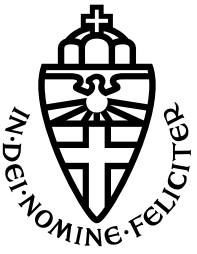
Radboud University Nijmegen



Faculty of Science

Elliptic curves: various models and their addition laws

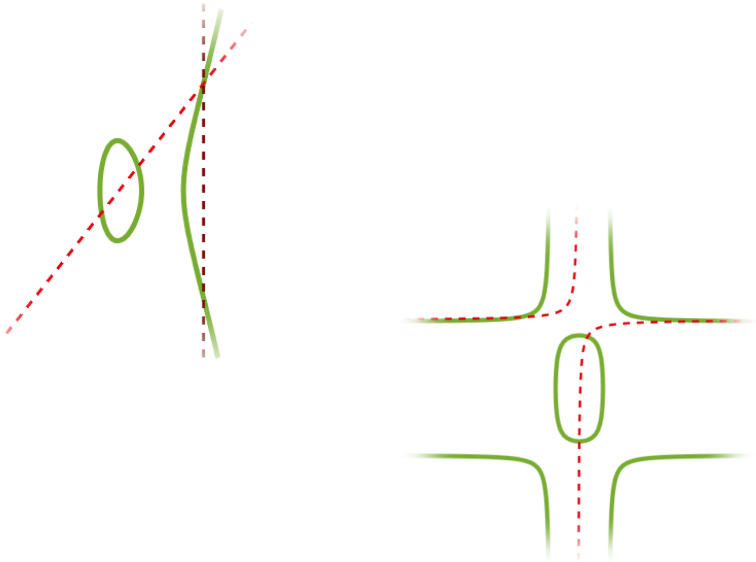
- a study in computer algebra -

Thesis MSc Mathematics

Author:
Marta Parada Seguí
Student nr.: 4599799

Supervisor:
dr. Wieb Bosma
IMAPP

Second reader:
dr. Monika Trimoska
ICIS



July 2022

Abstract

The study of elliptic curves and their addition laws is performed in different directions nowadays due to how useful they are across a wide variety of fields. This has led the concept of *elliptic curve* to take the form of various models. Throughout this thesis we will look closely into three of these models: the Weierstrass form, the Edwards model and the twisted Edwards model. Our goal is to understand how these models work and, in order to do so, we study their respective addition laws, aiming to find complete addition law systems of minimal cardinality for each model. We also learn how are the models connected to each other and how their addition laws behave through these relations. Aside from the bibliographical research needed to study these matters, we implement our findings in Magma and try to obtain our own results using computer algebra as much as possible. Lastly we offer an overview of other models for which similar studies could be done. Collecting all this information we hope to shine some light on how is it that these various models represent the same curves.

Acknowledgements

First and foremost I would like to thank my supervisor Wieb Bosma for his patience, wise guidance and inspiring curiosity. I have very much enjoyed our meetings where we explored the world of elliptic curves through computer algebra. I am also grateful to Monika Trimoska for jumping in as the second reader and to my student advisor Ina de Vries for her always friendly and very helpful attitude. And thank you Javi as well for that very enlightening summer talk that helped me sort out some important concepts in this topic.

I would also like to thank all the wonderful people that I have met during the time I have been working on this thesis, and that today I can proudly call my friends: you have all helped me with your kindness and support, making me feel like I also have a family in this foreign country. I highly doubt I would've managed to finish without you.

Lastly, of course, special thanks to my family: in spite of the distance, you have been very much by my side. I am grateful to my mom for her wise pieces of advice and for always keeping me grounded. I admire my dad for his never-ending enthusiasm while hearing me talk about so many things that are so far away from his realm of understanding. And last, but definitely not least, I will be forever grateful to have such a sister, someone that I can count on for the most random things and who is a constant source of support and inspiration.

Thank you all.

Contents

Introduction

Elliptic curves are a family of curves studied and used in several fields like number theory and cryptography. Their applications vary from key exchange algorithms [[15], §8.1.3] to pseudorandom number generators [[18],[35]]. And they are also important in integer factorization and primality testing algorithms [Ch. 7, [15]].

Apart from being the main focus of a lot of current studies, elliptic curves do have quite some history and have shown up many times even before the concept of *elliptic curve* was established. E. Brown and A. Rice did a great job collecting the history of how the study of elliptic curves came to be in [36] and we find there is no better way of introducing the concept of elliptic curve than by providing a summary of said collection.

It was as early as the 3rd or 4th century CE when Diophantus of Alexandria included in his book *Arithmetica* a problem that read "To divide a given number into two numbers such that their product is a cube minus its side", which translates to the equation $\bar{y}(a-\bar{y}) = \bar{x}^3 - \bar{x}$ where a is the given number. This was already an elliptic curve in disguise, namely $y^2 = x^3 - x + a^2/4$, applying the substitution $y = \bar{y} - a/2$, $x = -\bar{x}$.

Some centuries later, in 1225 Fibonacci published his book *Liber quadratorum* where he defined a positive integer n as congruent if $u^2 - n$, u^2 and $u^2 + n$ are all nonzero squares for some rational number u. But if n is congruent then it follows that there exists a nonzero y such that $y^2 = u^2(u^2-n)(u^2+n) = x^3 - n^2x$, for $x = u^2$. So we find another hidden family of elliptic curves that Fibonacci unknowingly started studying.

Even Fermat in the 17th century came close to the study of elliptic curves while posing his conjectures about the only integers satisfying $y^2 = x^3 - 2$ being $(3, \pm 5)$ and the only ones for $y^2 = x^3 - 4$ being $(2, \pm 2)$ and $(5, \pm 11)$. Also, in the second half of that century, Newton worked on classifying cubic curves of the form $y^2 = ax^3 + bx^2 + cx + d$. He stated the fact that the intersection of said curves with a line consists of three points and, if the line is tangent to the curve, then two of those three points are the same. This would eventually lead to the chord and tangent addition that gives the point on an elliptic curve their group structure.

It all started to get more concrete around the term "elliptic" when Euler came across a differential equation that he was able to solve without the need of finding a separation of variables for it: the rectification of the ellipse. That is, the study of calculating the length of an arc of an ellipse between two given points. He published a paper about it in 1738. A few decades later Legendre would become fascinated by the study of several non-elementary integrals similar to the one Euler solved, called elliptic integrals. Later on, Abel and Jacobi picked up the baton of the study of these integrals but they found the study of their inverses (or elliptic functions) to be more interesting.

Mid-19th century Eisenstein took the approach of starting with the periods of the elliptic functions to define said functions via infinite series. Ultimately, he proved that all elliptic functions of a certain form must satisfy differential equations of the form $[\bar{y}'(z)]^2 = p(\bar{y}(z))$, where p is a cubic polynomial with no repeated roots, thus linking the study of elliptic functions to that of equations involving cubic polynomials. In fact, *a curve of the form $y^2 = p(x)$ where $p(x)$ is a cubic polynomial with no repeated roots* is one of the many definitions for elliptic curves.

Another mathematician that studied elliptic functions was Weierstrass. Many of us have been introduced to the topic of elliptic curves by studying their Weierstrass form. But funny enough, Weierstrass never talked about elliptic curves in his work. He did, however, come up with the $\wp$-function as basis of his theory of elliptic functions. And this function provides a parametrization for a cubic curve such that $(\wp')^2 = 4\wp^3 - g_2\wp - g_3$, where g_2, g_3 are constants. Note that this follows Eisenstein result. And applying

the right substitution of variables, Weierstrass also found an expression for $\wp(u+v)$, thus giving the cubic curve above an addition formula.

Later on, in 1901 Poincaré published his paper *Sur les propriétés arithmétiques des courbes algébriques*, where he collected all these ideas about the study of points on cubic curves and addition formulae and so he set the basis for a new area of study. And since these curves of genus 1 require elliptic functions for their parametrization, they became later known as elliptic curves.

Now we have a better idea how the study of elliptic curves came to be, but what does actually define an elliptic curve? Nowadays in abstract mathematics the formal definition for them is the following:

An elliptic curve is a smooth, projective, algebraic curve of genus one, defined over a field K with a K-rational point $\mathcal{O}$.

We will start our own study of these curves by introducing some basic concepts and our notation for them in the preliminaries. Some notes on the coding parts are also included there. Then we'll move on to Chapter 1 to learn about the first model that was studied in this topic: the Weierstrass form. We will show its affine addition law and the geometric interpretation behind it, as well as a complete system of projective addition laws for it. This will be followed by the computational work we have done in Magma [8] regarding the addition laws on this model. In Chapter 2 we will look into the Edwards model, its affine addition and its relation to the Weierstrass model, showing the explicit maps that link them. And we will see some examples of how these maps work and preserve the addition laws in Magma . For the twisted Edwards model that is studied in Chapter 3 after we also have a complete system of projective addition laws, together with several maps that connect this model to the previous two. And of course the implementations in computer algebra and examples we have worked with regarding all of it.

In Chapter 4 we offer an overview of several other models for elliptic curves, their addition laws and ideas about how do they relate to at least one other model. Lastly, in the appendix we show all the functions we mention throughout the thesis.

Preliminaries

In the following lines we will establish some notations and define several concepts as we will use them in the coming chapters. Let's start by pointing out that, in the spirit of emphasizing the generality of the concept of elliptic curves, we will refer to a given curve always by $\mathcal{C}$. And each curve will be given by a specific model $\mathcal{E}_\alpha$, with different α for each model.

We will always work over a field K and sometimes there will be certain constraints over the characteristic of said field, which we will specify when necessary. In some cases said constraints offer some simplifications on a model, like in the case of the short Weierstrass model, and in some other cases they are necessary to avoid dealing with certain exceptional cases, as for example the Edwards and twisted Edwards models don't define an elliptic curve when $\text{char}(K) = 2$. The notation $\bar{K}$ denotes the algebraic closure of the field K.

Depending of what we are doing, we will work with points (x, y) in the affine plane $A(K^2)$ or in one of these two projective spaces: $\mathbb{P}^2$ and $\mathbb{P}_1 \times \mathbb{P}_1$. The equivalence relation for two points $P = (X : Y : Z)$, $Q \in \mathbb{P}^2$ is the usual $P \sim Q$ iff $Q = (\lambda X : \lambda Y : \lambda Z) = \lambda P$, where $\lambda \in K \backslash \{0\}$. And in the case of $\mathbb{P}_1 \times \mathbb{P}_1$, the points are of the form $((X : Z), (Y : T))$ where $(X : Z)$ and $(Y : T)$ are two points on the projective space $\mathbb{P}_1$ and the equivalence relation there is $P \sim Q$ iff $Q = \lambda P = (\lambda U : \lambda V)$, for $P = (U : V), Q \in \mathbb{P}^1$ and $\lambda \in K \backslash \{0\}$. We will show the embedding map for this projective space and its relation to $\mathbb{P}^2$ in Section 3.1.2.

Every model will have an affine and a projective definition over a field K. The affine one will be given by a polynomial $f_\alpha(x, y)$ and the projective one by the homogeneous version of $f_\alpha(x, y)$, namely the polynomial F_α on three variables X, Y, Z when working on $\mathbb{P}^2$ and on four variables X, Y, Z, T in the case of $\mathbb{P}_1 \times \mathbb{P}_1$.

Definition 1. Let $\mathcal{C}$ be an elliptic curve defined over a field K given in $\mathcal{E}_\alpha$ form, then we define its *K-rational points* to be the set $\mathcal{E}_\alpha(K) = \{(X : Y : Z) \in \mathbb{P}^2(K) \mid F_\alpha(X, Y, Z) = 0\}$, where F_α is the homogeneous projective polynomial that defines the model.

Together with an operation we call addition, the set $\mathcal{E}_\alpha(K)$ is an abelian group. The addition is computed in different ways depending on the model, but the most widely used method is the tangent and chord mentioned in the introduction. The identity of the addition is denoted by $\mathcal{O}$. In most models, $\mathcal{O}$ is the point at infinity $(0 : 1 : 0)$, but we'll see that this is not always the case. Since $\mathcal{E}_\alpha(K)$ is a group, its points have an order as elements of said group:

Definition 2. Let $\mathcal{C}$ be an elliptic curve defined over a field K in $\mathcal{E}_\alpha$ form, then we say that a point $P \in \mathcal{E}_\alpha(K)$ has *order* n if n is the least positive integer such that $nP = P + \overset{n}{\cdots} + P = \mathcal{O}$.

We define an addition law on the model $\mathcal{E}_\alpha$ as follows:

Definition 3. An addition law on a model $\mathcal{E}_\alpha$ defined over K is a tuple $S = \langle s_1, s_2, \ldots, s_k \rangle$ of polynomials over K in $2k$ variables such that for every pair P_1, P_2 of K-rational points on $\mathcal{E}_\alpha$ it holds that either $s_i(P_1, P_2) = 0$ for all i or the point P_3 defined by $s_1(P_1, P_2)$, $s_2(P_1, P_2)$, $\ldots$, $s_k(P_1, P_2)$ satisfies $P_3 = P_1 + P_2$, where $+$ denotes the addition in the abelian group $\mathcal{E}_\alpha(K)$.

In $A(K^2)$ we have that $k = 2$, $S = \langle x_3, y_3 \rangle$ and $p_3 = \big(s_1(x_1, y_1, x_2, y_2), s_2(x_1, y_1, x_2, y_2)\big)$ for $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2) \in \mathcal{E}_\alpha(K)$. If we are working on $\mathbb{P}^2$ then $k = 3$, $S = \langle X_3, Y_3, Z_3 \rangle$ and $P_3 = (X_3(P_1, P_2) : Y_3(P_1, P_2) : Z_3(P_1, P_2))$, whereas on $\mathbb{P}_1 \times \mathbb{P}_1$ we have that $k = 4$, $S = \langle X_3, Y_3, Z_3, T_3 \rangle$ and $P_3 = ((X_3(P_1, P_2) : Z_3(P_1, P_2)), (Y_3(P_1, P_2) : T_3(P_1, P_2)))$.

Definition 4. A pair of points $P_1, P_2 \in \mathcal{E}_\alpha(\bar{K})$ is called *exceptional* for a given addition law $S = \{s_i\}$ if all its polynomials vanish when evaluated in these points, i.e, if $s_i(P_1, P_2) = 0$ for all i.

The exceptional pairs can have different forms. For example, several models have an addition law such that $P_1 = P_2$ is the exceptional case, that is, it can't be used for point doubling. All addition laws have exceptional points over the algebraic closure of the field K, but with some conditions one can have a complete addition law:

Definition 5. A *complete addition law* over K for $\mathcal{E}_\alpha$ is an addition law on $\mathcal{E}_\alpha$ such that no pair of points $P_1, P_2 \in \mathcal{E}_\alpha(K)$ is exceptional for that addition law.

A similar, but slightly different property of an addition law is:

Definition 6. A *unified addition law* for $\mathcal{E}_\alpha$ over K is an addition law on $\mathcal{E}_\alpha$ such that it can be used both for point doubling and for adding two distinct points in $\mathcal{E}_\alpha(K)$.

Note that a unified addition law might not be complete, since it can still have exceptional cases other than point doubling even over K. If one wants to be able to add any pair of L-rational points on an elliptic curve, where L is any algebraic extension field of K, then one needs a complete system of addition laws:

Definition 7. A *complete system of addition laws* for $\mathcal{E}_\alpha$ is a collection of addition laws on $\mathcal{E}_\alpha$ such that no pair of points $P_1, P_2 \in \mathcal{E}_\alpha(\bar{K})$ is exceptional for all addition laws in the collection.

Note that we use the algebraic closure of the field in the above definition, meaning a complete system of addition laws can be used to add any two $\bar{K}$-rational points on the curve.

Last but not least, when talking about the relations between models, the following concepts regarding maps between models will come up:

Definition 8. Let $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ be two models for elliptic curves over a field K and $\big(\mathcal{E}_\alpha(K), +_\alpha\big)$, $\big(\mathcal{E}_\beta(K), +_\beta\big)$ their respective sets of K-rational points with their addition laws. One says that $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ are *isomorphic* iff there exists a group isomorphism from $\mathcal{E}_\alpha(K)$ to $\mathcal{E}_\beta(K)$, i.e., a bijective map $\phi\colon \mathcal{E}_\alpha(K) \to \mathcal{E}_\beta(K)$ such that $\phi(P \ +_\alpha \ Q) = \phi(P) \ +_\beta \ \phi(Q)$ for all $P, Q \in \mathcal{E}_\alpha(K)$.

Note that the existence of a group isomorphism $\phi\colon \mathcal{E}_\alpha(K) \ \to \ \mathcal{E}_\beta(K)$ implies that there exists another map $\phi^{-1}\colon \mathcal{E}_\beta(K) \ \to \ \mathcal{E}_\alpha(K)$, known as its inverse, which is also a group isomorphism.

It will be rarely possible to connect two models using an isomorphism, in which case we can hope for them to be *almost* isomorphic or what is the same, birationally equivalent:

Definition 9. Let $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ be two models for elliptic curves over a field K and $\mathcal{E}_\alpha(K)$, $\mathcal{E}_\beta(K)$ their respective sets of K-rational points. Then a rational map $\psi\colon \mathcal{E}_\alpha(K) \to \mathcal{E}_\beta(K)$ is said to be a *birational isomorphism* or a *birational mapping* iff there exist dense open sets $U_\alpha \subset \mathcal{E}_\alpha(K)$ and $U_\beta \subset \mathcal{E}_\beta(K)$ such that ψ is defined on U_α and the restriction of ψ to U_α, namely $\psi|_{U_\alpha}\colon U_\alpha \to U_\beta$, is an isomorphism.

By *rational map* we mean a map defined by a rational function, i.e., a function that can be expressed as an algebraic fraction where both the numerator and the denominator are polynomials.

Definition 10. Let $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ be two models for elliptic curves over a field K and $\mathcal{E}_\alpha(K)$, $\mathcal{E}_\beta(K)$ their respective sets of K-rational points. We have that $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ are *birationally equivalent* or *birationally isomorphic* iff there exists a birational isomorphism $\psi\colon \mathcal{E}_\alpha(K) \to \mathcal{E}_\beta(K)$.

A way of generalizing a given model for elliptic curves that is studied in several cases is applying a twist to said model. Depending on the degree of the twist there are quadratic, cubic and quartic twists. We will work with the first one:

Definition 11. Let $\mathcal{E}_\alpha$ be a model for elliptic curves over a field K. Then there exists another model $\mathcal{E}'_\alpha$, not necessarily isomorphic to $\mathcal{E}_\alpha$ over K, that is isomorphic to $\mathcal{E}_\alpha$ over a quadratic field extension of K. $\mathcal{E}'_\alpha$ is the *quadratic twist* of $\mathcal{E}_\alpha$.

Another type of map we will work with is the isogeny:

Definition 12. Let $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ be two models for elliptic curves over a field K and $\mathcal{O}_\alpha$, $\mathcal{O}_\beta$ the neutral elements of their respective addition laws. Then an *isogeny* is a homomorphism $\varphi\colon \mathcal{E}_\alpha(K) \to \mathcal{E}_\beta(K)$ with coefficients in K such that $\varphi(\mathcal{O}_\alpha) = \mathcal{O}_\beta$.

