the functionality provided by the previous. From simplest to most complex, the hierarchy of modules is `field` $\rightarrow$ `geometry` $\rightarrow$ `core` $\rightarrow$ `pairs`.

[1]For language references, tutorials and downloads see http://www.python.org

4.2 Fields

The underlying premise of universal geometry is that all calculations are performed over some field $\mathbb{F}$, not of characteristic two. It stands to reason that `pygeom` must be able to support field calculations over standard fields. In particular `pygeom` currently supports the fields $\mathbb{Q}$ and $\mathbb{Z}_p$ for primes p. The design also provides the flexibility for users to define their own fields if they need to perform calculations in a non-supported field.

4.2.1 `Field`

Like most procedural programming languages, Python has operators for adding, subtracting, multiplying and dividing variables. Since these represent the core field operations, we would like to be able to use them to operate on the variables we use to represent members of a field.

Another property of fields is that they each contain a zero element and a one element. Combined with the addition operator, it can be seen that every integer has a natural representation in any given field. As such we would like to support the addition, multiplication, etc of field variables with regular Python integers.

Finally, from the closure property of fields, we expect that any mathematical operation on a field variable would return a new variable from the same field.

These design considerations are all addressed in the implementation of fields in `pygeom`.

The class `Field` (Figure 4.1), from `field.py`, provides a mostly abstract base class from which the classes for different fields should derive. It contains a number of methods, all of which must be implemented by any subclass. The interface for this class overrides all of the standard mathematical operators, as well as equality testing. It also provides a small number of non-operator methods.

The `is_square()` method is a boolean method which determines whether a value is a square number in the field. The `sqrt()` will return the square root of a number, assuming that it is indeed a square in the field. The `reduce()` method takes a set of numbers from the field and scales them all by a common factor to get them into a standard form, which is subclass specific. This method is particularly useful in finding standard representations of lines and conics. Finally, the class method `random()` creates a random element of the

```
class Field(object):

    def __init__(self, value, *args, **kwargs):

    def __add__(self, other):

    def __sub__(self, other):

    def __mul__(self, other):

    def __neg__(self):

    def __div__(self, other):

    def __radd__(self, other):

    def __rsub__(self, other):

    def __rmul__(self, other):

    def __rdiv__(self, other):

    def __eq__(self, other):

    def __ne__(self, other):

    def is_square(self):

    def sqrt(self):

    def reduce(self, others):

    @classmethod
    def random(cls):
```

Figure 4.1: The `Field` class interface.

class. This method is particularly useful when generating test data.

4.2.2 `Rational`

The `Rational` class is a subclass of `Field` and implements the field $\mathbb{Q}$. Rational variables can be created either from pairs of integers, representing the numerator and denominator, or from individual integers. Internally, a standard `gcd` algorithm is used to ensure the the rational number is always stored in lowest terms. Since Python supports arbitrarily large integers, the `Rational` class can be used to represent fractions of arbitrary precision.

When generating random `Rational` variables, numerators and denominators in the range $[-10, 10]$ are used, however this can be modified by adjusting the values of `Rational._min_random` and `Rational._max_random` as required.

The `.reduce()` method on `Rational` objects multiplies the set of numbers by the lowest common multiple of their denominators, ensuring that the resulting set contains integers with no common factor.

The `.sqrt()` method uses an algorithm based on Newton's method[2]. Testing for squareness in `.is_square()` is done by taking the square root and checking it squares back to the original number. This will always work, since the integer square root algorithm used always returns the floor of the square root (e.g. $isqrt(x) = \lfloor\sqrt{x}\rfloor$). As such only a square number will satisfy `x == x.sqrt()*x.sqrt()`.

4.2.3 `FiniteField`

The fields $\mathbb{Z}_p$, which are the integers modulo p for prime p, are supported in `pygeom` by the `FiniteField` class. Each object of type `FiniteField` will have a member `._base`, which specifies the value of p. To avoid having to specify this value each time a new variable is created, a class variable `.base` is maintained. This can be set once and all subsequent object instantiations will use the same value. It remains to be seen whether this design choice is sound, as it may end up making it confusing for a user who regularly wishes to switch between different bases.

The `.reduce()` method multiplies each number in the set by the inverse of the first number, ensuring that the first number is always 1.