To define the degree of an isogeny we need to know what is the function field associated to a model $\mathcal{E}_\alpha$ defined by f_α over K. Firstly, given the polynomial ring $K[x, y]$ and the ideal $\langle f_\alpha \rangle$ generated by f_α, we have that the ring of functions of $\mathcal{E}_\alpha$ is:

$$K[\mathcal{E}_\alpha] = K[x, y]/\langle f_\alpha \rangle$$

And the field of fractions of $K[\mathcal{E}_\alpha]$, denoted as $K(\mathcal{E}_\alpha)$, is the field of (rational) functions of $\mathcal{E}_\alpha$.

With this notation we have that the degree of an isogeny is defined as $\deg \varphi = [K(\mathcal{E}_\alpha) : \varphi^*\big(K(\mathcal{E}_\beta)\big)]$, where $\varphi^* \colon K(\mathcal{E}_\alpha) \to K(\mathcal{E}_\beta)$ is an injection of function fields induced by φ such that $\varphi^*(f) = f \circ \varphi$. An isogeny of degree m is called an *m-isogeny*. Furthermore, the existence of an m-isogeny $\varphi \colon \mathcal{E}_\alpha \to \mathcal{E}_\beta$ does imply the existence of a unique isogeny $\widehat{\varphi} \colon \mathcal{E}_\beta \to \mathcal{E}_\alpha$ such that:

$$\varphi \circ \widehat{\varphi} = [m]_\beta \quad \textit{and} \quad \widehat{\varphi} \circ \varphi = [m]_\alpha$$

And this $\widehat{\varphi}$ is called the *dual isogeny* of φ. Here $[m]_\alpha$ and $[m]_\beta$ denote the *multiplication by m* maps for each model, i.e.:

$$\text{given } P \in \mathcal{E}_\alpha(K),\ [m]_\alpha P = mP = P +_\alpha \overset{m}{\cdots} +_\alpha P$$
$$\text{given } Q \in \mathcal{E}_\beta(K),\ [m]_\beta Q = mQ = Q +_\beta \overset{m}{\cdots} +_\beta Q$$

For more information about isogenies, one can refer to [[37], § 4], [[37], § 6] and [[13], § 13.1].

Notes on the Magma scripts:

In the coming chapters, whenever we talk about the computational work we have done, we will include some lines of code to illustrate some specifics. When we display some parts of code that include an output we will use "`>`" to indicate the input we give to Magma and the lines without this symbol will be the output. As we can see for example here:

```
> printf "Hello world. \nWelcome to my master thesis!";

Hello world.
Welcome to my master thesis!
```

Also every "homemade" function we will mention is in the appendix at the end. But for the complete scripts with all the examples and explanatory outputs we have decided to have an online repository for it: `MPS_TFMscripts`. All "`.m`" files are ready to be run in Magma and, in most cases, their computational time also allows to run them in the online Magma calculator. For the script where the computational time is too long the the output is available online.

For those interested in running some of the scripts in the online calculator, be aware that you will most likely need to fetch some functions from the `functions.m` file. So at the beginning of each file substitute these lines with the functions they mention:

```
load "../../functions.m";
//load Ead_AddL, Ead_AddL_evals, ProjCoord, AffCoord
```

Chapter 1

Weierstrass model

The first model we will be looking into is the Weierstrass one. It owes its name to the famous 18th century analyst, who came up with the $\wp$-function as basis of his theory of elliptic functions. And this function provides a parametrization for a cubic curve in the following way, where g_2, g_3 are constants:

$$(\wp')^2 = 4\wp^3 - g_2\wp - g_3$$

And, applying the right substitution of variables, Weierstrass also found an expression for $\wp(u+v)$, thus giving the cubic curve above an addition formula:

$$\wp(u+v) = \frac{1}{4}\left(\frac{\wp'(u) - \wp'(v)}{\wp(u) - \wp(v)}\right)^2 - \wp(u) - \wp(v)$$

Naturally with time the notation for these and the way of studying cubic curves in general changed quite a bit. Nowadays we define the Weierstrass equation in full generality to be:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Here the a_i are elements of some field K and the coefficients of the equation. There is the convention of numbering them in this way, where a_5 is missing. This is due to the fact that the coefficients are actually numbered following the degrees of the Z coordinate when the above equation is embedded in $\mathbb{P}^2$ and expressed in Jacobian coordinates. That is, applying the map $(x, y) \to (X/Z^2, Y/Z^3)$, which yields the following:

$$Y^2 + a_1XYZ + a_3YZ^3 = X^3 + a_2X^2Z^2 + a_4XZ^4 + a_6Z^6$$

In spite of following this notation convention, when we refer to the projective Weierstrass equation here it will be with the usual projection map $(x, y) \to (X/Z, Y/Z)$. And so the projective Weierstrass equation we will work with is:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

In this form, the Weierstrass equation has the point at infinity $\mathcal{O} = (0 : 1 : 0)$. Even when we use the affine equation, we must keep this point in mind. Note that any other projective point $(X : Y : Z)$ that follows this equation will be such that $Z \neq 0$, i.e., in the equivalence class of $(X/Z : Y/Z : 1) = (x : y : 1)$.

Let us remark as well the fact that not all curves defined by the Weierstrass equation are elliptic curves. For a curve defined by a Weierstrass equation to be considered elliptic with origin point $\mathcal{O}$ it needs to be smooth and cubic [[37],ch. III, 3.1]. A handy way to check this is through the value Δ, called the discriminant of the Weierstrass equation and usually defined via some other values:

$$b_2 = a_1^2 + 4a_2; \quad b_4 = 2a_4 + a_1a_3; \quad b_6 = a_3^2 + 4a_6; \quad b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$$

Note that the all the b_i depend on the a_i parameters so Δ is also fully determined by them. And if $\Delta \neq 0$ then the curve defined by the Weierstrass equation is smooth and cubic, therefore:

Definition 13. Let K be a field and $\mathcal{C}$ an elliptic curve over K, then its complete Weierstrass form is $\mathcal{E}_w\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$.

One might wonder if, given an elliptic curve $\mathcal{C}$, to what extent is a Weierstrass equation able to define it uniquely. Silverman gives in [[37], pg. 49] an extensive answer to this, but the bottom line is that it is unique up to isomorphism. And there is an invariant associated with each isomorphism class of curves, the j-invariant, defined as follows:

$$j = (b_2^2 - 24b_4)^3/\Delta$$

The model shown above, and which we will work with the most, is known as the *complete Weierstrass model*. But it is also very common to work with a shortened version of it where $a_1 = a_2 = a_3 = 0$, $a_4 = a$, $a_6 = b$ and $\Delta = 4a^3 + 27b^2$. So in that case the short Weierstrass equation looks like this:

$$y^2 = x^3 + ax + b$$

Note that this equation corresponds to Weierstrass' parametrization shown above, with the following relations: $\wp = x;\ \wp' = 2y;\ a = -g_2/4;\ b = -g_3/4$.

If K is a field such that $\mathrm{char}(K) \neq 2$, then the complete Weierstrass model can be simplified so that the terms xy and y disappear. Furthermore, if $\mathrm{char}(K) \neq 2, 3$, then the complete curve equation of $\mathcal{E}_w$ can be affinely transformed into a short Weierstrass equation using the following maps:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \longrightarrow \bar{y}^2 = \bar{x}^3 + a\bar{x} + b$$

$$a_1, a_2, a_3, a_4, a_6 \rightsquigarrow \ a = -b_2^2/48 + b_4/2,\ \ b = b_2^3/864 - (b_2b_4)/24 + b_6/4$$

$$(x, y) \longmapsto (\bar{x}, \bar{y}) = \left(x + \frac{b_2}{12},\ y + \frac{a_1x + a_3}{2}\right)$$

Now, without further ado, let us present the addition law for the complete Weierstrass model, its geometric interpretation and a complete system of addition laws. Afterwards, we'll show the computational implementations in Magma we have done of this.

1.1 Addition law

The formulae of the traditional addition law for the Weierstrass model are derived from the chord and tangent addition with $\mathcal{O} = (0 : 1 : 0)$ as the identity element. To begin with, we will present said formulae:

Let K be a field, $\mathcal{C}$ an elliptic curve over K and $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ two points on $\mathcal{C}$. Fix a_1, a_2, a_3, a_4, a_6 to be such that $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$. Then the addition law of $\mathcal{C}$ on its Weierstrass form $\mathcal{E}_w$ is as follows:

if $P_2 = \mathcal{O}$, *then:*

$$P_1 + P_2 = P_1 + \mathcal{O} = P_1$$

if $P_2 = -P_1 = (x_1, -y_1 - a_1x_1 - a_3)$, *then:*

$$P_1 + P_2 = \mathcal{O}$$

otherwise:

$$P_1 + P_2 = (x_3, y_3) = \left(\lambda^2 + a_1\lambda - a_2 - x_1 - x_2, -(\lambda + a_1)x_3 - \nu - a_3\right)$$

where:

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}, \quad \nu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}, \quad \text{if } P_1 = P_2$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad \nu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}, \quad \text{if } P_1 \neq \pm P_2$$

As mentioned above, the neutral element for this addition law is the projective point at infinity $\mathcal{O} = (0:1:0)$ and the inverse of an affine point $P = (x, y)$ is $-P = (x, -y - a_1x - a_3)$. It will become clear that $(x_3, y_3) \in \mathcal{E}_w(K)$ after taking a look at the geometric interpretation behind these formulae (see bellow). And that interpretation shows that indeed these formulae define an addition law.

Note that if we look at the addition law for the case $P_1 \neq \pm P_2$ and for a curve given by a short Weierstrass equation then the x-coordinate of the addition law corresponds to the expression for $\wp(u+v)$, where $\wp(u) = x_1$, $\wp(v) = x_2$, $\wp'(u) = 2y_1$ and $\wp'(v) = 2y_2$:

$$\wp(u+v) = \frac{1}{4}\left(\frac{\wp'(u) - \wp'(v)}{\wp(u) - \wp(v)}\right)^2 - \wp(u) - \wp(v) \quad \rightsquigarrow \quad x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2$$

1.1.1 Geometric interpretation

The addition formulae presented above appear when looking at the points in the intersection of an elliptic curve $\mathcal{C}$ and a certain line. In this section we'll take a look at how to define said line, thus providing the geometric interpretation behind the formulae. The method we will describe here defines an addition with $\mathcal{O}$ as neutral element [prop. 2.2, [37]] and it is known as the chord and tangent addition.

Let $\mathcal{C}$ be an elliptic curve given by a projective Weierstrass equation over a field K and let $\mathcal{O} = (0 : 1 : 0)$ be its point at infinity. Since said equation has degree 3, any line will intersect $\mathcal{C}$ in exactly three points over $\bar{K}$ - counting multiplicities, i.e., said points are not necessarily distinct.

Thus if we have two points on the curve $P, Q \in \mathcal{E}_w(K)$ and we define the line $L \subset \mathbb{P}^2$ passing through them, it will intersect $\mathcal{C}$ at a third K-rational point R. Note that in the case $P = Q$ the line L is the tangent to $\mathcal{C}$ in P. Let's consider now the line L' passing through R and $\mathcal{O}$. This line will also have a third intersection point in $\mathcal{C}$ over K, namely $R' = P + Q$.

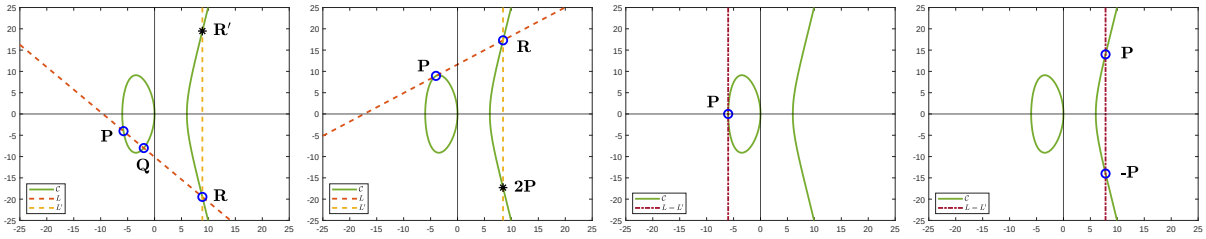
A bit more explicitly, the line L is either $XZ_P - X_PZ = 0$, if $Q = \mathcal{O}$, or $Y = \lambda X + \nu Z$ otherwise. Here λ and ν are the same as defined for the addition law above but in projective coordinates, which look like this:

$$\lambda = \frac{3X_P^2 + 2a_2X_PZ_P + a_4Z_P^2 - a_1Y_PZ_P}{2Y_PZ_P + a_1X_PZ_P + a_3Z_P^2}; \quad \nu = \frac{-X_P^3 + a_4X_PZ_P^2 + 2a_6Z_P^3 - a_3Y_PZ_P^2}{2Y_PZ_P^2 + a_1X_PZ_P^2 + a_3Z_P^3}, \quad \textbf{if } P = Q$$

$$\lambda = \frac{Z_PY_Q - Y_PZ_Q}{X_QZ_P - X_PZ_Q}; \quad \nu = \frac{Y_PX_Q - Y_QX_P}{X_QZ_P - X_PZ_Q}, \quad \textbf{if } P \neq \pm Q$$

And the second line we need will be L': $XZ_R - X_RZ = 0$ in any case. For further details on how to actually derive the explicit addition law formulae, please refer to [[37], Ch. III, §2].

Now let us illustrate how this works with a few examples. In the plots below we work with a curve $\mathcal{C}$ given by a short Weierstrass equation with parameters $a_4 = -36$ and $a_6 = 0$. In Figure (a) we compute the addition of the points $P = (-144/25, -504/125)$ and $Q = (-2, -8)$. Note that the line L' only intersects $\mathcal{C}$ in two points in the plot, since the third point of intersection is the point at infinity $\mathcal{O}$. The plot in Figure (b) shows how point doubling works with $P = (-4, \sqrt{80})$ and here both lines L, tangent in this case, and L' also intersect $\mathcal{C}$ in $\mathcal{O}$ so it doesn't appear in the plot. Figure (c) illustrates also a point doubling, but in this case the point has order 2, so $L = L'$ and the three points of intersection are P, with multiplicity 2, and $\mathcal{O}$. Lastly, in Figure (d) we can see $P \approx (7.811, 14)$ and its opposite $-P \approx (7.811, -14)$, with $L = L'$ having its third point of intersection in $\mathcal{O}$.



(a) $P + Q \approx (8.87, 19.47)$ (b) $2P \approx (8.45, -17.29)$ (c) $2P = \mathcal{O}$ (d) $P - P = \mathcal{O}$

1.1.2 Complete system of addition laws

The addition formulae presented above are definitely not the simplest of formulae to work with, especially due to the several case distinctions. And since they were published there has been of course quite some research done to improve on them. Like the work of W. Bosma and H. W. Lenstra in [9], where they study the minimal cardinality for a complete system of addition laws and how to obtain such a system. In this section we will show some of their results.

To begin with, let's consider an elliptic curve $\mathcal{C}$ over a field K given by the complete Weierstrass equation in projective form:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

Given two positive integers α and β, an *addition law of bidegree* (α, β) on $\mathcal{C}$ consists of an addition law $S = \{p_{X_3}, p_{Y_3}, p_{Z_3}\}$, where the three polynomials $p_{X_3}, p_{Y_3}, p_{Z_3} \in K[X_1, Y_1, Z_1, X_2, Y_2, Z_2]$ are bihomogeneous of bidegree (α, β), i.e., all three polynomials are homogeneous of degree α in the variables X_1, Y_1, Z_1 and homogeneous of degree β in the variables X_2, Y_2, Z_2.

Following this definition, for any given value of the tuple $(a, b, c) \in K \times K \times K$, these polynomials form an addition law of bidegree $(2, 2)$ on $\mathcal{C}$:

$$p_{X_3} = fp_{Z_3}, \qquad p_{Y_3} = gp_{Z_3}, \qquad p_{Z_3} = (as^*(X/Z) + bs^*(Y/Z) + c)Z_0$$

where:

$$f = \lambda^2 + a_1\lambda - \frac{X_1Z_2 + X_2Z_1}{Z_1Z_2} - a_2; \quad g = -(\lambda + a_1)f - \nu - a_3; \quad Z_0 = \frac{(X_1Z_2 - Z_1X_2)^3}{Z_1Z_2}$$

$$s^*(X/Z) = \kappa^2 + a_1\kappa - \frac{X_1Z_2 + X_2Z_1}{Z_1Z_2} - a_2; \quad s^*(Y/Z) = -(\kappa - a_1)s^*(X/Z) - \mu - a_3$$

for:

$$\lambda = \frac{Y_1Z_2 - Y_2Z_1}{X_1Z_2 - X_2Z_1}; \qquad \nu = -\frac{Y_1X_2 - Y_2X_1}{X_1Z_2 - X_2Z_1}$$

$$\kappa = \frac{Y_1Z_2 + Y_2Z_1 + a_1X_2Z_1 + a_3Z_1Z_2}{X_1Z_2 - X_2Z_1}; \qquad \mu = -\frac{Y_1X_2 + Y_2X_1 + a_1X_1X_2 + a_3X_1Z_2}{X_1Z_2 - X_2Z_1}$$

Bosma and Lenstra prove that the smallest complete system of addition laws on $\mathcal{C}$ needs at least 2 addition laws and, if two addition laws form a complete system, then each of them has bidegree (2,2) [Thm. 1, [9]]. They of course also give a complete system of addition laws. In fact, they provide a way to generate complete systems of addition laws for elliptic curves given in Weierstrass form using the family of addition laws shown above and their way of studying the exceptional points of each of said addition laws [Thm. 2, [9]].

Based on the results shown in [[9],§5], we can say for example that the two addition laws given by the tuples $(0, 0, 1)$ and $(0, 1, 0)$ form a complete system of addition laws. And this is what we state next, using the superscript notation to indicate what (a, b, c) tuple defines each polynomial:

Let K be a field, $\mathcal{C}$ an elliptic curve over K and $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ two points on $\mathcal{C}$. Fix a_1, a_2, a_3, a_4, a_6 to be such that $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$. Then the following define a complete set of addition laws on $\mathcal{C}$, when computed to be polynomials of bidegree (2,2):

$$P_1 + P_2 = \begin{cases} P_3 \;=\; (X_3 : Y_3 : Z_3) = \left(p_{X_3}^{(0,0,1)}(P_1, P_2) : p_{Y_3}^{(0,0,1)}(P_1, P_2) : p_{Z_3}^{(0,0,1)}(P_1, P_2)\right) \\ P_3{}' \;=\; (X_3' : Y_3' : Z_3') = \left(p_{X_3}^{(0,1,0)}(P_1, P_2) : p_{Y_3}^{(0,1,0)}(P_1, P_2) : p_{Z_3}^{(0,1,0)}(P_1, P_2)\right) \end{cases} =$$

$$= \begin{cases} \big(fZ_0 : gZ_0 : Z_0\big), & \text{if} \quad P_1 \neq P_2 \\ \big(fs^*(Y/Z)Z_0 : gs^*(Y/Z)Z_0 : s^*(Y/Z)Z_0\big), & \text{if} \quad Y_{P_1-P_2} \neq 0 \end{cases}$$

This system is indeed complete since, if $P_1 = P_2$ then $P_1 - P_2 = P_1 - P_1 = \mathcal{O} = (0 : 1 : 0)$. Thus it will never happen simultaneously that $P_1 = P_2$ and $Y_{P_1-P_2} = 0$, i.e., no pair of points $P_1, P_2 \in \mathcal{E}_w(\bar{K})$ is exceptional for both addition laws in the system.

1.2 Implementations in Magma

The time to see the facts introduced above in action has arrived. In this section some Magma code will be shown to present our implementations about the addition law for the Weierstrass model. We have programmed the chord and tangent addition as well as the complete system of addition laws.