The `.is_square()` method uses the fact that if x is a square number then $x^{(p-1)/2} \equiv 1 \pmod{p}$. The current implementation uses an $O(p)$ algorithm to do this calculation, however this could be optimized to $O(ln(p))$ using modular

[2]See http://en.wikipedia.org/wiki/Integer_square_root

exponentiation. The `.sqrt()` method uses an implementation of the Shanks-Tonelli algorithm[3]. If one assumes the Generalised Riemann Hypothesis then this algorithm can be shown to run in polynomial time with $O(ln^4 p)$ [8]. These algorithms were chosen for their ease of implementation over pure speed, and may be an area for optimization for future versions of `pygeom` .

4.2.4 Field of Algebraic Expressions

If we consider the set of algebraic expressions consisting of zero or more variables we can see that it constitutes a field. In theory we could thus implement a class to represent this field and perform calculations symbolically. Such a class would in fact let us perform symbolic calculations, allowing us to quickly perform the algebra needed to calculate general geometric theorems. Indeed, a prototype class was initially implemented with this goal in mind, however it quickly became clear that the amount of computation involved in reducing algebraic expressions to their simplest terms was impractical.

A future development goal for the `pygeom` package should be to integrate it with a symbolic computing package, such as Sympy[4], to allow such calculations to be performed quickly and efficiently.

4.2.5 Real Numbers

A canonical field in mathematics is $\mathbb{R}$, the set of real numbers. Real numbers are generally represented as floating point numbers in software. The floating point numbers can only represent a finite number of different values and as such are only an approximation to the real numbers. In particular, they do not obey the field axioms. The following example demonstrates this.

```
In [1]: 1e100 + 1e-100 == 1e100
Out[1]: True
```

In this case we have $a + b = a$, where $b \neq 0$, which violates the field axioms.

Consideration of any other representation scheme for the real numbers will also quickly run into problems and as such `pygeom` is not able to provide a class to represent the real numbers. Users who wish to approximate the real numbers are advised to use the `Rational` class.

[3]See http://planetmath.org/encyclopedia/ShanksTonelliAlgorithm.html
[4]http://code.google.com/p/sympy/

4.2.6 Examples

The following interactive Python session demonstrates some of ways in which pygeom field variables can be used.

```
>>> from pygeom.field import Rational
>>> # Create two varibles.
>>> x = Rational(1, 2)
>>> y = Rational(5, 3)
>>>
>>> # Print their values.
>>> print x, y
1/2 5/3
>>>
>>> # Field operations.
>>> x + y
13/6
>>> x*y
5/6
>>> x/y
3/10
>>> x - y
-7/6
>>> x + 10
21/2
>>>
>>> # We expect x*x to be a square number, but not
>>> # x on its own.
>>> (x*x).is_square()
True
>>> x.is_square()
False
>>>
>>> # We expect the square root of x*x to be x.
>>> (x*x).sqrt()
1/2
>>> (x*x).sqrt() == x
True
>>>
>>> # Multiply x and y by their LCM
>>> x.reduce([y])
```

```
[3/1, 10/1]
>>>
>>> # Generate some random numbers.
>>> Rational.random()
-10/7
>>> Rational.random()
-6/1
>>>
```

4.3 Geometries

As well as operating over a particular field, any calculations in planar universal geometry must also be done in the context of a particular geometry, represented by a quadratic form. The `geometry.py` module provide support for creating and working with such objects.

4.3.1 `Geometry`

The geometries of planar universal geometry are represented with objects of the `Geometry` class, whose interface is given in Figure 4.2. This class has a single data member, `.form`, which stores the quadratic form of the geometry. The form is stored as a 3-tuple `(a, b, c)`, representing the quadratic form $\begin{pmatrix} a & b \\ b & c \end{pmatrix}$.

The class has three methods. The `.det()` method calculates the determinant of the quadratic form, i.e. `G.det()` corresponds to Δ_G. The `.norm()` method calculates the norm of a `Point` object (see section 4.4.1), i.e if `X0` is a `Point` representing X_0 then `G.norm(X0)` corresponds to $\|\vec{X_0}\|_G$. The metric dot product of two `Point`s can be found with the `dot()` method, i.e. `G.dot(X0, X1)` corresponds to $\vec{X_0} \cdot_G \vec{X_1}$.

4.3.2 Chromogeometry

The `geometry.py` module also provides functions to create the three geometries of chromogeometry in a given field. The functions `blue(field)`, `red(field)` and `green(field)` will return `Geometry` objects corresponding to the respective geometries over the given field.

```
class Geometry(object):

    def __init__(self, a, b, c):

    def dot(self, point1, point2):

    def det(self):

    def norm(self, point):

    def __repr__(self):

    def __eq__(self, other):

    def __ne__(self, other):
```

Figure 4.2: The `Geometry` class interface.

4.4 Core Objects

There are three types of geometrical objects which we encounter in planar universal geometry; points, lines and conics. In `pygeom` these are represented in the classes `Point`, `Line` and `Conic` respectively, all of which are defined in `core.py`.

Core objects may optionally be associated with a particular geometry. Indeed, certain methods of the core object require a geometry to be specified. The methods which require a geometry to be specified are decorated with the `@check_geometry` decorator, which raises `GeometryError` if a geometry has not been specified.

The core classes have two common methods. The `.form()` method returns a tuple of numbers (`Field` objects) which are the canonical form of the object. The `.eval()` method check whether an (x, y) tuple “corresponds” to the object. For points this means that tuple is equal to the form of the point. For lines and conics, this means that the point $[x, y]$ lies on the line or conic.

4.4.1 Point

While the notions of a point and a vector are subtly distinct in an abstract sense, the two are sufficiently similar that `pygeom` represents them both using a single class. The `Point` class (Figure 4.3), when thought of as representing vectors, supports the basic vector space operations of addition and scalar multiplication. When the `Point` is associated with a particular geometry, the vector norm can also be taken using the `.norm()` method. The boolean method `.null()` determines whether the point is a null point in its geometry.

```
class Point(Core):

    def __init__(self, x, y, geometry=None):

    def form(self):

    def eval(self, x, y):

    def __repr__(self):

    def __sub__(self, other):

    def __mul__(self, other):

    def __rmul__(self, other):

    def __add__(self, other):

    def __div__(self, other):

    @check_geometry
    def null(self):

    @check_geometry
    def norm(self):

    def circle(self, quadrance):
```

Figure 4.3: The `Point` class interface.

Given a single point, the only geometric object we can construct is a circle. The `.circle()` method takes a quadrance and returns a `Conic` representing a circle centred at the point and having the given quadrance.

4.4.2 Line

The `Line` class (Figure 4.4) represents the line $\langle a : b : c \rangle$ by storing the tuple `(a, b, c)` as the form. Other than the core methods, this class only provides two new methods. The `.vector()` method returns a `Point` object, representing the vector representation of the line, e.g. the point $[-b, a]$. The boolean method `.null()` checks whether the line is a null line in its geometry.

```
class Line(Core):
    def __init__(self, a, b, c, geometry=None):

    def form(self):

    def eval(self, x, y):

    def __repr__(self):

    def vector(self):

    @check_geometry
    def null(self):
```

Figure 4.4: The `Line` class interface.

4.4.3 Conic

The `Conic` class represents the conic $\langle A : B : C : D : E : F \rangle$ by storing a tuple `(a, b, c, d, e, f)` as the form.

The boolean `.through()` method will determine whether the conic passes through the given point. The protected method `._point_on()` will create a point which does lie on the conic, so `conic.through(conic._point_on()) == True`. This method is protected as it is only expected to be used by the test suite (see chapter 5) and is not part of the public API.

If we take a conic object as representing the conic $\langle P : \vec{q} : r \rangle$ then the method `.det()` will return the value of the determinant Δ_P.

```
class Conic(Core):
    def __init__(self, a, b, c, d, e, f, geometry=None):

    def form(self):

    def __repr__(self):

    def eval(self, x, y):

    def through(self, point):

    def det(self):

    def _point_on(self):

    def is_parabola(self):

    @check_geometry
    def is_circle(self):

    @check_geometry
    def centre_quadrance(self):

    @check_geometry
    def focus_directrix(self):

    def co_diagonal(self, line):

    def tangent(self, point):

    def is_tangent(self, line):

    def pole(self, polar):

    def polar(self, pole):
```

Figure 4.5: The `Conic` class interface.

The method `.tangent()` will create the tangent line passing through the point given as a parameter. This method requires the point to lie on the conic. The method boolean `.is_tangent()` will check whether the given line is a tangent of the conic.

The method `.focus_directrix()` can be used to find both focus/directrix pairs, as well as the associated constant, of a conic which is not a circle. This method returns a tuple `(focus_direc_1, focus_direc_2, K)` where the first two elements are `PointLine` objects and the last is a field value.