For the geometric interpretation we will show some numerical examples. And for the system of addition laws we will write about how to get addition laws of bidegree $(2, 2)$ and how have we used this to create our own addition function for points on curves in Weierstrass form.

1.2.1 Geometric interpretation

Before we have explained how the chord and tangent method works when adding points on an elliptic curve given by a Weierstrass equation. And here we will lay out a walk-through in Magma of said method following a numerical example.

In the few lines of code below we do the following: as we will have to do for almost all the computations throughout this thesis, we start by defining a few algebraic structures and some variables. The first one we need is a base field `BF`, which in our case will be the finite field $\mathbb{F}_7$, and on top of that we define a `FunctionField` for the a_i coefficients. Having unknown projective variables `X`, `Y` and `Z` will also come in handy so we define them to be elements of a projective space of dimension 2, i.e., $\mathbb{P}^2$. Then we give values to the a_i so they determine the curve we want to work with, which in this case is given by $a_1 = 2$, $a_2 = a_3 = 0$, $a_4 = 4$ and $a_6 = 5$. And then said curve is defined and stored in `C`. We also define a pair of points `P` and `Q` and we can easily check that they are indeed on the curve.

```
> BF := FiniteField(7);
> R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
> R2<X,Y,Z> := ProjectiveSpace(R1,2);

> a1 := 2; a2 := 0; a3 := 0; a4 := 4; a6 := 5;
> C := Curve(R2, (Y^2)*Z + a1*X*Y*Z + a3*Y*Z^2 - X^3-a2*(X^2)*Z-a4*X*Z^2
 - a6*Z^3);
> P := [ 1, 4, 1 ];  Q := [ 3, 2, 1 ];
> C; P in C; Q in C;

Curve over Multivariate rational function field
of rank 5 over GF(7) defined by
6*X^3 + 2*X*Y*Z + Y^2*Z + 3*X*Z^2 + 2*Z^3

true (1 : 4 : 1)
true (3 : 2 : 1)
```

So now we have all the elements we need to add `P` and `Q` using the chord and tangent method. And we start by computing the values `l`, `n` i.e., λ and ν respectively, so we can define the line `L`. And then we look at the points in the intersection of `L` and `C`:

```
> Xp := P[1]; Yp := P[2]; Zp := P[3];
> Xq := Q[1]; Yq := Q[2]; Zq := Q[3];
> l := BF!((Zp*Yq - Yp*Zq)/(Xq*Zp-Xp*Zq));
> n := BF!((Yp*Xq - Yq*Xp)/(Xq*Zp-Xp*Zq));
```

```
> L := Curve(R2,Y - l*X - n*Z);

> Points(Intersection(C,L));

{@ (3 : 2 : 1), (2 : 3 : 1), (1 : 4 : 1) @}
```

Here we see that the third point of intersection is `R = (2 : 3 : 1)`, so the next step is to define the line `Lp` passing through `R` and $\mathcal{O}$ and check the intersection with the curve to find `P+Q`:

```
> R := [2, 3, 1]; Xr := R[1]; Zr := R[3];
> Lp := Curve(R2,X*Zr - Xr*Z);
> Points(Intersection(C,Lp));

{@ (2 : 0 : 1), (2 : 3 : 1), (0 : 1 : 0) @}
```

As we can see, we have that `P + Q = (2 : 0 : 1)`.

Like we showed in 1.1.1, with different λ and ν this method can also be used for some special cases like point doubling. In that case, the procedure looks like this:

```
> l := (3*Xp^2+2*a2*Xp*Zp+a4*Zp^2-a1*Yp*Zp)/(2*Yp*Zp+a1*Xp*Zp+a3*Zp^2);
> n := (-Xp^3+a4*Xp*Zp^2+2*a6*Zp^3-a3*Yp*Zp^2)/(2*Yp*Zp^2+a1*
  Xp*Zp^2+a3*Zp^3);
> L := Curve(R2,Y - l*X - n*Z);
> Points(Intersection(C,L));

{@ (6 : 0 : 1), (1 : 4 : 1) @}

> R := [6, 0, 1]; Xr := R[1]; Zr := R[3];
> Lp := Curve(R2,X*Zr - Xr*Z);
> Points(Intersection(C,Lp));

{@ (6 : 0 : 1), (6 : 2 : 1), (0 : 1 : 0) @}
```

From the second intersection we can see that `2P = (6 : 2 : 1)`. Note as well that in the first intersection there are only two points. That is because, since we are computing the double of `P`, that point actually has multiplicity 2 in this case, as can be seen below:

```
> pts := Points(Intersection(C,L));
> [ <p, IntersectionNumber(C,L,p)> : p in pts];

[ <(6 : 0 : 1), 1>, <(1 : 4 : 1), 2> ]
```

All these computations can be put into a function so it computes the addition of any two points on a given elliptic curve in Weierstrass form. And this is precisely what our function `Ew_AddL_geom` does, so by providing a base field `BF` to work on (must be a finite field or the rationals), a list `lst` with the coefficients of the curve and two points `P`, `Q` we can obtain `P+Q` as computed using this method. This function can be found in the appendix [§A.2, Fn.4] and two more examples are available in the file `WC_AdL_geom.m` from the online repository: one example over the field $\mathbb{Q}$ of characteristic 0 and another over a finite field of large characteristic.

1.2.2 Addition laws of bidegree (2,2)

Explicit formulae for three different addition laws are provided in the last pages of [9] but the X and Y coordinates of the addition law corresponding to $a = 0$, $b = 1$, $c = 0$ are wrong so we need to find a way to generate the formulae ourselves.

Aside from that error, if one were to directly use the system of equations as we have presented it in section 1.1.2 they would observe that said system works for the most part except when it comes to point doubling. Thus if we want it to work for any case we need the formulae to be polynomials of bidegree

$(2,2)$. In order to compute them in such a way we have created the function `Ew_AddL_b2calc`, which uses a base field `BF`, the list of a_i parameters `lst` and the list `abc_lst` of values a, b, c to compute the corresponding addition law with bidegree $(2,2)$. And as output it not only gives the addition law `[X3,Y3,Z3]`, but also the lists of coefficients of the three polynomials, namely `[X3_ci,Y3_ci,Z3_ci]`, and the list of monomials `monF`. We have arranged it this way since it makes it easier to use the output in other different settings.

As to what computations does `Ew_AddL_b2calc` actually do, it depends on which coordinate of the addition law it is computing. At first p_{Z_3} is a rational expression but as soon as the polynomial division is computed and the condition that $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ represent points on the curve is applied, it becomes a polynomial of bidegree $(2,2)$. And in the case of p_{X_3} and p_{Y_3}, `Ew_AddL_b2calc` needs to solve a system of 88 equations of the form $\sum(\prod a_i)\alpha_{\beta,\gamma} = 0$, where the a_i are the parameters of the Weierstrass equation and the $\alpha_{\beta,\gamma}$ are the unknowns, coefficients of a generic polynomial of bidegree $(2,2)$.

The idea behind how we build such a system starts with defining a generic polynomial of bidegree $(2,2)$:

$$F = \sum_{\substack{\beta_i\geq 0,\sum\beta_i=2\\ \gamma_i\geq 0,\sum\gamma_i=2}} \alpha_{\beta_1,\beta_2,\beta_3,\gamma_1,\gamma_2,\gamma_3} X_1^{\beta_1}Y_1^{\beta_2}Z_1^{\beta_3}X_2^{\gamma_1}Y_2^{\gamma_2}Z_2^{\gamma_3} = \sum_{\substack{\beta_i\geq 0,\sum\beta_i=2\\ \gamma_i\geq 0,\sum\gamma_i=2}} \alpha_{\beta,\gamma} X_1^{\beta_1}Y_1^{\beta_2}Z_1^{\beta_3}X_2^{\gamma_1}Y_2^{\gamma_2}Z_2^{\gamma_3}$$

Then, taking into account that p_{X_3} is a rational expression of the form $p_{X_3} = n_{X_3}/d_{X_3}$, we state that we want p_{X_3} to be expressed as F:

$$p_{X_3} = F \quad \Rightarrow \quad n_{X_3} = d_{X_3}F \quad \Rightarrow \quad n_{X_3} - d_{X_3}F = 0$$

Now we have the polynomial $p = n_{X_3} - d_{X_3}F$ with coefficients of the form $\sum(\prod a_i)\alpha_{\beta,\gamma}$ and monomials that are products of X_1, Y_1, Z_1, X_2, Y_2, Z_2. And, after applying the condition that $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are points on the curve, p has 88 terms, although it is not of bidegree $(2,2)$. But one way to solve the equation $p = 0$ is making sure that all coefficients of p are 0, that is, solving the 88 equations of the form $\sum(\prod a_i)\alpha_{\beta,\gamma} = 0$. By doing so, we will get values for the $\alpha_{\beta,\gamma}$ coefficients such that $p_{X_3} = F$ will hold.

An analogous process happens for p_{Y_3}. To see how to do all this in Magma , feel free to take a look at the source code of `Ew_AddL_b2calc` in [§A.1, Fn.1].

We will use some outputs from `Ew_AddL_b2calc` in our addition function in the next section since we need the coordinates of P_3 and P_3' to be polynomials of bidegree $(2,2)$. Because calling `Ew_AddL_b2calc` every time we want to add two points involves a lot of computations, we have run it once for `BF := Rationals()`, `lst := [a1,a2,a3,a4,a6]` and `abc_lst := [0,0,1]` to obtain P_3 and again with `abc_lst := [0,1,0]` to get P_3', this time getting the correct formulae without the mistake from [9]. And then we simply have taken the outputs to work directly with them in the two functions we have developed: `Ew_AddL`, to check general facts about the system of addition laws, and `Ew_AddL_eval`, to actually compute point addition. We will look into these functions in the next section.

1.2.3 Complete system of addition laws

We have shown above how to compute the addition of two points on an elliptic curve in Magma using the geometric approach. Now we will do the same but using the explicit formulae with bidegree $(2,2)$ that we just computed. So here we will present the function we have created to compute the addition of points on curves given by the Weierstrass equation using the complete system addition laws of bidegree $(2,2)$ discussed in Section 1.1.2.

Let's take a look at `Ew_AddL` first. This function takes as input a base field `BF` (must be rational or finite field) and the list `lst` of the a_i parameters and outputs `[[X3_001,Y3_001,Z3_001], [X3_010,Y3_010,Z3_010]]`, i.e., the system of addition laws. The list of parameters can have either two elements for the short Weierstrass equation or five, for the complete one. Since we want our fact checks to be as general as possible, we will be working with all five parameters so we start by defining the pertinent algebraic structures and obtaining the output from `Ew_AddL`:

```
> BF := Rationals();
> R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
```

```
> R2<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(R1,6);

> Ew1 := Y1^2*Z1+a1*X1*Y1*Z1+a3*Y1*Z1^2-X1^3-a2*X1^2*Z1-a4*X1*Z1^2-a6*Z1^3;
> Ew2 := Y2^2*Z2+a1*X2*Y2*Z2+a3*Y2*Z2^2-X2^3-a2*X2^2*Z2-a4*X2*Z2^2-a6*Z2^3;
> I := ideal< R2 | Ew1, Ew2>;
> QQ<x1,y1,z1,x2,y2,z2> := quo<R2 | I>;

> X3_001 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[1][1]);
> Y3_001 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[1][2]);
> Z3_001 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[1][3]);

> X3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][1]);
> Y3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][2]);
> Z3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][3]);
```

Note that we also define the ideal generated by the Weierstrass equation evaluated in $(X_1 : Y_1 : Z_1)$ and in $(X_2 : Y_2 : Z_2)$. And then we create the quotient ring with the equivalence relation induced by said ideal. Having this quotient ring will allow us to easily apply the condition that $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are points on the curve when needed.

We will need to use this quotient ring for example to check that the output of both addition laws is a point on the curve. So we evaluate the curve equation in P_3 and P_3', coerce the result into the quotient ring and see that the result is 0, as it should be:

```
> QQ ! Evaluate(Ew1,[X3_001,Y3_001,Z3_001,1,1,1]);

0


> QQ ! Evaluate(Ew1,[X3_010,Y3_010,Z3_010,1,1,1]);

0
```

Another condition for the system of addition laws to be well defined is that both addition laws give the same output when the pairs of points being added are not exceptional for either of them. We check that this holds in these lines:

```
> f_XZ := X3_001*Z3_010; g_XZ := Z3_001*X3_010;
> printf "\nX3*Z'3 = X'3*Z3 ? \n   "; (QQ ! f_XZ) eq (QQ ! g_XZ);

X3*Z'3 = X'3*Z3 ?
   true

> f_YZ := Y3_001* Z3_010; g_YZ := Z3_001*Y3_010;
> printf "\nY3*Z'3 = Y'3*Z3 ? \n   "; (QQ ! f_YZ) eq (QQ ! g_YZ);

Y3*Z'3 = Y'3*Z3 ?
   true
```

Lastly, some checks about the exceptional pairs of points should be done, as we will next.

Exceptional points

From the theory we know the conditions on the exceptional pairs for both addition laws, namely that P_1, P_2 are exceptional for P_3 iff $P_1 = P_2$ and they are exceptional for P_3' iff $Y_{P_1-P_2} = 0$. Proving this with MAGMA would mean to prove the following points:

1. $P_1 = P_2 \;\Rightarrow\; P_3(P_1, P_2) = (0 : 0 : 0)$
2. $Y_{P_1-P_2} = 0 \;\Rightarrow\; P_3{}' = (0 : 0 : 0)$

3. $P_3{}'(P_1, P_1) \neq (0:0:0)$ for any $P_1 \in \mathcal{E}(K)$

4. $P_3(P_1, P_2) \neq (0:0:0)$ for any $P_1, P_2 \in \mathcal{E}(K)$ such that $Y_{P_1-P_2} \neq 0$

Ideally we would do all this symbolically and in as general terms as possible. But the large formulae don't allow us to do this in a practical way. In general terms we have checked that the case $P_1 = P_2$ is indeed exceptional for P_3, i.e., point 1 in the list above:

```
> Evaluate(X3_001,[X1,Y1,Z1,X1,Y1,Z1]);

0

> Evaluate(Y3_001,[X1,Y1,Z1,X1,Y1,Z1]);

0

> Evaluate(Z3_001,[X1,Y1,Z1,X1,Y1,Z1]);

0
```

Next, to give an intuitive example that point 3 holds, we'll look at the case $a_1 = a_2 = a_3 = a_4 = 0$, $a_6 \neq 0$ and $Z_1 = 1$ over the rationals, i.e., the Weierstrass equation $Y_2 = X_3 + a_6$. In this case, the P_3' addition law is the following:

```
> a1 := 0; a2 := 0; a3 := 0; a4 := 0;
> X3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][1]);
> Y3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][2]);
> Z3_010 := (R2 ! Ew_AddL(BF,[a1,a2,a3,a4,a6])[2][3]);
>  "X3_010 = ", X3_010; "Y3_010 = ", Y3_010;
 "Z3_010 = ", Z3_010;

X3_010 =  X1*Y1*Y2^2 - 3*a6*X1*Y1*Z2^2 - 6*a6*X1*Z1*Y2*Z2 +
    Y1^2*X2*Y2 - 6*a6*Y1*Z1*X2*Z2 - 3*a6*Z1^2*X2*Y2
Y3_010 =  9*a6*X1^2*X2*Z2 + 9*a6*X1*Z1*X2^2 + Y1^2*Y2^2 -
    9*a6^2*Z1^2*Z2^2
Z3_010 =  3*X1^2*X2*Y2 + 3*X1*Y1*X2^2 + Y1^2*Y2*Z2 +
    Y1*Z1*Y2^2 + 3*a6*Y1*Z1*Z2^2 + 3*a6*Z1^2*Y2*Z2
```

And when evaluated in the case $P_1 = P_2$, this addition law looks like this:

```
> "X3_010_PP = ", X3_010_PP;
> "Y3_010_PP = ", Y3_010_PP;
> "Z3_010_PP = ", Z3_010_PP;

X3_010_PP =  2*X1*Y1^3 - 18*a6*X1*Y1
Y3_010_PP =  18*a6*X1^3 + Y1^4 - 9*a6^2
Z3_010_PP =  6*X1^3*Y1 + 2*Y1^3 + 6*a6*Y1
```

Now the idea is to check that `X3_010_PP = Y3_010_PP = Z3_010_PP = 0` can't happen, because if it does, then $P_1 = P_2$ would also be an exceptional case for P_3'. We can start by factorizing `X3_010_PP`:

```
> Factorization(X3_010_PP);

[
    <Y1, 1>,
    <Y1^2 - 9*a6, 1>,
    <X1, 1>
]
```

Let's see how making any of these three factors equal to 0 while `Y3_010_PP = 0` or `Z3_010_PP = 0` leads to a contradiction:

$$X_1 = 0 \Rightarrow \begin{cases} Z_3' = 2Y_1^3 + 6a_6 Y_1 = 0 \;\Rightarrow\; Y_1^2 = -3a_6 \\ \mathcal{C} : Y_1^2 = a_6 \end{cases} \qquad \Rightarrow \quad a_6 = -3a_6$$

$$Y_1^2 = 9a_6 \Rightarrow \begin{cases} Z_3' = (X_1^3 + 4a_6)Y_1 = 0 \;\Rightarrow\; X_1^3 = -4a_6 \\ \mathcal{C} : 9a_6 = X_1^3 + a_6 \;\Rightarrow\; X_1^3 = 8a_6 \end{cases} \qquad \Rightarrow \quad a_6 = -2a_6$$

$$Y_1 = 0 \Rightarrow \begin{cases} Y_3' = 18a_6 X_1^3 - 9a_6^2 = 0 \;\Rightarrow\; X_1^3 = a_6/2 \\ \mathcal{C} : X_1^3 + a_6 = 0 \end{cases} \qquad \Rightarrow \quad a_6 = -1/2a_6$$

None of the equalities on the right holds since we are working over the rationals and $a_6 \neq 0$. So we see how in this case P_1, P_2 such that $P_1 = P_2$ is an exceptional pair for P_3 but not for P_3'.

Regarding points 2 and 4, there is the added challenge of finding an explicit expression for P_1 and P_2 such that $Y_{P_1,P_2} = 0$, which involves again a large non-linear multivariate system of equations to solve. For point 4 we have found an alternate work-around and for point 2 we will show a numerical example later on.

So point 4 is about checking that $Y_{P_1-P_2}$ is not an exceptional case for P_3. A more general statement that also proves this is that $P_1 = P_2$ is indeed the only exceptional case for P_3. Note that this sounds similar but it is completely different from point 1.

The key to our work-around has been to "play" with two ideals in the ring we call `R2` in our Magma code, which is $\mathbb{Q}(a_1, a_2, a_3, a_4, a_6)\ [X_1, Y_1, Z_1, X_2, Y_2, Z_2]$:

$$I = \langle F_w(X_1, Y_1, Z_1),\ F_w(X_2, Y_2, Z_2),\ X_3,\ Y_3,\ Z_3 \rangle$$

$$J = \langle F_w(X_1, Y_1, Z_1),\ F_w(X_2, Y_2, Z_2),\ X_1Z_2 - X_2Z_1,\ Y_1Z_2 - Y_2Z_1,\ X_1Y_2 - X_2Y_1 \rangle$$

Here $F_w = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3$ is the projective polynomial that defines the Weierstrass form. We need to include this polynomial evaluated in both points to let Magma know that the variables X_1, Y_1, Z_1, X_2, Y_2, Z_2 actually define points on the curve.

The ideal I is defined then by the polynomials that define P_3 and J is defined by three polynomials that express the case $P_1 = P_2$, since:

$$P_1 = P_2 \;\Rightarrow\; \frac{X_1}{Y_1} = \frac{X_2}{Y_2},\ \frac{X_1}{Z_1} = \frac{X_2}{Z_2},\ \frac{Y_1}{Z_1} = \frac{Y_2}{Z_2} \;\Rightarrow\; X_1Y_2 = X_2Y_1,\ X_1Z_2 = X_2Z_1,\ Y_1Z_2 = Y_2Z_1$$

The general idea is to prove that $I = J$, because in that case it would mean that X_3, Y_3 and Z_3 vanish exactly when $X_1Z_2 - X_2Z_1$, $Y_1Z_2 - Y_2Z_1$, $X_1Y_2 - X_2Y_1$ vanish as well. And thus we would have that point 4 holds.

In order to see that $I = J$, we check both inclusions: $I \subset J$ and $J \subset I$. Magma agrees that $I \subset J$ holds, but when we check $J \subset I$ it yields `false`. There are a number of reasons why this would happen so we compute the Gröbner basis for J and check one by one if all the elements of said basis are in I. And once again we find `false` outputs.

Keeping in mind that we are actually only interested in when do the elements of both of these ideal vanish, we can still prove point 4 by checking if any of the power of the ideal J is a subset of I. We define the n-th power of J as follows:

$$J^n = \{\alpha \cdot \beta \mid \alpha \in J, \beta \in J^{n-1}\}$$

So the plan is to start by computing J^2 and see if $J^2 \subset I$. If not, try with J^3 and so on until we find an n such that $J^n \subset I$.

We have faced some problems running days-long computations so we haven't been able to see the end result of computing this for the complete Weierstrass model. The code to do so and the output showing how far we have been able to run the computation are available online in the input file WC_AdL_001exceptionalpts_completeWC.m and in the output file WC_AdL_001exceptionalpts_completeWC_output.txt, respectively.

We have however completed the calculations for the short Weierstrass case, which is still a pretty general case since it is isomorphic to the complete Weierstrass form except for the cases of K begin a field of characteristic 2 or 3. And we have found that $J^4 \subset I$. The code to see how it's done is also available online in the file WC_AdL_001exceptionalpts_shortWC.m.

For the symbolic checks we haven't done we trust [Thm. 2 [9]], which tells us that our system of addition laws is complete. And so we proceed to see it deal with some numerical examples.

Numerical examples

For this kind of examples we have the function `Ew_AddL_eval`, which takes as input a base field `BF`, the list `lst` of curve parameters and two points `P` and `Q` with projective coordinates. As output of course it yields `P+Q`, computed using whichever addition law is suitable in each case. `Ew_AddL_eval` is programmed in the spirit of working for cases as general as possible. So, for example, it can take as input both numerical values and undetermined parameters and/or variables. Furthermore, we want to work with a `PolynomialRing` defined over a `FunctionField` as much as possible, since that allows us to work with ideals and other algebraic structures when needed. However, `FunctionField` is not supported over certain rings, as for example the reals. Thus we include the option to change to a `PolynomialRing` defined over another `PolynomialRing` when needed. This is done by these lines at the beginning of the function:

```
if (Type(BF) eq Type(RealField())) or (Type(BF) eq Type(ComplexField())) then
    fl := true;
else
    fl := false;
end if;

if fl then
    R1<a1,a2,a3,a4,a6> := PolynomialRing(BF,5);
else
    R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
end if;
R2<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(R1,6);
```

Afterwards the function reads the input data and goes ahead to decide which addition law will it use by assessing whether it's dealing with point doubling or something else:

```
if fl then
    if (Abs(BF!(Xp - Xq)) gt 1E-10) and (Abs(BF!(Yp - Yq)) gt 1E-10) and
(Abs(BF!(Zp - Zq)) gt 1E-10) then
        //here we use (a,b,c) = (0,1,0)
        X3 := R2!(-a1*a2*Xp^2*Xq^2 + ... + a3*a4^2*Zp^2*Zq^2);
        Y3 := R2!((-a2^2 + 3*a4)*Xp^2*Xq^2 + ... - 9*a6^2*Zp^2*Zq^2);
        Z3 := R2!(3*a1*Xp^2*Xq^2 + ... + (a3^3 + 3*a3*a6)*Zp^2*Zq^2);
    else
        //here we use (a,b,c) = (0,0,1)
        X3 := R2!(-a2*Xp^2*Xq*Zq + ... + 3*a6*Zp^2*Xq*Zq);
        Y3 := R2!(-3*Xp^2*Xq*Yq + ... + (-a3^2 - 3*a6)*Zp^2*Yq*Zq);
        Z3 := R2!(3*Xp^2*Xq*Zq + ... + a3*Zp^2*Yq*Zq);
    end if;
else
    if (Xp eq Xq) and (Yp eq Yq) and (Zp eq Zq) then
```

```
            //here we use (a,b,c) = (0,1,0)
            X3 := R2!(-a1*a2*Xp^2*Xq^2 + ... + a3*a4^2*Zp^2*Zq^2);
            Y3 := R2!((-a2^2 + 3*a4)*Xp^2*Xq^2 + ... - 9*a6^2*Zp^2*Zq^2);
            Z3 := R2!(3*a1*Xp^2*Xq^2 + ... + (a3^3 + 3*a3*a6)*Zp^2*Zq^2);
        else
            //here we use (a,b,c) = (0,0,1)
            X3 := R2!(-a2*Xp^2*Xq*Zq + ... + 3*a6*Zp^2*Xq*Zq);
            Y3 := R2!(-3*Xp^2*Xq*Yq + ... + (-a3^2 - 3*a6)*Zp^2*Yq*Zq);
            Z3 := R2!(3*Xp^2*Xq*Zq + ... + a3*Zp^2*Yq*Zq);
        end if;
    end if;
```

Recall the fact that we took the addition laws from the output of `Ew_AddL_b2calc` and due to their length, they appear shortened above. Also remember that in the case `(a,b,c) = (0,1,0)` the `X3` and `Y3` coordinates are different from the ones shown in the original paper due to the fixed error. Note as well that here we also separate the case of a rational or finite base field from the rest. This is due to the fact that, more often than not, computations that should evaluate to 0 don't yield an exact 0 when working with `float` values. So instead of checking if any said variable is not equal to zero, we check that its value is "far away enough" from zero., i.e., if it's greater than 10^{-10}.

Let's take a look now at a couple of examples to take `Ew_AddL_eval` for test run. First we'll try something with some unknowns over a finite field:

```
> BF := FiniteField(11);
> R1<a1,a2,a3,a4,a6> := PolynomialRing(BF,5);
> R2<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(R1,6);

> Ew_AddL_eval(BF,[R1|0, a2, 0, 5, a6],[R2|3/4,Y1,Z1],[R2|0,-7,Z2]);

[
    6*Y1*Z2 + 6*a6*Z1*Z2^2 + Z1 + 2*Z2^2,
    7*Y1^2*Z2 + 3*a6*Y1*Z1*Z2^2 + 5*Y1*Z1 + Y1*Z2^2 + 10*a6*Z1^2*Z2 +
        3*Z1*Z2 + 6*a2*Z2,
    10*Y1^2*Z2^2 + 5*Z1^2 + Z1*Z2^2 + 4*a2*Z2^2
]
```

Now for a curve given by a complete Weierstrass equation over the rationals:

```
> BF := Rationals();
> R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
> R2<x,y> := AffineSpace(R1,2);
> a1 := 19; a2 := 2; a3 := -13/7; a4 := -5; a6 := -6;
> C := Curve(R2, y^2+a1*x*y+a3*y-x^3-a2*x^2-a4*x-a6);
> P1 := [2, -253/7, 1]; P2 := [-33/49, 4978/343, 1];

> PQ := Ew_AddL_eval(BF,[a1, a2, a3, a4, a6],P1,P2); PQ;
> printf "\n Is P+Q in the curve? %o \n",
 R2![R1|PQ[1]/PQ[3],PQ[2]/PQ[3]] in C;

[
    9708017/117649,
    -5204592/823543,
    -2248091/117649
]

Is P+Q in the curve? true
```

What we couldn't do before symbolically regarding point 2, we can do know with a numerical case: check that P_1, P_2 such that $Y_{P_1-P_2}$ is an exceptional pair for P_3'. Let's consider the same curve

as in the previous example and the points `P := [-33/49, 4978/343, 1]`, `Q := [8, 27/7, 1]` and `mQ := [8, -154,1]`, the opposite of `Q`. Then we can easily check that $Y_{P-Q} = 0$ and that `P` and `Q` are exceptional for P_3':

```
> Ew_AddL_eval(BF,lst,P,mQ);

[
    -76765625/117649,
    0,
    76765625/117649
]

> Evaluate(X3_010, P cat Q);
> Evaluate(X3_010, P cat Q);
> Evaluate(X3_010, P cat Q);

0
0
0
```

But of course `Ew_AddL_eval` does not fall for exceptional cases and always gives the right output:

```
> "Point doubling: \n ", Ew_AddL_eval(BF,lst,[-3, 412/7, 1],
 [-3, 412/7, 1]);

> "P,Q such that P - Q = [-1 ,0 ,1]: \n", Ew_AddL_eval(BF,lst,
 [-33/49, 4978/343, 1],[8, 27/7, 1]);

Point doubling:
[
    55997392/343,
    -921640/7,
    69934528/343
]

P,Q such that P - Q = [-1 ,0 ,1]:
[
    -2392016875/117649,
    291210705500/823543,
    76765625/117649
]
```

These and more examples can be found in the online file `WC_AdL_eval.m`. And the source code for both `Ew_AddL` and `Ew_AddL_eval` can be found in [§A.1, Fn.2] and [§A.3, Fn.6] respectively.

Chapter 2

Edwards model

Let's study now the addition law for another form of elliptic curves: the Edwards curves. This model was introduced by H. Edwards in 2007 in [19] and it has the advantage that it is overall simpler to work with than the Weierstrass model. We will see why in the coming sections.

A point that might be a bit confusing is that the normal form Edwards introduced is not what most people use nowadays, instead it was as follows:

The normal form of an elliptic curve $\mathcal{C}$ over a number field K is $E\colon x^2+y^2=c^2(1+x^2y^2)$, where $c\in K$ is such that $c^5\neq c$.

Note that the condition on c implies that $c\notin\{0,1\}$. Edwards proved that all elliptic curves can be represented in this normal form, but in some cases it is necessary to work on an extension of the original field over which the curve was defined. In order to include a larger class of elliptic curves over the original field, Bernstein and Lange in [5] expanded the concept of Edwards curves to:

The Edwards form of an elliptic curve $\mathcal{C}$ over a number field K is $\mathcal{E}_d^c\colon x^2+y^2=c^2(1+dx^2y^2)$, where $c,d\in K\backslash\{0,1\}$ and $cd(1-dc^4)\neq 0$.

And this is the form most authors refer to as "Edwards model" or "Edwards-Bernstein model". However, the most used case of this form is the one where $c=1$ and we will work mainly in such case. Thus let us introduce a formal definition for it:

Definition 14. Let K be a field such that $\mathrm{char}(K)\neq 2$ and $\mathcal{C}$ an elliptic curve over K, then its Edwards form is $\mathcal{E}_d\colon x^2+y^2=1+dx^2y^2$, where $d\in K\backslash\{0,1\}$.

In the following sections we'll study the addition law for this model, which will prove to be simpler than the addition law introduced in the previous chapter. We will also see how this model relates to the Weierstrass one, which will give us an idea of all the curves that can be represented in Edwards form. And we will leave the geometric interpretation for the Edwards addition law for the next chapter, since it is generalized for the twisted Edwards model.

2.1 Addition law

The addition law for elliptic curves in Edwards form was first stated in [19] for the normal form and in [[5], § 3] Bernstein and Lange adapted it for the Edwards model and studied it. In this section we will present the addition law and its main properties as shown in [5].

First, let's introduce the addition law:

Let K be a field such that $\mathrm{char}(K)\neq 2$, $\mathcal{C}$ an elliptic curve over K and (x_1,y_1), (x_2,y_2) two points on $\mathcal{C}$. Fix $c,d\in K\backslash\{0,1\}$ to be such that $cd(1-dc^4)\neq 0$. Then the addition law of $\mathcal{C}$ on its Edwards form $\mathcal{E}_d^c\colon x^2+y^2=c^2(1+x^2y^2)$ is as follows:

$$(x_1,y_1)+(x_2,y_2)=(x_3,y_3)=\left(\frac{x_1y_2+y_1x_2}{c(1+dx_1x_2y_1y_2)},\quad\frac{y_1y_2-x_1x_2}{c(1-dx_1x_2y_1y_2)}\right)$$

The set of K-rational points $\mathcal{E}_d^c(K)$ forms an abelian group under this addition law. The neutral element for this addition law is $(0, c)$ and the inverse of a point $P = (x, y)$ is $-P = (-x, y)$. It holds that $(x_3, y_3) \in \mathcal{E}_d^c(K)$ [Thm. 3.1 [5]]. And, if d is a non-square in K, this addition law is complete in K, i.e., there are no exceptional pairs of points in $\mathcal{E}_d^c(K)$ for it [Thm. 3.3 [5]].

This fact might seem to contradict Theorem 1 in [9], but it actually just refers to a different situation: in [9] the authors are working and studying the exceptional points on the algebraic closure of K, while here the result is over K, so possible exceptional points that are not in $\mathcal{E}_d^c(K)$ are not taken into account.

Also the output of this addition law corresponds to the output of the standard addition law on a birationally equivalent elliptic curve on Weierstrass form. The correspondence is the following, as detailed in [Thm. 3.2, [5]]:

$$h\colon \mathcal{E}_d^c\colon x^2 + y^2 = c^2(1 + x^2y^2) \longrightarrow \mathcal{E}'\colon \frac{1}{e}v^2 = u^3 + \left(\frac{4}{e} - 2\right)u^2 + u, \text{ where } e = 1 - dc^4 \text{ and}$$

$$h(x, y) = \begin{cases} \infty, & \text{if } (x, y) = (0, c) \\ (0, 0), & \text{if } (x, y) = (0, -c) \\ \left(\frac{c+y}{c-y}, \frac{2c(c+y)}{c-y}\right), & \text{if } x \neq 0 \end{cases}$$

According to [5], if $(x_1, y_1), (x_2, y_2) \in \mathcal{E}_d^c(K)$ and $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, then $h(x_1, y_1) \oplus h(x_2, y_2) = h(x_3, y_3)$, where $\oplus$ is used here to indicate the addition law on the Weierstrass model.

Note that $\mathcal{E}'$ is a quadratic twist of $\mathcal{E}_w$. In the next section we will study more thoroughly how Weierstrass and Edwards models relate and so we will explain this further.

2.2 Relation to Weierstrass model

Taking into account that both the Edwards and the Weierstrass forms are models for elliptic curves, the question that comes naturally is: how do they relate to each other? The key to answer this question is provided by D.J. Bernstein and T. Lange in [[5], §2]. D. Nguyen has studied this deeper in [32] and it is also part of the work M.R. Dam did in [16]. And here we are going to show some of their most relevant results for our research.

The short answer to our question could be "they are birationally equivalent". But, as most things in life, this is not so simple. Theorem 2.1 in [5] gives us a more complete insight in the matter, as it states the following:

Theorem 1. Let K be a field with $\text{char}(K) \neq 2$. Let $\mathcal{C}$ be an elliptic curve over K with model $\mathcal{E}_w$ such that the group $\mathcal{E}_w(K)$ has an element of order 4. Then:

1. there exists $d \in K \backslash \{0, 1\}$ such that the curve defined by $\mathcal{E}_d\colon x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent over K to a quadratic twist of $\mathcal{E}_w$;
2. if $\mathcal{E}_w(K)$ has a unique element of order 2 then there is a nonsquare $d \in K$ such that the curve defined by $\mathcal{E}_d\colon x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent over K to a quadratic twist of $\mathcal{C}$; and
3. if K is finite and $\mathcal{E}_w(K)$ has a unique element of order 2 then there is a non-square $d \in K$ such that the curve defined by $\mathcal{E}_d\colon x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent over K to $\mathcal{C}$.

So, as we can see, curves in Weierstrass and Edwards forms are indeed birationally equivalent, but when the right conditions apply. Although some of these conditions are not as restrictive as one might think at first sight. For example, the following hold:

Proposition 1. Let $\mathcal{C}$ be an elliptic curve defined over K.

1. If $\mathcal{C}$'s model is $\mathcal{E}_w$, then there exists an extension K' of K such that $\mathcal{E}_w(K')$ has an element of order 4 and $[K'\colon K] \leq 3$.
2. If $\mathcal{C}$ is in $\mathcal{E}_d$ form, then $\mathcal{E}_d(K)$ always has at least one element of order 4.

Proof. Regarding the first part of the proposition, in [[32], p.6] we have a recipe to find said element of order 4: the general idea is to find a point $P = (u_p, v_p) \in \mathcal{E}_w(K)$ with $v_p \neq 0$ such that the tangent line to $\mathcal{C}$ in P intersects $\mathcal{C}$ in a point $R = (u_R, 0)$, with $u_R \in \overline{K}$. We know that R will have order 2 by construction and thus P will have order 4 since $2P = -R = R \Rightarrow 4P = 2R = \mathcal{O}_w$. To find P, one can build a system of 2 equations with variables u_p, v_p and a solution in an extension K' of K.

Looking more closely at that system of equations, we see that to solve it we have to find a solution for $f(u_p) = 0$, where f is a polynomial of degree 3. And with u_p we can compute v_p. Thus if $u_p \in K$, then $K' = K$ and otherwise, $u_p \in K'$ and $[K' \colon K] = 3$.

As for the second part of the proposition, we can proof that the points $\pm P = (\pm 1, 0)$ always have order 4: it is obvious that $\pm P \in \mathcal{E}_d(K)$ and, using the addition law from the previous section, we can see that, for instance: $2P = (0, -1) \Rightarrow 3P = 2P + P = (-1, 0) = -P \Rightarrow 4P = P + 3P = P - P = \mathcal{O}_d$. •

Therefore, from this proposition we see that all curves either in form $\mathcal{E}_w$ or $\mathcal{E}_d$ have a point of order 4. And from the proof of Theorem 1, which can be found in [[5], p.4], we can extract some other interesting facts, which we compile and elaborate on here:

Remark 1. Notes on Theorem 1:

1. If $P = (u_p, v_p) \in \mathcal{E}_w(K)$ is the point of order 4, then $d = 1 - 4u_p^3/v_p^2$. Note that this d is not necessarily a non-square. Note also that this only establishes how to go from $\mathcal{E}_w$ to $\mathcal{E}_d$ but not the other way around, since we need P.

2. If d is a square, then $\mathcal{E}_d(K)$ has at least 2 points of order 2, namely $2P = 2\big(u_p, v_p\big)$ and $\Big(u_p \frac{\sqrt{d}+1}{\sqrt{d}-1}, 0\Big)$. Thus if $\mathcal{E}_d(K)$ has a unique point of order 2, then d is non-square.