If a conic is a grammola then given one diagonal, we can find its co-diagonal as well as the grammola constant. The `.co_diagonal()` method will return a `(Line, Field)` tuple representing these values, given a valid diagonal line.

If a conic is a circle then the method `.centre_quadrance()` will return a `(Point, Field)` tuple representing the centre and the quadrance of the circle.

The pair of methods `.is_parabola()` and `.is_circle()` are boolean functions to checks whether the conic is a parabola or a circle respectively. The methods `.pole()` and `.polar()` allow poles and polars to be calculated. They are inverse functions of each other so `conic.polar(conic.pole(line)) == line`.

4.5 Paired Objects

While the core objects are the fundamental building blocks of the `pygeom` library, they have limited power when taken as individual entities. However when we pair the objects together we open up a world of possible constructions. The `pairs.py` module provides three classes representing pairs of `Lines` and `Points`. Working with these classes we can construct many interesting geometrical objects.

4.5.1 `LineSegment`

If we take two distinct points X_0 and X_1 they represent a segment on a line. The `LineSegment` class (Figure 4.6) is constructed from two `Point` objects and can be used to construct objects with respect to these points and the line passing through them.

The quadrance between the two points can be found with the `.quadrance()` method. The midpoint between the two points, being the point on the line which is equiquadrance from the two points, can be found using the `.midpoint()` method. Likewise, the `.perp_bisector()` will construct the line passing through the midpoint, perpendicular to the line through the two points.

```
class LineSegment(object):

    def __init__(self, point1, point2):

    def __eq__(self, other):

    @check_geometry
    def midpoint(self):

    @check_geometry
    def perp_bisector(self):

    @check_geometry
    def quadrance(self):

    @check_geometry
    def quadrola(self, K):
```

Figure 4.6: The `LineSegment` class interface.

The two points of a `LineSegment` can also be used to form a quadrola. If X_0 and X_1 are the two points represented by the `LineSegment` and K is a number then the `.quadrola()` method will construct a `Conic` representing the quadrola satisfying $A(Q(X, X_0), Q(X, X_1), K) = 0$.

4.5.2 `Vertex`

A pair of non-parallel lines taken together meet at a vertex. This motivates the `pygeom` class `Vertex` (Figure 4.7), which represents a pair of lines, although we don't restrict ourselves to non-parallel lines. The point of intersection of the lines can be accessed through the `.point` class member. The boolean method `.parallel()` allows the user to check whether the lines are parallel or not.

One of the fundamental ideas in universal geometry is the measure of the spread between two lines. The `.spread()` method will calculate the spread of the lines in a vertex. The boolean `.perpendicular()` method uses the spread to determine whether the two lines are perpendicular in their geometry.

For any non-parallel pair of lines, there will be two bisectors. These two bisectors themselves form a new vertex and can be constructed with the `.bisect()` method.

```
class Vertex(object):

    def __init__(self, line1, line2):

    def __eq__(self, other):

    def parallel(self):

    @check_geometry
    def spread(self):

    @check_geometry
    def perpendicular(self):

    @check_geometry
    def bisect(self):

    @check_geometry
    def grammola(self, K):
```

Figure 4.7: The `Vertex` class interface.

The two lines of a vertex can be used as the diagonals in the construction of a grammola, using the `.grammola()` method, which returns a new `Conic` object.

4.5.3 `PointLine`

The most versatile combination of core objects is to be found when a point and a line are combined. The `PointLine` class (Figure 4.8) represents this pairing and provides a number of methods to make new constructions. It also provides the boolean method `.on()` to check whether the point lies on the line.

The simplest construction is that of a new line which is parallel to the line and passes through the point. The `.parallel()` method constructs this line and should not be confused with the `.parallel()` method of the `Vertex` class.

The `.altitude()` method returns a new `PointLine` representing the altitude, and its foot, of the point. A new `PointLine` can be constructed which is a reflection of the point through the line. This construction is performed by the

```
class PointLine(object):

    def __init__(self, point, line):

    def __eq__(self, other):

    def on(self):

    @check_geometry
    def reflection(self):

    @check_geometry
    def altitude(self):

    def parallel(self):

    @check_geometry
    def quadrance(self):

    @check_geometry
    def construct_quadrance(self, quadrance):

    @check_geometry
    def construct_spread(self, spread):

    @check_geometry
    def parabola(self):

    @check_geometry
    def conic(self, K):
```

Figure 4.8: The `PointLine` class interface.

`.reflection()` method.