3. The quadractic twist of $\mathcal{E}_w$ in the 2nd instance of the theorem is also explicitly shown in the proof and is the following:

$$\mathcal{E}'_w \colon \frac{u_p}{1-d}\bar{s}^2 = \bar{r}^3 + a_2\bar{r}^2 + a_4\bar{r}$$

   Applying the change of coordinates $(\bar{r}, \bar{s}) = \Big(\bar{u}, \frac{\sqrt{1-d}}{\sqrt{u_p}}\bar{v}\Big)$ to $\mathcal{E}'_w$ yields a curve in form $\mathcal{E}_w$ such that $a_1 = a_3 = a_6 = 0$.

4. Note that in point 3 of the theorem we state that $\mathcal{E}_d$ is birationally equivalent to $\mathcal{E}_w$ directly, not to any quadratic twist of it. That is because if K is finite and d is not a square in K, then $\mathcal{E}_w$ is actually isomorphic over K to either $\mathcal{E}'_w$ or to another similar quadratic twist.

   Note as well that, if K is finite, then $u_p/(1-d)$ will be a square if and only if u_p and $1-d$ are both squares or if neither of them is, since the product of two non squares in a finite field is a square. From this it also follows that, if d and $u_p/(1-d)$ are both not squares, then $du_p/(1-d)$ would also be a square. This gives rise to two cases, assuming d is not a square in K:

   (a) If K is finite and $\frac{u_p}{(1-d)}$ is a square in K, then $\mathcal{E}_w$ is isomorphic to $\mathcal{E}'_w$, and thus birationally equivalent to $x^2 + y^2 = 1 + dx^2y^2$.

   (b) If K is finite and $\frac{u_p}{(1-d)}$ is not a square in K, then $\frac{du_p}{1-d}$ is a square and $\mathcal{E}_w$ is isomorphic to $\mathcal{E}'_w$ but changing d for $1/d$ and $\bar{r}$ for $-\bar{r}$. This yields the following:

$$\mathcal{E}''_w \colon \frac{du_p}{1-d}\bar{s}^2 = \bar{r}^3 + a_2\bar{r}^2 + a_4\bar{r}$$

   Therefore, in this case $\mathcal{E}_w$ will be birationally equivalent to $x^2 + y^2 = 1 + (1/d)x^2y^2$ and the change of coordinates to get to get $\mathcal{E}_w$ from $\mathcal{E}''_w$ will be $(\bar{r}, \bar{s}) = \Big(\bar{u}, \frac{\sqrt{1-d}}{\sqrt{du_p}}\bar{v}\Big)$.

While this case distinction is necessary and valuable considering a general unknown d, we will focus on the cases where $d = 1 - 4u_p^3/v_p^2$. And in that case, note that $u_p/(1-d) = v_p^2/(4u_p^2)$, which is always a square. So it will suffice for us to focus from here on out on the birational equivalence between $\mathcal{E}_d$ and $\mathcal{E}_w$ through the quadratic twist $\mathcal{E}'_w$.

Let's consider a curve in $\mathcal{E}_w$ form such that $a_1 = a_3 = a_6 = 0$, which we will refer to as $\bar{\mathcal{E}}_w$. To handle the general case for the complete Weierstrass, we have the following relation where u_{2p} is such

that $2P = 2(u_p, v_p) = (u_{2p}, v_{2p})$ for the point P of order 4:

$$\mathcal{E}_w \colon v^2 + a_1uv + a_3v = u^3 + a_2u^2 + a_4u + a_6 \longrightarrow \bar{\mathcal{E}}_w \colon \bar{v}^2 = \bar{u}^3 + \bar{a}_2\bar{u}^2 + \bar{a}_4\bar{u}$$

$$a_1, a_2, a_3, a_4, a_6 \rightsquigarrow \bar{a}_2 = a_2 - 3u_{2p} + \frac{a_1^2}{4},$$

$$\bar{a}_4 = a_4 + 3u_{2p}^2 - 2a_2u_{2p} + \frac{a_1a_3}{2} - \frac{a_1^2u_{2p}}{2}$$

$$(u, v) \longmapsto (\bar{u}, \bar{v}) = \left(u + u_{2p}, v + \frac{a_1u + a_3}{2}\right)$$

So for $d = 1 - 4u_p^3/v_p^2$, $u_p/(1-d)$ is always a square and thus $\bar{\mathcal{E}}_w$ is isomorphic to the quadratic twist $\mathcal{E}'_w$, which is as well isomorphic to this other curve $\mathcal{E}'$:

$$\bar{\mathcal{E}}_w \longrightarrow \mathcal{E}'_w \longrightarrow \mathcal{E}' \colon \frac{1}{1-d}s^2 = r^3 + 2\frac{1+d}{1-d}r^2 + r$$

$$(\bar{r}, \bar{s}) \longmapsto (r, s) = \left(\frac{\bar{r}}{u_p}, \frac{\bar{s}}{u_p}\right)$$

$$(\bar{u}, \bar{v}) \longmapsto (r, s) = \left(\frac{\bar{u}}{u_p}, \frac{2\bar{v}}{v_p}\right)$$

Note that, with some changes in notation, $\mathcal{E}'$ is equivalent to the quadratic twist of $\mathcal{E}_w$ given in the previous section. The parameters of $\bar{\mathcal{E}}_w$ can also be written in terms of P as follows:

$$\bar{a}_2 = \frac{v_p^2}{u_p^2} - 2u_p = 2\frac{1+d}{1-d}u_p \; ; \quad \bar{a}_4 = u_p^2$$

Now that we know $\mathcal{E}'$ and how it related to $\mathcal{E}_w$, we are ready to finally introduce the rational map provided by Bernstein and Lange in [[5], §2]:

There are a finite number of points such that $s(r+1) = 0$ and $x(1-y) = 0$, i.e., there are finitely many exceptional points to these maps. Thus they settle a birational equivalence between $\mathcal{E}'$ and $\mathcal{E}_d$.

$$\mathcal{E}' \longrightarrow \mathcal{E}_d \qquad\qquad \mathcal{E}_d \longrightarrow \mathcal{E}'$$

$$(r, s) \longmapsto (x, y) = \left(\frac{2r}{s}, \frac{r-1}{r+1}\right) \qquad (x, y) \longmapsto (r, s) = \left(\frac{1+y}{1-y}, \frac{2(1+y)}{x(1-y)}\right)$$

Getting all these maps together, we can explicitly see the relation between $\mathcal{E}_w$ and $\mathcal{E}_d$:

Let $P = (u_p, v_p) \in \mathcal{E}_w(K)$ be a point of order 4 and $d = 1 - 4u_p^3/v_p^2$. Then the following maps establish a birational equivalence between $\bar{\mathcal{E}}_w$ and $\mathcal{E}_d$:

$$\phi \colon \bar{\mathcal{E}}_w \longrightarrow \mathcal{E}_d \qquad\qquad \phi^{-1} \colon \mathcal{E}_d \longrightarrow \bar{\mathcal{E}}_w$$

$$(\bar{u}, \bar{v}) \longmapsto (x, y) = \left(\frac{v_p\bar{u}}{u_p\bar{v}}, \frac{\bar{u} - u_p}{\bar{u} + u_p}\right) \qquad (x, y) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{u_p(1+y)}{1-y}, \frac{v_p(1+y)}{x(1-y)}\right)$$

The limitation of these maps is the fact that we need to know the point P. So even though we have the reverse, it'd be nice to have a map that does not depend on P. Luckily, there is a way to rewrite $\mathcal{E}_d$ to be a quartic curve and then rewrite that one into $\mathcal{E}_w$ form, as it is done in [[16], §4.3]. This process produces the following:

Let $d \in K \backslash \{0, 1\}$ and $A = 2y - (2dy + d + 1)x^2 + 2$. Then the following maps are well-defined:

$$\psi\colon \bar{\mathcal{E}}_w \longrightarrow \mathcal{E}_d \qquad\qquad \psi^{-1}\colon \mathcal{E}_d \longrightarrow \bar{\mathcal{E}}_w$$

$$\bar{a}_2, \bar{a}_4 \rightsquigarrow \ d = \frac{\bar{a}_2}{2} - 1 \qquad\qquad d \rightsquigarrow \ \bar{a}_2 = 2(d+1), \bar{a}_4 = (d-1)^2$$

$$(\bar{u}, \bar{v}) \longmapsto (x, y) = \left(\frac{-2\bar{u}}{\bar{v}}, \frac{\bar{v}^2 - \bar{a}_2\bar{u}^2 - 2\bar{u}^3}{2(\bar{a}_2 - 2)\bar{u}^2 - \bar{v}^2}\right) \qquad (x, y) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{A}{x^2}, \frac{-2A}{x^3}\right)$$

One more interesting fact about these maps is that they share the following property with the ϕ, ϕ^{-1} maps: given the point $(1, 0) \in \mathcal{E}_d(K)$, $\psi^{-1}(1, 0) \in \mathcal{E}_w(K)$ will always be a point of order 4. And, since ψ^{-1} doesn't depend on any point of order 4, it can actually be used to find a point of order 4 in any given Weierstrass curve as long as we know the parameter d for its birational equivalent Edwards curve. Furthermore, for such d we know now that a point of order 4 is of the form $(u_p, v_p) = (-d + 1, 2d - 2)$.

2.3 Implementations in Magma

Our work in MAGMA regarding the Edwards model is the implementation of both of the birational equivalences between this model and the Weierstrass one. We also have taken some examples to show that the addition is preserved through these maps. However, we haven't implemented the Edwards addition law since we have its generalization for the twisted Edwards model (see 3.1.2 and 3.3.2).

So let us start by showing what can be done with the maps we have for the Bernstein - Lange birational equivalence, i.e., the ϕ maps. There are two functions for each map for the birational equivalence: `Ew2Ed_P4_symb` and `Ed2Ew_P4_symb` yield the maps symbolically so general checks can be done while `Ew2Ed_P4_eval` and `Ed2Ew_P4_eval` deal with the numerical cases. These maps are a bit more general than the ϕ explicitly shown above since they have the change from complete Weierstrass to $\bar{\mathcal{E}}_w$ also implemented in them so they can be used more widely.

`Ew2Ed_P4_symb` [Fn. 10] takes as input the base field `BF`, the list `lst` of curve parameters a_i and the point `P4` of order 4. And as a parameter we have the double of said point `PP4`, set as a default to `[0,0]`. But we might need to change it if our input is in complete Weierstrass form to adjust the coordinates to the $(-3, 6)$, which is the input our map is actually defined for. This is done by these lines of code in the function:

```
u2p := R ! PP4[1]; v2p := R ! PP4[2];

up +:= u2p; vp +:= (a1*up + a3)/2;
u +:= u2p; v +:= (a1*u + a3)/2;
```

On the other hand, we have `Ed2Ew_P4_symb` [Fn. 11] which works in a similar way but in the other direction. In this case, the input `lst` is for the curve parameter d. And as parameters, aside from `PP4 := [0,0]` we also have `aa1 := 0` and `aa3 := 0`. The following snip of code shows the adjustments made in the function to the output `[*[a1,a2,a3,a4,a6],[c_u,c_v]*]`. Note that unless one of the function parameters is non-zero, the output will be in $\bar{\mathcal{E}}_w$ form:

```
c_u -:= u2p; c_v -:= (a1*(c_u-u2p)+a3)/2;
a2 -:= -3*u2p + (a1^2)/4;
a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;
```

With these two functions we can retrieve the ϕ maps to use whenever needed. Or, for example, we can check if the maps are inverse of each other as follows:

```
> BF := Rationals();
> R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
> Rw_var<u,v> := FunctionField(R,2);
> Rd_var<x,y> := FunctionField(R,2);
```

These first lines above establish the fields we will need and define the names of the variables we will be using. The notation is the same we have used so far: $(u, v) \in \mathcal{E}_w(K)$ and $(x, y) \in \mathcal{E}_d(K)$.

Next we get $\phi(u, v)$ using `Ew2Ed_P4_symb`, for which we will use the notation `[xx,yy]`:

```
> xx := Evaluate(Ew2Ed_P4_symb(BF,[0,a2,0,a4,a6],[up,vp])[2][1],[u,v]);
> yy := Evaluate(Ew2Ed_P4_symb(BF,[0,a2,0,a4,a6],[up,vp])[2][2],[u,v]);
> [xx,yy];

[
    vp/up*u/v,
    (u - up)/(u + up)
]
```

Similarly, `Ed2Ew_P4_symb` provides us with $\phi^{-1}(x, y)$:

```
> uu := Evaluate(Ed2Ew_P4_symb(BF,[d],[up,vp])[2][1],[x,y]);
> vv := Evaluate(Ed2Ew_P4_symb(BF,[d],[up,vp])[2][2],[x,y]);
> [uu,vv];

[
    (-up*y - up)/(y - 1),
    (-vp*y - vp)/(x*y - x)
]
```

And now all is left is to see if $(u, v) = \phi^{-1}(\phi(u, v))$ and if $(x, y) = \phi(\phi^{-1}(x, y))$:

```
> [u,v] eq [Evaluate(uu,[xx,yy]), Evaluate(vv,[xx,yy])];

true

> [x,y] eq [Evaluate(xx,[uu,vv]), Evaluate(yy,[uu,vv])];

true
```

Next, let's take a peak at the inner workings of `Ew2Ed_P4_eval` and `Ed2Ew_P4_eval`. The input for these functions is the same as for their symbolic counterparts, adding only one more necessary piece of data: the point `p` on which we want to apply the birational equivalence. And their outputs are `[*[d],[c_x,c_y]*]` and `[*[a1,a2,a3,a4,a6],[c_u,c_v]*]` respectively, i.e., the parameters of the birationally equivalent curve in each case and, naturally, the coordinates of the corresponding point.

Both functions read their input and, of course, do the necessary calculations to yield their corresponding output, being careful not to do any divisions by zero, That is the reason why some `if` statements are in place, ready to return an error if that were to happen. For example, this is the core of `Ed2Ew_P4_eval`:

```
denom_x := (1-y); denom_y := x*(1-y);
if denom_x eq 0 then
    return "Error: birat.eq. not def. for this point since (denom_x eq 0).";
elif denom_y eq 0 then
 return "Error: birat.eq. not def. for this point since (denom_y eq 0).";
else
    a2 := 2*(1+d)/(1-d)*up; a4 := up^2; a6 := 0;
    c_u := up*(1+y)/denom_x; c_v:= vp*(1+y)/denom_y;

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;

    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

    return [*[a1,a2,a3,a4,a6],[c_u,c_v]*];
end if;
```

Let's take a look at a couple of numerical examples now. The first example we will show starts with a curve in $\bar{\mathcal{E}}_w$ form with parameters $a_2 = 10$, $a_4 = 9$ and $(-3, \pm 6) \in \mathcal{E}_w(\mathbb{Q})$. Both this points are of order 4, so we will use one of them as `P4` and another one as `p`. So here we step up the data for the example:

```
> BF := Rationals();
> R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
> Rw_var<u,v> := AffineSpace(R,2);
> Rd_var<x,y> := AffineSpace(R,2);

> a1 := 0; a2 := 10; a3 := 0; a4 := 9; a6 := 0;
> C := Curve(Rw_var, v^2 - u^3 - a2*u^2 - a4*u - a6);
> lst := [a1,a2,a3,a4,a6];
> P4 := [-3, -6]; p:= [-3, 6];
```

With this data we can run `Ew2Ed_P4_eval` and check if the resulting point is in the resulting curve in Edwards form:

```
> ch := Ew2Ed_P4_eval(BF,lst,P4,p);

> d := ch[1][1];
> C := Curve(Rd_var, 1 + d*x^2*y^2 - x^2 - y^2); C;
> p_ad := Rd_var![ch[2][1],ch[2][2]]; "\n p_ad := ", p_ad;
> tf,pt := p_ad in C; "\n Is p_ad in the EdC? \n\t", tf;

Curve over Multivariate rational function field of rank 8 over Rational
Field defined by
4*x^2*y^2 - x^2 - y^2 + 1

p_ad :=  (-1, 0)

Is p_ad in the EdC?
         true
```

So far so good! Now we can take this curve in $\mathcal{E}_d$ form and bring it back to the $\mathcal{E}_w$ model, with the twist of the fact that we will go for a complete Weierstrass model this time. So we need a different point `P4` of order 4, its double `PP4` and values for `a1` and `a3`. We do not have to specify `a6` since it will be computed from the other parameters. With some reverse-engineering calculations, we have all the necessary data for this case:

```
> lst := [d];
> C := Curve(Rd_var, 1 + d*x^2*y^2 - x^2 - y^2); C;
> p := p_ad;
> P4 := [-3, -4545/2]; PP4 := [0,0]; a1 := 724; a3 := 6705;
```

So running now `Ed2Ew_P4_eval` we have the following:

```
> ch := Ed2Ew_P4_eval(BF,lst,P4,p : PP4 := [0,0], a1 := a1, a3 := a3);

> a1 := ch[1][1]; a2 := ch[1][2]; a3 := ch[1][3];
  a4 := ch[1][4]; a6 := ch[1][5];
> C := Curve(Rw_var, v^2 + a1*u*v + a3*v - u^3 - a2*u^2 - a4*u - a6); C;
> p_w := Rw_var![ch[2][1],ch[2][2]]; "\n p_w := ", p_w;
> tf,pt := p_w in C; "\n Is p_w in Ew? \n\t", tf;

Curve over Multivariate rational function field of rank 8 over Rational
Field defined by
-u^3 + 131034*u^2 + 724*u*v + 2427201*u + v^2 + 6705*v + 44957025/4

 p_w :=  (-3, -4521/2)

 Is p_w in Ew?
         true
```

As we can see, the output curve is in complete Weierstrass form. If we compare the point $(-3, 6)$ we started with, given in $\bar{\mathcal{E}}_w$ form, and `p_w`, only the y coordinate has been affected by the change to complete Weierstrass since in this case u_{2p} happened to be 0.

All these examples shown here are over the rationals, but these functions also work over finite fields. The function `Ew2Ed_P4_eval` can also take a curve in complete Weierstrass form as an input, provided one has a point P of order 4. The full code for these two functions is at §B.1, functions 14 and 15, and more examples of their applications can be found in the online file `WC_EdC_birat_eq_BL.m`.

Since we are working with birational equivalences, addition should be preserved through them as long as we work within the function's domain. So we can give it a try and see what happens. For the addition in the Weierstrass model, we will use the built-in MAGMA addition. As for the one in the Edwards model we will use our own function, `Ead_AddL_eval`, which we will properly introduce in the next chapter since it can also be used for curves in twisted Edwards form. To use `Ead_AddL_eval` we also need `ProjCoord` and `AffCoord`, which are the functions to change the point coordinates from projective to affine and vice versa.

To show that addition is preserved in the $\mathcal{E}_d \to \mathcal{E}_w$ direction we will take Curve1174 from the SafeCurves collection, which is a curve over a prime order finite field. The two points `p_d`, `q_d` are on the curve and `pq_d = p_d + q_d`. Also we need a point `P4` of order 4 over the birationally equivalent curve in Weierstrass form. All this data is collected here:

```
> BF := FiniteField(2^251 - 9);
> R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
> Rd_var<x,y> := AffineSpace(R,2);
> Rw_var<u,v> := AffineSpace(R,2);
> Rw<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(BF,8);

> d := -1174; lst := [d];
> P4 := [1175, 36185027886661311069865932815214971204146870208012676\
   26233049500247285298889];
> p_d := [15826190977259115419545470064537397633810913888463948334922\
   96309729998839514, 30375380136041545047641157286514376465195135343O\
   52234227548270556891959925901;
> q_d := [25497503640559382086589595223914842196525899309317700357971 6\
   2350828388088291, 20974626828864940446672510259195016374189608062 05\
   09071752107886722442588196];
> PQ_d := Ead_AddL_eval(BF,[1,d], ProjCoord(Rw,p_d), ProjCoord(Rw,q_d));
> pq_d := AffCoord(BF,PQ_d); pq_d;

[ 21891505637633942422961588113811859011504113453541054350763343072387\
10824582, 18245582922659057195632353535646071914959389000854639199830 6\
08845290933594 ]
```

Now we have all we need to apply `Ed2Ew_P4_eval` to `p_d`, `q_d` and `pq_d`. This way we will get the list `lst_w` of parameters for a curve in Weierstrass form and the points `p_w`, `q_w` and `pq_w`. We can check whether these points are on the output curve:

```
> p_w := Ed2Ew_P4_eval(BF,lst,P4,p_d);
> q_w := Ed2Ew_P4_eval(BF,lst,P4,q_d)[2]; q_w := [BF|q_w[1],q_w[2]];
> pq_w := Ed2Ew_P4_eval(BF,lst,P4,pq_d)[2];pq_w := [BF|pq_w[1],pq_w[2]];

> lst_w := p_w[1]; p_w := [BF|p_w[2][1],p_w[2][2]];
> a1 := BF!lst_w[1]; a2 := BF!lst_w[2]; a3 := BF!lst_w[3];
> a4 := BF!lst_w[4]; a6 := BF!lst_w[5];
> Cw := EllipticCurve([BF|a1,a2,a3,a4,a6]);

> printf "\n lst_w = %o \n p_w = (%o,%o) \n q_w = (%o, %o) \n
  pq_w = (%o, %o)\n", lst_w, p_w[1], p_w[2], q_w[1], q_w[2],
  pq_w[1], pq_w[2];
```

```
lst_w = [ 0, 36185027886661311069865932815214971204146870208012676\
 26233049500247285298893, 0, 1380625, 0 ]
p_w = (34745638966469546211163154470295892346059316855318560549903\
 7344339048175889 7,77534652829123326267423017815958764318697652120\
 60214592526945118665084 3913)
q_w = (24687119919297300286254622219213650992904559945118412892428\
 07254924434774776, 70115884773787536539653718399941008884662634857\
 31156712802169814189987742 40)
pq_w = (27712550718996304134268722645311724235688527372233307130 15\
 44519372526749874 2, 15847385793825117409925789648529101141513113097961677587824\
 97961677587824495745789834514 67)

> printf "Are they really on the output curve? \n\t p_w -> %o, q_w -> %o,
    pq_w -> %o \n", p_w in Cw, q_w in Cw, pq_w in Cw;

Are they really on the output curve?
         p_w -> true, q_w -> true, pq_w -> true
```

So the last thing to test is if computing `r_w = p_w + q_w` using the Weierstrass addition law will yield `pq_w`:

```
> r_w := Cw!p_w + Cw!q_w;
> printf " r_w = (%o, %o) \n", r_w[1], r_w[2];

r_w = (27712550718996304134268722645311724235688527372233307130 1544519372\
  5267498742, 15847385793825117409925789648529101141513113097961677587824\
  49574578983451467)
```

And yes it does: `pq_w = r_w`! So the addition is preserved in this case.

Of course we also have examples on the other direction $\mathcal{E}_w \to \mathcal{E}_d$, as this one from a complete Weierstrass to Edwards over $\mathbb{Q}$:

```
> BF := Rationals();
> R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
> Rd_var<x,y> := AffineSpace(R,2);
> Rw_var<u,v> := AffineSpace(R,2);
> Rw<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(BF,8);

> lst := [BF| 27, -23321/132, 98, -1440071/1089, -2401 ]; u2p := 0;
> Cw := EllipticCurve(lst);
> lst := [a1,a2,a3,a4,a6]; P4 := [ -26/33, -1214/33 ];
> p_w := [ 144/25,-201087/1375 ];
> q_w := [ 30688841142001/43970426240400,
  -16489454909765872019419/291568775808617208000];
> pq_w := (Cw!p_w)+(Cw!q_w); pq_w := [pq_w[1], pq_w[2]]; pq_w;

[ 144/25, -147503/1375 ]
```

Applying the birational equivalence we obtain the curve parameter d and the points `p_d`, `q_d` and `pq_d`, which are supposedly on the curve. And so we can compute `r_d = p_d + q_d` and check if it holds that `pq_d = r_d`:

```
> p_d := Ew2Ed_P4_eval(BF,lst,P4,p_w); p_d := [p_d[2][1],p_d[2][2]];
> q_d := Ew2Ed_P4_eval(BF,lst,P4,q_w); q_d := [q_d[2][1],q_d[2][2]];
> pq_d := Ew2Ed_P4_eval(BF,lst,P4,pq_w); pq_d := [pq_d[2][1],pq_d[2][2]];

> d := p_d[1][1]; lst_d := [BF!1,d];
> Cd := Curve(Rd_var, x^2 +  y^2 - 1 - d*(x^2)*(y^2));
```

```
> printf "\n lst_d = [%o] \n p_d = (%o,%o) \n q_d = (%o, %o) \n pq_d =
    (%o, %o)\n", d, p_d[1], p_d[2], q_d[1], q_d[2], pq_d[1], pq_d[2];

lst_d = [59/33]
p_d = (1980/3349,2701/2051)
q_d = (-73468399352040/98315324391601, -65332207270801/3954524986799)
pq_d = (-1980/3349, 2701/2051)

> printf "Are they really on the output curve? \n\t p_d -> %o, q_d -> %o,
  pq_d -> %o \n", p_d in Cd, q_d in Cd, pq_d in Cd;

Are they really on the output curve?
         p_d -> true, q_d -> true, pq_d -> true

> r_d := Ead_AddL_eval(BF,lst_d, ProjCoord(Rw,p_d), ProjCoord(Rw,q_d));
> r_d := AffCoord(BF,r_d);
> printf " r_d = (%o, %o) \n", r_d[1], r_d[2];

 r_d = (-1980/3349, 2701/2051);
```

As it can be seen, `pq_d` is indeed equal to `r_d` and so in this case the addition is also preserved through the birational equivalence.

As well as these maps work, the requirement of a point of order 4 on the curve in Weierstrass form to make it all work does make things more complicated, since then we have to find a way of finding said point. The proof of proposition 1 in previous section does give us a way to do so if we already know the Weierstrass model parameters. Another option would be to compute the rational points of the curve in Weierstrass form and hope that we come across at least one with order 4. But what if we only have the parameters of a curve on Edwards form and we need a birationally equivalent curve in Weierstrass form?

Here is where the maps provided by Dam in [16] come into play, i.e., the ψ maps. We have implemented these maps in the functions `Ew2Ed_eval` and `Ed2Ew_eval`, included in the appendix at §B.1, functions 16 and 17 resp.. And we have done analogous work to the one just shown here for the ϕ maps but with these functions that don't need `P4` as input. Needless to say that the output of both maps match in all the examples we have tried.

Furthermore, we have used `Ed2Ew_eval` to help us with examples like the Curve1174 one shown before, where we needed a `P4` to do anything. The way that works is applying `Ed2Ew_eval` for any given d for and the point $(1, 0)$ and as output one gets the parameters of the birationally equivalent curve in $\bar{\mathcal{E}_w}$ form (unless specified otherwise) and a point of order 4 in said curve:

```
> d := -1174; lst := [d];
> ch := Ed2Ew_eval(BF,lst,[1,0]);

> lst_w := [BF!ch[1][i] : i in [1..5]]; "lst = ", lst_w;
> P4 := [BF| ch[2][1], ch[2][2]]; "P4 = ", P4;

lst =  [ 0, 36185027886661311069865932815214971204146870208012676262\
33049500247285298893, 0, 1380625, 0 ]
P4 =  [ 1175, 36185027886661311069865932815214971204146870208012676262\
233049500247285298889 ]

> Cw := EllipticCurve(lst_w);
> "4*P4 = ", 4*Cw!P4;

4*P4 =  (0 : 1 : 0)
```

Chapter 3

Twisted Edwards model

Building on top of what we have showed in the previous chapter, we will introduce now the twisted Edwards model for elliptic curves. This model comes as a further generalization of Edward's normal form, as it includes more curves over finite fields as compared to the Edwards model presented in the previous section.

The twisted Edwards model for elliptic curves is first presented in [3] and it is defined as follows:

Definition 15. Let K be a field such that $\text{char}(K) \neq 2$ and $\mathcal{C}$ an elliptic curve over K, then its twisted Edwards form is $\mathcal{E}_{a,d}\colon ax^2 + y^2 = 1 + dx^2y^2$, where $a, d \in K\backslash\{0\},\ a \neq d$.

Note that in the case $a = 1$ this model is the same as the Edwards model.

In this chapter we will study the addition law for this form of elliptic curves, including its geometric interpretation and its exceptional points. We will also dedicate a section to show its relation to the Edwards and the Weierstrass models. In [3], the correspondence between the twisted Edwards and the Montgomery models is also thoroughly studied, but this will not be included here.

3.1 Addition law

As it is to be expected, the addition law for elliptic curves on twisted Edwards form looks quite similar to the one for the Edwards model. More specifically, the addition law in this case is the following:

Let K be a field such that $char(K) \neq 2$, $\mathcal{C}$ an elliptic curve over K and (x_1, y_1), (x_2, y_2) two points on $\mathcal{C}$. Fix a, d to be such that $a, d \in K\backslash\{0\},\ a \neq d$. Then the addition law of $\mathcal{C}$ on its twisted Edwards form $\mathcal{E}_{a,d}$ is as follows:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + dx_1 x_2 y_1 y_2}, \quad \frac{y_1 y_2 - ax_1 x_2}{1 - dx_1 x_2 y_1 y_2} \right)$$

The neutral element for this addition law is $(0, 1)$ and the inverse of a point $P = (x, y)$ is $-P = (-x, y)$. As it is explained in [[3],§6], the correctness of this addition law is given by the fact that it coincides with the Edwards addition law for $\mathcal{E}_d\colon \bar{x}^2 + y^2 = 1 + (d/a)\bar{x}^2y^2$ with parameter $\bar{d} = d/a$ amd variables $(\bar{x}, y) = (\sqrt{a}x, y)$. This relation with the Edwards addition law also tells us that this is a complete addition law if a is a square in K and d/a is a non-square in K, i.e., this addition law is complete if a is a square in K and d is not a square in K.

3.1.1 Geometric interpretation

Unlike in the case of Weierstrass curves, the geometric interpretation of the addition law of both the Edwards curve and the twisted Edwards curve is not a very well known fact. But there are still some papers that study the topic. T. Hales explains it in [22] as a generalization of the hyperbolic addition on the circle to the hyperbolic addition on the curve defined by $\mathcal{E}_{a,d}$. A different motivation for this addition law can be found in [1], where they build a conic that intersects $\mathcal{E}_{a,d}$ to find the sum of two points. This approach is the one we will go into some detail here.

For starters, let's consider the embedding into $\mathbb{P}^2$ of the curve defined over the field K by $\mathcal{E}_{a,d}$ via the usual projection map $(x, y) \to (X/Z, Y/Z)$, which is $E\colon \left(aX^2 + Y^2\right)Z^2 = Z^4 + dX^2Y^2$. Then the

neutral element of the addition law is $\mathcal{O} = (0 : 1 : 1)$. Let's also consider the point $\mathcal{O}' = (0 : -1 : 1)$ and let $P_1, P_2 \in \mathcal{E}_{a,d}(K)$ be two points on E. Last but not least, we have the two points at infinity $\Omega_1 = (1 : 0 : 0)$, $\Omega_2 = (0 : 1 : 0)$.

We can now define the conic $\mathcal{C}$ as the conic passing through $\Omega_1, \Omega_2, \mathcal{O}', P_1$ and P_2. By [Thm.1, [1]], we know that this conic has the following form, where $\alpha, \beta, \gamma \in \mathbb{P}^2(K)$ are uniquely determined up to scalars:

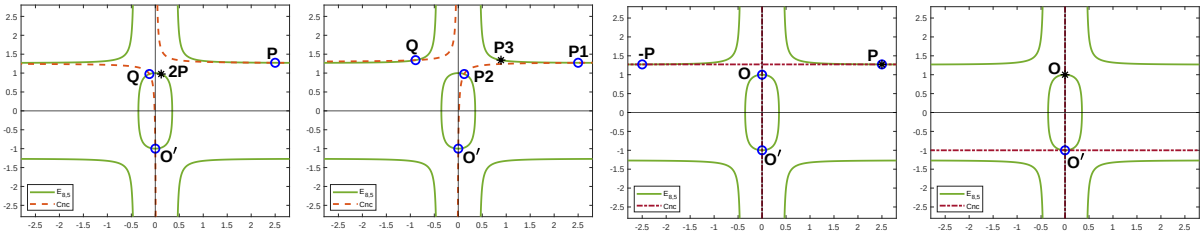
$$\mathcal{C}\colon \alpha\big(Z^2 + YZ\big) + \beta XY + \gamma XZ = 0$$

The coefficients α, β, γ are different depending on P_1 and P_2. The explicit formulae can be found in [1] and in our MAGMA implementation, which will be presented in Section 3.3.1.

Bezóut's theorem tells us that the intersection of $\mathcal{C}$ and E has 8 points in $\overline{K}$, counting multiplicities. Note that $\Omega_1, \Omega_2, \mathcal{O}' \in \mathcal{E}_{a,d}(K)$ and also that Ω_1 and Ω_2 are singular points of multiplicity 2. So the intersection of $\mathcal{C}$ and E is the set $\{P_1, P_2, \Omega_1, \Omega_2, \mathcal{O}', Q\}$, where Q is the eighth intersection point. From the proof of Theorem 2 in [1] we know that, if $P_3 = P_1 + P_2$, then $Q = -P_3$. Note that $Q \in \mathcal{C}$ and $Q, P_3 \in \mathcal{E}_{a,d}$ but $P_3 \notin \mathcal{C}$.

To sum up, the sum of two points $P_1, P_2 \in \mathcal{E}_{a,d}(K)$ is the inverse of the eighth intersection point of E and the conic $\mathcal{C}$ going through $\Omega_1, \Omega_2, \mathcal{O}', P_1$ and P_2.

In order to help visualize this geometric interpretation, we include some plots below. The curve used has parameters $a = 8$, $d = 5$ and we have $P = P_1 = (5/2, 14/11)$ as our base point. An interesting thing to be observed here is how $P_3 = -Q$ is not on the conic unless $P_3 = -P_3$ or $P_3 = P_1$. Also we can see that there is a different conic for each pair of points to be added, and in some cases it turns out to be a degenerate conic as it happens here in Figure (c) and Figure (d). Figure (a) shows the doubling of a point and Figure (b) is the addition of two different points. Figure (c) illustrates that $\mathcal{O}$ is indeed the neutral element of the addition law. And Figure (d) shows how $\mathcal{O}'$ is of order 2.



(a) $2P \approx (0.12, 0.97)$ (b) $P_1 + P_2 \approx (0.89, 1.34)$ (c) $P + \mathcal{O} = P$ (d) $2\mathcal{O}' = \mathcal{O}$

3.1.2 Complete system of addition laws

The addition law explained here is indeed complete but only over the field K and under the condition of a being a square and d being non-square in K. Since our aim is to be able to work in the most general case possible, we will present here a complete system of addition laws for twisted Edwards curves that Bernstein and Lange introduce in [6]. These formulae will be presented for the twisted Edwards curve model but they can naturally be used for Edwards curves as well by setting $a = 1$.

The system of addition laws consists of the addition law shown at the beginning of this section (referred to as "original addition law" in [6]) and the addition law shown in [24] by Hisin et al. (or "dual addition law"). This dual addition law is obtained by rewriting the original addition law so it does not depend on d and the result is the following:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3) = \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2}\right)$$

The outputs of this addition law are the same as the ones from the original law but this dual addition law has different exceptional cases. Said exceptional cases are stated in Theorem 2.2 of [24]. We will also study the exceptional cases for both addition laws below.

The complete system of addition laws is presented for the projective closure of the curve $\mathcal{C}$ defined by $\mathcal{E}_{a,d}$ in $\mathbb{P}^1 \times \mathbb{P}^1$, which is:

$$\bar{E} = \{((X:Z),(Y:T)) \in \mathbb{P}^1 \times \mathbb{P}^1 \mid aX^2T^2 + Y^2Z^2 = Z^2T^2 + dX^2Y^2\}$$

The embedding map is $(x, y) \mapsto ((x : 1), (y : 1))$ and the points at infinity are those such that $(X : Z) = (1 : 0)$ or $(Y : T) = (1 : 0)$. With this setup, the curve has 4 points at infinity: $((X : Z), (Y : T)) = ((1 : 0), (\pm\sqrt{a/d} : 1))$, minimally defined over $K(\sqrt{a/d})$, and $((X : Z), (Y : T)) = ((1 : \pm\sqrt{d}), (1 : 0))$, defined at least over $K(\sqrt{d})$.

While explaining the geometric interpretation of the original addition law, we have also shown the embedding E of $\mathcal{C}$ into $\mathbb{P}^2$. The rational map $((X : Z), (Y : T)) \mapsto (XT : YZ : TZ)$ from $\mathbb{P}^1 \times \mathbb{P}^1$ to $\mathbb{P}^2$ maps $\bar{E}$ onto E, both points $((1 : 0), (\pm\sqrt{a/d} : 1))$ to Ω_1 and both points $((\pm\sqrt{d} : 1), (1 : 0))$ to Ω_2.

Finally, the complete system of addition laws is:

Let K be a field such that char$\neq 2$ and fix a, d to be such that $d \in K\backslash\{0\}$, $a \neq d$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\mathcal{E}_{a,d}$ and $\bar{E}$ its projective closure in $\mathbb{P}^1 \times \mathbb{P}^1$. Also let $P_1 = ((X_1 : Z_1), (Y_1 : T_1))$, $P_2 = ((X_2 : Z_2), (Y_2 : T_2))$ be two points on the curve. Then the complete set of addition laws of $\mathcal{C}$ is:

$$P_1 + P_2 = \begin{cases} P_3 &= ((X_3 : Z_3), (Y_3 : T_3)) \\ P'_3 &= ((X'_3 : Z'_3), (Y'_3 : T'_3)) \end{cases} =$$

$$= \begin{cases} ((X_1Y_2Z_2T_1 + X_2Y_1Z_1T_2 : Z_1Z_2T_1T_2 + dX_1X_2Y_1Y_2), \\ \quad (Y_1Y_2Z_1Z_2 - aX_1X_2T_1T_2 : Z_1Z_2T_1T_2 - dX_1X_2Y_1Y_2)), \text{ if defined} \\ ((X_1Y_1Z_2T_2 + X_2Y_2Z_1T_1 : aX_1X_2T_1T_2 + Y_1Y_2Z_1Z_2), \\ \quad (X_1Y_1Z_2T_2 - X_2Y_2Z_1T_1 : X_1Y_2Z_2T_1 - X_2Y_1Z_1T_2)), \text{ if defined} \end{cases}$$

Theorem 6.1 in [6] states that this system is indeed complete since in all cases at least one of these holds:

- ▷ $(X_3, Z_3) \neq (0, 0)$ and $(Y_3, T_3) \neq (0, 0)$.
- ▷ $(X'_3, Z'_3) \neq (0, 0)$ and $(Y'_3, T'_3) \neq (0, 0)$.

Moreover, in case both addition laws do produce a non-zero output, then the outputs are the same since $X_3Z'_3 = X'_3Z_3$ and $Y_3T'_3 = Y'_3T_3$. The fact that P_3 and P'_3 are also in the curve is proven in Theorem 6.2 of the same paper.