If one tries to construct a new line which passes through the point and forms a given spread with the original line they will find two solutions. These two solutions can be constructed in the form of a `Vertex` using the `.construct_spread()` method. Likewise, there are, in general, two new points which lie on the line and are a given quadrance from the original point. A `LineSegment` can be constructed to represent these two solutions using the `.construct_quadrance()` method.

If we take the point and line as a focus and directrix pairing, we can construct a conic with constant K with the `.conic()` method. Likewise, a parabola can be constructed with the `.parabola()` method. This is equivalent to calling `.conic()` with a value of 1.

4.6 Examples

To give a demonstration of how the library can be used, we will use it to solve the following problem:

> Calculate the following in the field $\mathbb{Z}_{13}$ working with the red geometry. Given the points $A = [3, 7]$, $B = [4, 12]$, $C = [9, 2]$, verify that the altitudes of the triangle meet at a single point (the orthocentre). Find the circumcentre of the triangle and verify that it is equiquadrance from each of the vertices of the triangle

To solve this, we put the following code into a file called `example.py`.

```
from pygeom.field import FiniteField
from pygeom.geometry import red
from pygeom.core import Point
from pygeom.pairs import LineSegment, PointLine, Vertex

# Set up the field
f = FiniteField
f.base = 13

geometry = red(f)

# Create the points
A = Point(f(3), f(7), geometry)
B = Point(f(4), f(12), geometry)
C = Point(f(9), f(2), geometry)

# Create the lines of the triangle
AB = LineSegment(A, B)
AC = LineSegment(A, C)
BC = LineSegment(B, C)
```

```
# Create the altitudes
alt_a = PointLine(A, BC.line).altitude().line
alt_b = PointLine(B, AC.line).altitude().line
alt_c = PointLine(C, AB.line).altitude().line

# Calculate the points of intersection of the altitudes
O_ab = Vertex(alt_a, alt_b).point
O_bc = Vertex(alt_b, alt_c).point
O_ca = Vertex(alt_c, alt_a).point

# Check that the points are indeed equal
assert O_ab == O_bc == O_ca
print ‘‘Orthocentre:’’, O_ab

# Find the circumcentre
C_a = BC.perp_bisector()
C_b = AC.perp_bisector()
C_0 = Vertex(C_a, C_b).point
print ‘‘Circumcentre:’’, C_0

# Check the quadrances
Q_a = LineSegment(A, C_0).quadrance()
Q_b = LineSegment(B, C_0).quadrance()
Q_c = LineSegment(C, C_0).quadrance()
assert Q_a == Q_b == Q_c
```

When run, it gives the following result.

```
$ python example.py
Orthocentre: [10 (13), 10 (13)]
Circumcentre: [3 (13), 12 (13)]
Circumquadrance: 1 (13)
```

We have found the orthocentre, circumcentre and circumquadrance as required. The `assert` statements have also verified that the perpendicular bisectors meet at a single point and that the circumcentre is equiquadrance from the three vertices.

This example shows that `pygeom` can easily be used to perform calculations in planar universal geometry when attempting to solve specific problems.

Chapter 5

Testing

For a mathematical software library to be of practical value, the user must trust it to provide correct results. Furthermore it must handle all possible inputs the user might give it, either in producing a correct result or else informing the user that their input is invalid. To provide this level of assurance it is not sufficient for the library developer to simply claim that their software meets these standards, they must be able to demonstrate it. Furthermore, the user must be able to reproduce this demonstration when the software is installed on their own systems.

Such demonstrations generally come in the form of a *test suite*, which is a piece of software designed to utilise the library in a systematic way, verifying the results produced. The `pygeom` library provides such a test suite, which is described below. This test suite aims to provide a level of assurance to the users of the library, however it should be noted that the suite is necessarily incomplete, in that it does not test *every* possibly combination of inputs. As such there may be bugs in the library which are not detected by the test suite. As the old adage goes, testing can only prove the presence of bugs, not their absence.

Mathematical software libraries such as `pygeom` have the nice property that their inputs and outputs represent mathematical objects. By taking advantage of the mathematical properties of these objects, it can generally be confirmed that the software is performing as expected. For example, if we were to construct the foot of an altitude to a line, a test could confirm that the foot did indeed lie on the original line.

In this chapter we outline some of the techniques used to test `pygeom` with the aim of assuring any user that it will perform as expected. The users of the library are encouraged to inspect and use these tests to independently verify the correctness of the library.