Exceptional points

Each addition law in the system shown above has its exceptional points. Some of these are shown in [24] for the affine coordinates. In Section 8 of [6] one can also find an explanation on how to find these points and Theorem 8.1 in that same section gives their explicit projective coordinates. Nonetheless, we have also studied these exceptional points using a different approach to find them. And so we'll show it here.

Firstly, let's recall what a pair of exceptional points is: a pair of points $P_1 = ((X_1 : Z_1), (Y_1 : T_1))$ and $P_2 = ((X_2 : Z_2), (Y_2 : T_2))$ is considered exceptional for an addition law if the result of adding them with said addition law is a set of coordinates that is not valid. Since we are dealing with points in $\mathbb{P}^1 \times \mathbb{P}^1$, the kind of coordinates that can't happen are those where $X = Z = 0$ and/or $Y = T = 0$.

With this in mind, we are going to build systems of polynomials for each pair (X_3, Z_3), (Y_3, T_3), (X'_3, Z'_3) and (Y'_3, T'_3) such that their solutions give us all the exceptional points. The most direct way to do is to just equal to zero the corresponding polynomials of each pair. The resulting systems of equations will be clearly underdetermined. But taking into account some considerations, we can still make it work:

1) We are actually working with projective coordinates, so assuming none of the variables is zero we can rewrite the system as follows: $\alpha_i = X_i/Z_i$, $\beta_i = Y_i/T_i$, $i \in \{1, 2\}$. The cases of one or more variables being zero can be studied apart.

2) Solutions where there are some zero variables can only be such that only 1 coordinate of each point is zero. Otherwise the point would either not be on the curve or not have valid coordinates.

3) We don't need explicit solutions for each variable: finding relations between P_1 and P_2 will already be enough for us to see what the exceptional points are.

Let's see how the process goes for (X_3, Z_3). The system of equations to solve will be the following:

$$\begin{cases} X_1Y_2Z_2T_1 + X_2Y_1Z_1T_2 = 0 \\ Z_1Z_2T_1T_2 + dX_1X_2Y_1Y_2 = 0 \end{cases}$$

Let's assume all the variables are non-zero. Then using the change of variables given in point 1 above, the system looks like this:

$$\begin{cases} \alpha_1\beta_2 = -\alpha_2\beta_1 \\ d\alpha_1\alpha_2\beta_1\beta_2 = -1 \end{cases}$$

As explained in point 3, we are looking for a relation between P_1 and P_2 so let's assume we know P_1 and try to solve the system for P_2. Thus we have a system of 2 equations and 2 unknowns - α_2 and β_2 - which is a fully determined system with solutions:

$$\alpha_2 = \frac{1}{\pm\sqrt{d}\beta_1},\ \beta_2 = \frac{1}{\mp\sqrt{d}\alpha_1} \ \Rightarrow\ \frac{X_2}{Z_2} = \pm\frac{T_1}{\sqrt{d}Y_1},\ \frac{Y_2}{T_2} = \mp\frac{Z_1}{\sqrt{d}X_1} \ \Rightarrow\ P_2 = \big((T_1 : \pm\sqrt{d}Y_1), (Z_1 : \mp\sqrt{d}X_1)\big)$$

Now let's study the cases where one or more variables are zero:

By point 2 above, we know that at most only one coordinate of each point can be zero. And by looking at the addition law polynomials, we see that if one variable is zero in any polynomial, then the polynomial turns into a monomial and it can only be zero if yet another variable is zero. Thus we can conclude that all the solutions we will find in this case will be of the form $u_i = v_j = 0$ such that $i \neq j$.

So evaluating the polynomials at $X_1 = 0, \ldots, T_1 = 0$ we find the rest of the solutions:

$$X_i = T_j = 0,\ Y_i = Z_j = 0 \ \text{ such that } i, j \in \{1, 2\},\ i \neq j$$

But let's see, for example, how a pair of points such that $Y_1 = Z_2 = 0$ looks like. We know that both P_1 and P_2 are points on the curve so:

$$Y_1 = 0 \ \Rightarrow\ aX_1^2T_1^2 = Z_1^2T_1^2 \ \Rightarrow\ Z_1 = \pm\sqrt{a}X_1 \ \Rightarrow\ P_1 = \big((1 : \pm\sqrt{a}), (0 : 1)\big)$$

$$Z_2 = 0 \ \Rightarrow\ aX_2^2T_2^2 = dX_2^2Y_2^2 \ \Rightarrow\ Y_2 = \pm\sqrt{a/d}T_2 \ \Rightarrow\ P_2 = \big((1 : 0), (\pm\sqrt{a} : \mp\sqrt{d})\big)$$

From this we can see that this is just a specific case of the solution to the system shown above. The same happens to the rest of these extra solutions.

Thus we have all the exceptional points for the pair $\big(X_3, Z_3\big)$. And applying a similar process to the other three pairs, we get the exceptional points for each pair:

Let K be a field such that char$\neq 2$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\mathcal{E}_{a,d}$ and $\bar{E}$ its projective closure in $\mathbb{P}^1 \times \mathbb{P}^1$. Also let $P_1 = ((X_1 : Z_1), (Y_1 : T_1))$, $P_2 = ((X_2 : Z_2), (Y_2 : T_2))$ be two points on the curve and $P_3 = ((X_3 : Z_3), (Y_3 : T_3))$, $P_3' = ((X_3' : Z_3'), (Y_3' : T_3'))$ be as defined above by the addition law system. The exceptional points for said system are the following:

$$\hookrightarrow (X_3, Z_3) = (0, 0) \iff P_2 = \big((T_1 : \pm\sqrt{d}\,Y_1), (Z_1 : \mp\sqrt{d}\,X_1)\big).$$

$$\hookrightarrow (Y_3, T_3) = (0, 0) \iff P_2 = \big((Z_1 : \pm\sqrt{ad}\,X_1), (\pm\sqrt{a}\,T_1 : \sqrt{d}\,Y_1)\big).$$

$$\hookrightarrow (X_3', Z_3') = (0, 0) \iff P_2 = \big((Y_1 : \pm\sqrt{a}\,T_1), (\mp\sqrt{a}\,X_1 : Z_1)\big).$$

$$\hookrightarrow\ (Y_3', T_3') = (0, 0) \iff P_2 = \big(\big(\pm X_1 : Z_1\big), \big(\pm Y_1 : T_1\big)\big).$$

Note that for the original law, the exceptional pairs of points either include some square of d or they include an infinity point. Thus it holds that the original addition law in its affine form is complete if d is not a square in K.

Do these exceptional points also match the others shown in our references? Theorem 2.2 in [24] shows the exceptional points for the dual addition law with affine coordinates. Using the embedding map onto $\mathbb{P}^1 \times \mathbb{P}^1$, we see that two of these exceptional points correspond to ours and the other two are infinity points, so they are not exceptional in the projective closure we work with.

And in the case of [6], we can see that our results also match theirs. They do provide us with an interesting way of relating P_1 and P_2, given $P_1 = ((X_1 : Z_1), (Y_1 : T_1))$:

$$\hookrightarrow\ P_2 = \big(\big(T_1 : \pm\sqrt{d}Y_1\big), \big(Z_1 : \mp\sqrt{d}X_1\big)\big) \iff P_2 - P_1 = \big(\big(\pm\sqrt{d} : 1\big), \big(1 : 0\big)\big).$$

$$\hookrightarrow\ P_2 = \big(\big(Z_1 : \pm\sqrt{ad}X_1\big), \big(\pm\sqrt{a}T_1 : \sqrt{d}Y_1\big)\big) \iff P_2 - P_1 = \big(\big(1 : 0\big), \big(\pm\sqrt{a/d} : 1\big)\big).$$

$$\hookrightarrow\ P_2 = \big(\big(Y_1 : \pm\sqrt{a}T_1\big), \big(\mp\sqrt{a}X_1 : Z_1\big)\big) \iff P_2 - P_1 = \big(\big(1 : \pm\sqrt{a}\big), \big(0 : 1\big)\big).$$

$$\hookrightarrow\ P_2 = \big(\big(\pm X_1 : Z_1\big), \big(\pm Y_1 : T_1\big)\big) \iff P_2 - P_1 = \big(\big(0 : 1\big), \big(\pm 1 : 1\big)\big).$$

Now, looking at all these results, one could wonder if there is any overlap between the exceptional points of each pair, i.e., if it is possible that the four coordinates of either the original or the dual addition law are zero. Among the several ways one could check that, we have chosen to work with Gröbner bases.

This means that for each point $P = ((X : Z), (Y : T)) \in \{((X_3 : Z_3), (Y_3 : T_3)), ((X_3' : Z_3'), (Y_3' : T_3'))\}$ we are going to define the ideal I generated by the polynomials that define X, Y, Z and T. Then we are going to compute the Gröbner basis $GB = \{g_1, \ldots, g_n\}$ of said ideal so we have that $I = \langle g_1, \ldots, g_n\rangle$ and solve the system $g_1 = 0, \ldots, g_n = 0$. By construction we know that any point $Q \in \mathbb{P}^1 \times \mathbb{P}^1$ will be a solution to the system if and only if $X(Q) = Y(Q) = Z(Q) = T(Q) = 0$. So we can find the exceptional points this way.

For example, let's consider the original addition law. The Gröbner basis of the ideal generated by X_3, Z_3, Y_3 and T_3 is a set of 15 polynomials and the solutions to the system defined by them are the following sets of variables, when they are zero:

$$\{X_1, Z_1\}, \{Y_1, T_1\}, \{X_2, Z_2\}, \{Y_2, T_2\}, \{X_1, Y_1, Z_2\}, \{X_1, Y_1, T_2\}, \{X_1, Z_2, T_2\}, \{Y_1, Z_2, T_2\}$$

But point 2 above tells us that none of these can happen so we can conclude that there are no pairs of exceptional points for the whole original addition law. After doing similar checks for the dual addition law and the whole system of addition laws, we have also found no solutions. Thus here we have another way of checking that the system of addition laws is indeed complete.

3.2 Relation to other models

We have previously introduced the Weierstrass model and the Edwards model for elliptic curves. And in this section we are going to study how do they relate to the twisted Edwards model.

3.2.1 Relation to Edwards model

One way of looking at the twisted Edwards model is as a generalization of the Edwards model. In this sense, any curve in $\mathcal{E}_d$ form is also in $\mathcal{E}_{a,d}$ form for $a = 1$. And any curve in form $\mathcal{E}_{a,d}$ can also be rewritten following the Edwards model $\mathcal{E}_d$ with parameter $\bar{d} = d/a$.

But, as its name suggests, an elliptic curve in twisted Edwards form can also be seen as a quadratic twist of another elliptic curve in Edwards form. The map between $\mathcal{E}_d$ and $\mathcal{E}_{a,d}$ is an isomorphism over $K(\sqrt{a})$. Note that if a is a square in K, then $K(\sqrt{a}) = K$ and in that case $\mathcal{E}_d$ is isomorphic to $\mathcal{E}_{a,d}$ over K. The map is the following:

$$\mathcal{E}_d \longrightarrow \mathcal{E}_{a,d} \qquad\qquad \mathcal{E}_{a,d} \longrightarrow \mathcal{E}_d$$

$$\bar{d} \rightsquigarrow \ a, d = a\bar{d} \qquad\qquad a,\ d \rightsquigarrow \ \bar{d} = \frac{d}{a}$$

$$(\bar{x}, \bar{y}) \longmapsto (x, y) = \left(\frac{\bar{x}}{\sqrt{a}}, \bar{y}\right) \qquad\qquad (x, y) \longmapsto (\bar{x}, \bar{y}) = \left(\sqrt{a}x, y\right)$$

3.2.2 Relation to Weierstrass model: birational equivalence

In the previous chapter about the Edwards model we have studied the relation between the Edwards and the Weierstrass models. Knowing that the curves in twisted Edwards form are quadratic twists of the ones in Edwards form gives us a very obvious way to connect the twisted Edwards model to the Weierstrass one.

And said connection is the following: since $\mathcal{E}_{a,d}$ is isomorphic over $K(\sqrt{a})$ to $\mathcal{E}_d$ and $\mathcal{E}_w$ is birationally equivalent to $\mathcal{E}_{a,d}$, then we know that $\mathcal{E}_{a,d}$ is birationally equivalent to $\mathcal{E}_w$ over $K(\sqrt{a})$.

Keeping in mind that we have two birational equivalences from the previous chapter, the ϕ and the ψ maps, then we also have two in this case: Φ and Ψ, respectively. Following the same notation as before for $\bar{\mathcal{E}}_w$ to be a curve in Weierstrass form such that $a_1 = a_3 = a_6 = 0$. we have the following:

Let $P = (u_p, v_p) \in \mathcal{E}_w(K)$ be a point of order 4 and $d = a\big(1 - 4u_p^3/v_p^2\big)$. Then the following maps establish a birational equivalence between $\mathcal{E}_w$ and $\mathcal{E}_{a,d}$:

$$\Phi\colon \bar{\mathcal{E}}_w \longrightarrow \mathcal{E}_{a,d} \qquad\qquad \Phi^{-1}\colon \mathcal{E}_{a,d} \longrightarrow \bar{\mathcal{E}}_w$$

$$\bar{a}_2, \bar{a}_4,\ P \rightsquigarrow \ d = a\left(1 - \frac{4u_p^3}{v_p^2}\right) \qquad\qquad a,\ d \rightsquigarrow \ \bar{a}_2 = 2u_p\frac{a+d}{a-d},\ \bar{a}_4 = u_p^2$$

$$(\bar{u}, \bar{v}) \longmapsto (x, y) = \left(\frac{v_p\bar{u}}{\sqrt{a}u_p\bar{v}}, \frac{\bar{u} - u_p}{\bar{u} + u_p}\right) \qquad\qquad (x, y) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{u_p(1+y)}{1-y}, \frac{v_p(1+y)}{\sqrt{a}x(1-y)}\right)$$

Let $d \in K\backslash\{0, 1\}$ and $A = 2y - (2(d/a)y + (d/a) + 1)\sqrt{a}x^2 + 2$. Then the following maps are well-defined:

$$\Psi\colon \bar{\mathcal{E}}_w \longrightarrow \mathcal{E}_{a,d} \qquad\qquad \Psi^{-1}\colon \mathcal{E}_{a,d} \longrightarrow \bar{\mathcal{E}}_w$$

$$\bar{a}_2, \bar{a}_4 \rightsquigarrow \ a,\ d = a\left(\frac{\bar{a}_2}{2} - 1\right) \qquad\qquad a,\ d \rightsquigarrow \ \bar{a}_2 = 2\left(\frac{d}{a} + 1\right),\ \bar{a}_4 = \left(\frac{d}{a} - 1\right)^2$$

$$(\bar{u}, \bar{v}) \longmapsto (x, y) = \left(\frac{-2\bar{u}}{\sqrt{a}\bar{v}}, \frac{\bar{v}^2 - \bar{a}_2\bar{u}^2 - 2\bar{u}^3}{2(\bar{a}_2 - 2)\bar{u}^2 - \bar{v}^2}\right) \qquad\qquad (x, y) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{A}{ax^2}, \frac{-2A}{a\sqrt{a}x^3}\right)$$

In the computational section below we'll show a few examples of the fact that these birational equivalences do preserve addition between the models.

3.2.3 Relation to Weierstrass model: 2-isogeny

There is of course more than one way two models can relate. And so, in spite of already having shown a birational equivalence between the Weierstrass and the twisted Edwards models, in this section we are going to explore how are they isogenous.

For such purpose we will show the work of Bernstein et al in [3]. They choose to study the relation between the twisted Edwards model and the Weierstrass model using the connection of these models to the elliptic curves in Montgomery form. And the result is an isogeny between the models, as it is stated in [Thm. 5.1, [3]] and here:

Theorem 2. Let K be a field such that $\mathrm{char}(K) \neq 2$. Then every elliptic curve over K in $\mathcal{E}_w$ form and with three K-rational points of order 2 is 2-isogenous over K to an elliptic curve in $\mathcal{E}_{a,d}$ form.

If the curve in $\mathcal{E}_w$ form does not have 3 rational points of order 2 in K, we can find an extension of the field where it does have such points.

The proof of this theorem gives us the recipe to build the isogenies of degree 2 between both models, which we will do here next. Part of the process includes the Montgomery model that we haven't introduced so far. There is however an overview of it in the next chapter, Section 4.1.

Firstly, let's consider the curve $\mathcal{C}$ with model $\mathcal{E}_w : v^2 = u'^3 + a_2u'^2 + a_4u' + a_6$ and let $(u'_0, 0)$, $(u'_1, 0)$, $(u'_2, 0)$ be its three K-rational points of order 2. Note that this means that u'_0, u'_1 and u'_2 are the three roots of the polynomial $u'^3 + a_2u'^2 + a_4u' + a_6$, so it factors as $(u' - u'_0)(u' - u'_1)(u' - u'_2)$. Consider now the change of coordinates $u' = u + u'_0$. Then the polynomial looks like this: $u(u - u_1)(u - u_2)$, where $u_1 = u'_1 - u'_0$ and $u_2 = u'_2 - u'_0$. Thus we can rewrite the model of the curve as follows:

$$E : v^2 = u^3 - (u_1 + u_2)u^2 + (u_1u_2)u$$

And there is a 2-isogeny from this form to the following one:

$$\bar{E} : \bar{v}^2 = \bar{u}^3 + 2(u_1 + u_2)\bar{u}^2 + (u_1 - u_2)^2\bar{u}$$

Said isogeny and its dual are given by:

$$E \longrightarrow \bar{E}$$
$$(u, v) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{v^2}{u^2}, \frac{v(u_1u_2 - u^2)}{u^2}\right)$$

$$\bar{E} \longrightarrow E$$
$$(\bar{u}, \bar{v}) \longmapsto (u, v) = \left(\frac{\bar{v}^2}{4\bar{u}^2}, \frac{\bar{v}((u_1 - u_2)^2 - \bar{u}^2)}{8\bar{u}^2}\right)$$

Next is the isomophism that exists between $\bar{E}$ and the Montgomery model $E_{M,A,B} : B\bar{y}^2 = \bar{x}^3 + A\bar{x}^2 + \bar{x}$:

$$\bar{E} \longrightarrow E_{M,A,B}$$
$$u_1, u_2 \rightsquigarrow A = \frac{2(u_1 + u_2)}{u_1 - u_2},\ B = \frac{1}{u_1 - u_2}$$
$$(\bar{u}, \bar{v}) \longmapsto (\bar{x}, \bar{y}) = (\bar{u}B, \bar{v}B) = \left(\frac{\bar{u}}{u_1 - u_2}, \frac{\bar{v}}{u_1 - u_2}\right)$$

$$E_{M,A,B} \longrightarrow \bar{E}$$
$$A, B \rightsquigarrow u_1 = \frac{A + 2}{4B},\ u_2 = \frac{A - 2}{4B}$$
$$(\bar{x}, \bar{y}) \longmapsto (\bar{u}, \bar{v}) = \left(\frac{\bar{x}}{B}, \frac{\bar{y}}{B}\right) = (\bar{x}(u_1 - u_2), \bar{y}(u_1 - u_2))$$

And the last step is the birational equivalence between $E_{M,A,B}$ and the curve in twisted Edwards form $\mathcal{E}_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$:

$$E_{M,A,B} \longrightarrow \mathcal{E}_{a,d}$$
$$A, B \rightsquigarrow a = \frac{A + 2}{B},\ B = \frac{A - 2}{B}$$
$$(\bar{x}, \bar{y}) \longmapsto (x, y) = \left(\frac{\bar{x}}{\bar{y}}, \frac{\bar{x} - 1}{\bar{x} + 1}\right)$$

$$\mathcal{E}_{a,d} \longrightarrow E_{M,A,B}$$
$$a, d \rightsquigarrow A = \frac{2(a + d)}{a - d},\ B = \frac{4}{a - d}$$
$$(x, y) \longmapsto (\bar{x}, \bar{y}) = \left(\frac{1 + y}{1 - y}, \frac{1 + y}{x(1 - y)}\right)$$

Now, putting all of this together we get the 2-isogenies between E and $\mathcal{E}_{a,d}$:

$$\varphi\colon E \longrightarrow \mathcal{E}_{a,d} \qquad\qquad \widehat{\varphi}\colon \mathcal{E}_{a,d} \longrightarrow E$$

$$u_1, u_2 \rightsquigarrow a = 4u_1,\ d = 4u_2 \qquad\qquad a, d \rightsquigarrow u_1 = \frac{a}{4},\ u_2 = \frac{d}{4}$$

$$(u,v) \longmapsto (x,y) = \left(\frac{v}{u_1u_2 - u^2}, \frac{v^2 - (u_1 - u_2)u^2}{v^2 + (u_1 - u_2)u^2}\right) \qquad (x,y) \longmapsto (u,v) = \left(\frac{1}{4x^2}, \frac{(a-d)y}{8x(y^2-1)}\right)$$

Note that $\varphi^{-1} \neq \widehat{\varphi}$ and, since they are 2-isogenies, it holds that $\varphi\big(\widehat{\varphi}(p)\big) = 2p$ and $\widehat{\varphi}\big(\varphi(q)\big) = 2q$ for any $p \in \mathcal{E}_{a,d}(K)$, $q \in \mathcal{E}_w(K)$.