5.1 Unit Testing

Unit testing[1] is a method of testing individual components of a software system. The software system is broken down into components, or *units*, and tests are written which exercise these units independently from each other. A unit might be an individual module, class or function. Each *unit test* will execute a particular unit and then check to make sure the results are as expected. Each unit may have multiple unit tests, each checking different aspects of the result. Furthermore, each unit test may be run multiple times, using different inputs but checking the same output condition each time.

A benefit of unit testing is that when errors are detected, they can generally be tracked down easily, since small pieces of code are being run in a relatively independent manner. This reduces the number of places the developer must search to find the underlying cause of an error, which leads to faster debugging. A disadvantage of unit testing is that it will not detect errors which arise as a result of interaction between different modules. This can be particularly problematic in very large, tightly coupled software systems, such as a word processor or operating system. In the case of `pygeom`, which is a simple, loosely coupled system, this is not so much of a concern.

5.2 Fuzz Testing

Another form of testing which is used by `pygeom` is *fuzz testing*. Fuzz testing involves calling the library with random input and ensuring that it handles the data correctly. This allows a wide range of inputs to be tested while also generating input conditions which might have been difficult to manually construct or else might have been overlooked by the developer.

A drawback of fuzz testing, compared to manually constructed examples, is that it cannot use explicit expected output results and must rely on mathematical identities and other invariants. For example, if we wanted to test an `add()` function, which simply added two variables, we could construct an explicit example to test it.

[1] http://en.wikipedia.org/wiki/Unit_testing

```
x = 5
y = 10
assert add(x, y) == 15
```

Using fuzz testing, we can only test known properties such as the commutative property, e.g.

```
a = random()
b = random()
assert add(a, b) == add(b, a)
```

In a mathematical library such as `pygeom` we have many such identities available to use and therefore we can effectively make use of fuzz testing.

5.3 Coverage Testing

When testing a software library it is important to know that the entire library has been tested (or to at least know which parts have *not* been tested!). If a particular piece of the library, be it an entire function or a single line, has not been tested, then the developer cannot be sure that it will work as expected. *Coverage testing* is a technique which keeps track of each line of code as it is executed by the test suite and then provides a report at the end showing which lines of code were executed and which were not.

Coverage testing provides a useful metric of the extensiveness of a test suite, however the results must be used with caution. Simply executing every line of code once is by no means sufficient to claim that the library is fully tested, as each line must be able to deal with many different input conditions. As such, complete coverage of the library should be considered a necessary but not sufficient condition for a test suite to be considered complete.

5.4 `pygeom` Test Framework

The `pygeom` library has a test suite which uses a combination of the above techniques. This test suite is built on top of the Nose test framework, which is "a unittest-based testing framework for python that makes writing and running tests easier"[2]. Running the test suite is done using the `nosetest` program and produces the following results:

[2] http://code.google.com/p/python-nose/

```
$ nosetests
................................................
----------------------------------------------------------------------
Ran 48 tests in 124.761s

OK
```

If we edit the library to create an error, such as changing the `Conic.centre_quadrance()` method to return `K+1` instead of `K`, nosetest will indicate the error as follows.

```
$ nosetests
....F...........................................
======================================================================
FAIL: test_conic.test_fuzz_circle
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Library/Python/2.5/site-packages/nose-0.10.3-py2.5.egg/nose/case.py", line 182, in runTest
    self.test(*self.arg)
  File "/Users/timleslie/src/pygeom/tests/test_conic.py", line 116, in test_fuzz_circle
    assert new_K == K
AssertionError

----------------------------------------------------------------------
Ran 48 tests in 121.957s

FAILED (failures=1)
```

By examining the code in the failing test (`test_fuzz_circle` in `test_conic.py`) we see that the return value of `centre_quadrance()` was not equal to the expected value. This corresponds nicely with the nature of the artificial bug and shows how this test framework can help in detecting and tracking down errors.

5.4.1 Coverage

The Nose test framework supports coverage testing as an additional command line option. When run with coverage testing enabled, `nosetest` gives the following results for the `pygeom` library.

```
$ nosetests --with-coverage --cover-erase --cover-package=pygeom
................................................
Name              Stmts   Exec  Cover   Missing
-----------------------------------------------
pygeom                0      0   100%
pygeom.core         180    178    98%   27, 33
pygeom.field        209    209   100%
pygeom.geometry      28     28   100%
pygeom.pairs        198    198   100%
pygeom.util          96     96   100%
----------------------------------------------
TOTAL               711    709    99%
----------------------------------------------------------------------
Ran 48 tests in 853.834s

OK
```

The report shows, for each module in the package, how many executable statements were found (`Stmts`), the number of these statements which were actually executed (`Exec`), the coverage as a percentage (`Cover`), and the line numbers of those lines which were not executed (`Missing`).

The two lines which do not get executed by the test suite are the bodies of abstract methods in the `Core` class and thus cannot be executed by design. The results thus tell us that the `pygeom` test suite has effectively complete coverage. This ensures that the entire library is free from trivial bugs, e.g. lines of code which would not work correctly for *any* input. As mentioned above however, we must consider the tests themselves to gauge the completeness of the test suite.

5.5 `pygeom` Test Suite

The `pygeom` test suite consists of a number of Python modules, each of which contain a set of test functions. Each test function constitutes a single unit test. These functions will generally create some test data, perform some operations on it using the `pygeom` library and then verify that the result is as expected. Each of these test modules is described below.

5.5.1 `test_field.py`

This module contains a series of tests for the `Field`, `FiniteField` and `Rational` classes. Each of the field operations is tested on concrete examples with known results to ensure that the most trivial calculations give correct numerical results. Fuzz testing is then used to test the twelve field axioms[3] on randomly

[3]http://mathworld.wolfram.com/FieldAxioms.html

generated values. This ensures that the implementation does indeed implement the mathematical objects it claims to.

5.5.2 `test_rational.py`

The `test_rational.py` module adds some extra tests which are specific to the `Rational` class. These tests ensure that initialisation of rational values behaves correctly over a wide range of cases and also that the internal representation correctly removes common any common factors.

5.5.3 `test_core.py`

The `Point` and `Line` classes are tested by the `test_core.py` module. A range of initialisation tests are performed to ensure that creating objects behaves as expected. As well as this, fuzz testing is performed on random points to ensure that `Point` objects behave properly as vectors. This requires making sure that vector addition and scalar multiplication work as expected.

5.5.4 `test_conic.py`

The `Conic` class is tested separately from the other core objects, as it has a significantly larger API to cover. The module `test_conic.py` uses fuzz testing to generate points and lines which in turn are used to create general conics, parabola, circle, quadrolas and grammolas. The generated conics are then tested to ensure that they satisfy the respective properties which have been derived in Chapter 3.

5.5.5 `test_pairs.py`

The `test_pairs.py` module works in much the same way as the `test_conics.py` module. It uses fuzz testing to generate random points and lines, then combines these into pair objects. These pair objects then have their methods called, and the returned objects are compared with the expected results from Chapter 2.

5.6 Summary

This chapter has outlined the `pygeom` test suite. These tests should allow the user to trust that the library will perform correctly in a wide range of situations. Furthermore, they will serve as useful tool in any future developments of the library, as the developers will be able to quickly verify that any changes they make do not break existing functionality.

Chapter 6

Conclusion

The theories of planar universal geometry provide a bridge between the more general theories of universal geometry and the specific geometries of chromogeometry. In this thesis we have developed a number of new theories related to lines and conics in planar universal geometry, as well as recasting some old theories within this new framework. It is hoped that these will provide a foundation for further investigations in this field.

We have also presented the `pygeom` library, which allows numerical calculations to be performed in planar universal geometry. By virtue of the fact that chromogeometry and rational trigonometry arise as special cases of planar universal geometry, `pygeom` is able to do calculations within these regimes.

Bibliography

[1] N. J. Wildberger. *Divine Proportions: Rational Trigonometry to Universal Geometry*. Wild Egg, 2005.

[2] N. J. Wildberger. A rational approach to trigonometry. *Math Horizans*, November 2008.

[3] N. J. Wildberger. The ancient greeks present: Rational trigonometry, 2008, arXiv:0806.3481.

[4] N. J. Wildberger. Affine and projective universal geometry, 2006, arXiv:math/0612499.

[5] N. J. Wildberger. One dimensional metrical geometry. *Geometriae Dedicata*, 128:145–166.

[6] N. J. Wildberger. Chromogeometry. 2008, arXiv:0806.3617.

[7] N. J. Wildberger. Chromogeometry and relativistic conics, 2008, arXiv:0806.2789.

[8] Henri Cohen. *A course in computational algebraic number theory*. Springer, 1993.