Note as well that, if $u_0 = 0$, then both models E and $\mathcal{E}_w$ are the same, so the maps shown above would suffice. In case we are working with a curve in Weiertrass form such that $a_6 \neq 0$, then we will need to put it in E form, which can be easily done with the change of coordinates provided before, namely $(u', v) \mapsto (u, v) = (u' - u'_0, v)$. Regarding the coefficients, they are related as follows:

$$u_1 = u'_1 - u'_0; \quad u_2 = u'_2 - u'_0$$

$$a_2 = -(u'_0 + u'_1 + u'_2); \quad a_4 = u'_0u'_1 + u'_0u'_2 + u'_1u'_2; \quad a_6 = -u'_0u'_1u'_2$$

3.3 Implementations in Magma

Given all of this information about twisted Edwards curves, it is time to see some of it in practice. We will start by checking some facts about the geometric interpretation of the original addition law and then see how it works with some specific curves. Afterwards we'll do the same with the complete addition law system. We will also work with the exceptional points of each addition law.

Later on we'll see how the maps between the other models and this one work, illustrated as well with some numerical examples.

3.3.1 Geometric interpretation

Our objective when we thought of implementing the geometric interpretation in Magma was to manage to generate the addition law formulae. But computing the symbolic intersection of $\mathcal{E}_{a,d}$ and the conic $\mathcal{C}$ has proven to be rather challenging. Instead we have checked that all the points $P_1, P_2, \Omega_1, \Omega_2, \mathcal{O}', Q = -P_3$ are indeed in both the curve and the conic.

To do so, we have started by defining a few algebraic structures, as shown below. The basis field we will be using is the rationals as defined by `BF`, but it could also be any finite field. The computations we will do afterwards, however, won't work over the real or complex fields. The variables `X1,Y1,Z1` are the coordinates of P_1 and, similarly, `X2,Y2,Z2` are the ones for P_2. And `X,Y,Z` are reserved for general unknowns.

```
> BF := Rationals();
> R1<a,d> := FunctionField(BF,2);
> R2<X1,Y1,Z1,X2,Y2,Z2> := PolynomialRing(R1,6);
> R3<X,Y,Z> := PolynomialRing(R2,3);
```

In order to define P_1 and P_2 as points on the elliptic curve, we need to find a way of algebraically expressing the conditions that make P_1 and P_2 points on the curve: the curve polynomial evaluates to zero in both points. And so we create an ideal `I_crv` generated by the twisted Edwards model equation evaluated at both points. Then we can build the quotient ring of `R2` modulo `I_crv` and so, to apply the fact that P_1 and P_2 are points on the curve, it will suffice to coerce results into said quotient ring.

```
> f_crv1 := (a*(X1^2) + (Y1^2))*Z1^2 - Z1^4 - d*(X1^2)*(Y1^2);
> f_crv2 := (a*(X2^2) + (Y2^2))*Z2^2 - Z2^4 - d*(X2^2)*(Y2^2);
```

```
> I_crv := ideal<R2|f_crv1,f_crv2>;
> QI_crv := R2/I_crv;
```

We also need the result of the addition, i.e., P_3. Section 6 of [3] provides us with several parametrizations of the addition law in $\mathbb{P}_2$ and we will use one of them here:

```
> A := Z1*Z2; B := A^2; C := X1*X2; D := Y1*Y2;
> E := d*C*D; F := B-E; G := B+E;
> X3 := A*F*((X1+Y1)*(X2+Y2)-C-D); "X3 =", X3;
> Y3 := A*G*(D-a*C); Y3; "Y3 =", Y3;
> Z3 := F*G; Z3; "Z3 =", Z3;

X3 = -d*X1^2*Y1*Z1*X2*Y2^2*Z2 - d*X1*Y1^2*Z1*X2^2*Y2*Z2 +
    X1*Z1^3*Y2*Z2^3 + Y1*Z1^3*X2*Z2^3
Y3 = -a*d*X1^2*Y1*Z1*X2^2*Y2*Z2 + d*X1*Y1^2*Z1*X2*Y2^2*Z2 -
    a*X1*Z1^3*X2*Z2^3 + Y1*Z1^3*Y2*Z2^3
Z3 = -d^2*X1^2*Y1^2*X2^2*Y2^2 + Z1^4*Z2^4
```

The next step is to define the elliptic curve and the conic. The conic has 3 different sets of coefficients depending on P_1 and P_2. So we actually need to define three conics and we will use different suffixes to differentiate them: `"_PP"` is for the point doubling case, `"_POp"` is for adding to $\mathcal{O}'$ and `"_PQ"` for the rest.

```
> f_crv := (a*(X^2) + (Y^2))*Z^2 - Z^4 - d*(X^2)*(Y^2);

> cz_PP := X1*Z1*(Z1-Y1);
> cxy_PP := d*(X1^2)*Y1-Z1^3;
> cxz_PP := Z1*(Z1*Y1 - a*X1^2);
> f_cnc_PP := cz_PP*(Z^2 + Y*Z)+cxy_PP*X*Y+cxz_PP*X*Z;

> cz_POp := -X1; cxy_POp := Z1; cxz_POp := Z1;
> f_cnc_POp := cz_POp*(Z^2 + Y*Z)+cxy_POp*X*Y+cxz_POp*X*Z;

> cz_PQ := X1*X2*(Y1*Z2-Y2*Z1);
> cxy_PQ := Z1*Z2*(X1*Z2-X2*Z1 + X1*Y2 - X2*Y1);
> cxz_PQ := X2*Y2*(Z1^2)-X1*Y1*(Z2^2)+Y1*Y2*(X2*Z1-X1*Z2);
> f_cnc_PQ := cz_PQ*(Z^2 + Y*Z)+cxy_PQ*X*Y+cxz_PQ*X*Z;
```

Now we are all set up to check whether the intersection of the curve and the conic has all the points is supposed to have or not. By definition, we know that P_1 and P_2 are on the curve. Also it is trivial that $\mathcal{O}'$, Ω_1 and Ω_2 are on the curve as well. But we can easily check that with Magma as well by evaluating the curve polynomial and checking if it is indeed 0. Similarly, we can check that P_3 and $-P_3$ are also on the curve:

```
> printf "Op: %o; O1: %o; O2: %o; \n", Evaluate(f_crv,[0,-1,1]),
    Evaluate(f_crv,[1,0,0]), Evaluate(f_crv,[0,1,0]);
> printf "P3: %o; -P3: %o; \n", QI_crv!Evaluate(f_crv,[X3,Y3,Z3]),
    QI_crv!Evaluate(f_crv,[-X3,Y3,Z3]);

Op: 0; O1: 0; O2: 0;
P3: 0; -P3: 0;
```

Note that we coerce some evaluations into `QI_crv` when needed, that is, when they are not 0 straight away so we need to apply the fact that P_1 and P_2 are on the curve.

Now for the conic we can start by checking the case of adding to $\mathcal{O}'$ with the script below. Here `X3_POp`, `Y3_POp` and `Z3_POp` denote the result of the addition of P_1 and $\mathcal{O}'$:

```
> printf "Op: %o; O1: %o; O2: %o; \n", Evaluate(f_cnc_POp,[0,-1,1]),
    Evaluate(f_cnc_POp,[1,0,0]), Evaluate(f_cnc_POp,[0,1,0]);
```

```
> printf "P1: %o; -P3: %o; \n", QI_crv!Evaluate(f_cnc_POp,[X1,Y1,Z1]),
    Evaluate(f_cnc_POp,[-X3_POp,Y3_POp,Z3_POp]);
> printf "P3: %o; \n",QI_crv!Evaluate(f_cnc_POp,[X3_POp,Y3_POp,Z3_POp]);

Op: 0; O1: 0; O2: 0;
P1: 0; -P3: 0;
P3: 2*X1*Y1*Z1^7 - 2*X1*Z1^8;
```

Since the conic polynomial evaluates to 0 in $\mathcal{O}'$, Ω_1, Ω_2, P_1 and $-P_3$ we know that these points are on the conic as well as on the curve. Thus in this case we can see that the geometric interpretation holds. Note as well that, zero cases aside, the conic is not zero in P_3 so we see that in general P_3 is not on the conic.

In the general case these checks look like this:

```
> printf "Op: %o; O1: %o; O2: %o; \n", Evaluate(f_cnc_PQ,[0,-1,1]),
    Evaluate(f_cnc_PQ,[1,0,0]), Evaluate(f_cnc_PQ,[0,1,0]);
> printf "P1: %o; P2: %o; -P3: %o; \n", Evaluate(f_cnc_PQ,[X1,Y1,Z1]),
    Evaluate(f_cnc_PQ,[X2,Y2,Z2]), QI_crv!Evaluate(f_cnc_PQ,[-X3,Y3,Z3]);

Op: 0; O1: 0; O2: 0;
P1: 0; P2: 0; -P3: 0;
```

Since all evaluations yield 0, we know that in the general case all these six points are on the conic. Note that here we do evaluate the conic both in P_1 and P_2. Previously we didn't need to do this because P_2 in that case was $\mathcal{O}'$.

The checks for the point doubling case yield analogous results but we will not show them here although the online script TwEdC_AdL_geom.m does include them as well.

So now we have all the calculations that prove that $\mathcal{O}'$, Ω_1, Ω_2, P_1, P_2 and $-P_3$ are indeed in the intersection of $\mathcal{E}_{a,d}$ and the conic. And, in spite of not being able to compute the intersection itself symbolically, once we start working with specific examples we can indeed compute the intersection points and their multiplicities. Let's take for example the curve $\mathcal{C}$ given in $\mathcal{E}_{a,d}$ form for $a = 17$ and $d = 82$:

```
> a := 17; d := 82;
> Crv := Curve(R2, (a*(X^2) + (Y^2))*Z^2 - Z^4 - d*(X^2)*(Y^2)); Crv;

Curve over Multivariate rational function field of rank 2 over Rational
Field defined by -82*X^2*Y^2 + 17*X^2*Z^2 + Y^2*Z^2 - Z^4
```

We also have the points on the curve $P_1 = (1 : 4/9 : 1)$ and $P_2 = (-72/1393 : -1361/1231 : 1)$. With this data, we can define in MAGMA the conic passing through P_1, P_2, Ω_1, Ω_2 and $\mathcal{O}'$ with the corresponding coefficients:

```
> cz := X1*X2*(Y1*Z2-Y2*Z1);
> cxy := Z1*Z2*(X1*Z2-X2*Z1 + X1*Y2 - X2*Y1);
> cxz := X2*Y2*(Z1^2)-X1*Y1*(Z2^2)+Y1*Y2*(X2*Z1-X1*Z2);
> Cnc := Conic(R2,cz*(Z^2 + Y*Z)+cxy*X*Y+cxz*X*Z); Cnc;

Conic over Multivariate rational function field of rank 2 over Rational
Field defined by -53066/1714783*X*Y + 1998256/15433047*X*Z -
137384/1714783*Y*Z - 137384/1714783*Z^2
```

So now we can ask MAGMA to give us the points of the intersection of `Crv` and `Cnc`. We can also check the multiplicity of said points:

```
> pts := Points(Intersection(Crv,Cnc));
> pts_mult := [IntersectionNumber(Crv,Cnc,p) : p in pts];
> pts; pts_mult;
```

```
{@ (0 : -1 : 1), (-72/1393 : -1361/1231 : 1), (1 : 4/9 : 1),
(1935265/5286047 : -5977204/16708329 : 1), (0 : 1 : 0), (1 : 0 : 0) @}

[ 1, 1, 1, 1, 2, 2 ]
```

So here we can see that in the intersection we have indeed $\mathcal{O}'$, P_2, P_1 with multiplicity 1 and the infinity points with multiplicity 2. Besides there is a sixth point $Q = (1935265/5286047 : -5977204/16708329 : 1)$ with multiplicity 1, which is supposed to be $-P_3$.

Using the parametrization of the addition law shown in the symbolic computations, we can compute the addition of P_1 and P_2 and check that Q is indeed $-P_3$:

```
> x3 := Evaluate(X3,[X1,Y1,Z1,X2,Y2,Z2]);
> y3 := Evaluate(Y3,[X1,Y1,Z1,X2,Y2,Z2]);
> z3 := Evaluate(Z3,[X1,Y1,Z1,X2,Y2,Z2]);
> P3 := R2 ! [x3/z3, y3/z3, 1]; P3;

(-1935265/5286047 : -5977204/16708329 : 1)
```

An interesting fact to check is what happens with the multiplicities of the points in the intersection when there are less than 6 points in said intersection. This happens in the following scenarios:

↪ Adding a point to $\mathcal{O}'$: when $\mathcal{O}'$ is part of the addition, then its multiplicity doubles, as we can see in this example of the intersection points of $\mathcal{C}$ with the conic defined by the addition $P_2 + \mathcal{O}'$:

```
{@ (-72/1393 : -1361/1231 : 1), (0 : -1 : 1), (-72/1393 : 1361/1231 : 1),
(0 : 1 : 0), (1 : 0 : 0) @}

[ 1, 2, 1, 2, 2 ]
```

↪ Point doubling: if the point to be doubled is not $\mathcal{O}'$, then we get 5 points in the intersection and the point to be doubled has multiplicity 2. As for example the following situation, where we compute the conic defined by the addition $P_2 + P_2$:

```
{@ (0 : -1 : 1), (-336070031472/3727880166337 : 3460788697921/2153081307841 :
1), (-72/1393 : -1361/1231 : 1), (0 : 1 : 0), (1 : 0 : 0) @}

[ 1, 1, 2, 2, 2 ]
```

And if what we are computing is $2\mathcal{O}'$, then we get an intersection with only 4 points, where $\mathcal{O}'$ has multiplicity 3:

```
{@ (0 : -1 : 1), (0 : 1 : 1), (0 : 1 : 0), (1 : 0 : 0) @}

[ 3, 1, 2, 2 ]
```

↪ $Q = -P_3$ happens to be equal to either P_1, P_2 or $\mathcal{O}'$: looking at the other cases, one can already expect what will happen here: whichever point Q equals to will have its multiplicity raised by 1. And so it happens, as we can see in these two examples:

Intersection points of $\mathcal{C}$ with the conic defined by the addition of $P_2 = (x_2 : y_2 : z_2)$ and $(x_2 : y_2 : z_2)$, where the solution is $\mathcal{O}$:

```
{@ (-72/1393 : -1361/1231 : 1), (0 : -1 : 1), (-72/1393 : 1361/1231 : 1),
(0 : 1 : 0), (1 : 0 : 0) @}

[ 1, 2, 1, 2, 2 ]
```

Intersection points of $\mathcal{C}$ with the conic defined by the addition $P_2 + (-2P_2)$:

```
{@ (0 : -1 : 1), (-336070031472/3727880166337 : 3460788697921/2153081307841 :
1), (-72/1393 : -1361/1231 : 1), (0 : 1 : 0), (1 : 0 : 0) @}

[ 1, 1, 2, 2, 2 ]
```

↪ Infinity points are involved in the addition: the geometric interpretation we are studying here is for an addition law that has the infinity points as exceptional points. So as expected it doesn't work for any addition involving Ω_2. But surprisingly enough, it works for a specific case involving Ω_1, the one about the addition $\mathcal{O}' + \Omega_1$:

```
{@ (0 : -1 : 1), (0 : 1 : 0), (1 : 0 : 0) @}

[ 2, 2, 4 ]
```

Taking all of this into account, we have written a function `Ead_AddL_geom` that computes the addition of two points on a twisted Edwards elliptic curve in the projective space $\mathbb{P}_2$ using the geometric interpretation instead of the addition law formulae. The function can be found in the appendix section A.2, function 5. Naturally it has the limitations of this addition law and the ones derived from the fact that most of the MAGMA commands used in it only work over the rationals or a finite field. However, in the following section we present a function that can be more widely used.

3.3.2 Complete system of addition laws

The computational work we have done related to the system of addition laws begins with seeing how some theoretical facts actually hold. And afterwards, we developed a function to actually compute the addition of two points on a curve in twisted Edwards form.

We start by creating a function `Ead_AddL` that takes as input a list with the curve parameters a, d and outputs both addition laws for the given curve. So this way we can access the addition laws whenever we need them. This is function 3 in §A.1 and the checks we will show now can be found online in the file `TwEdC_AdL.m`.

Let's begin by checking that the outputs of both addition laws are the same, provided no exceptional points happen. In the same way as what we did in the previous section, we need to define the quotient ring with an equivalence relation such that $\alpha \sim 0 \iff \alpha \in (\mathcal{E}_{a,d})$:

```
> R1<a,d> := FunctionField(BF,2);
> R2<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(R1,8);
> P := [[X1,Z1],[Y1,T1]]; Q := [[X2,Z2],[Y2,T2]];

> Ead1 := Z1^2*T1^2 + d*X1^2*Y1^2 - a*X1^2*T1^2 - Y1^2*Z1^2;
> Ead2 := Z2^2*T2^2 + d*X2^2*Y2^2 - a*X2^2*T2^2 - Y2^2*Z2^2;
> I := ideal< R2 | Ead1, Ead2>;
> QQ<X1,Y1,Z1,T1,X2,Y2,Z2,T2> := quo<R2 | I>;
```

Then we get the original addition laws from `Ead_AddL`:

```
> X3_o := (R2 ! Ead_AddL([a,d])[1][1][1]);
> Y3_o := (R2 ! Ead_AddL([a,d])[1][2][1]);
> Z3_o := (R2 ! Ead_AddL([a,d])[1][1][2]);
> T3_o := (R2 ! Ead_AddL([a,d])[1][2][2]);
```

Similarly we store the dual addition law, but using the suffix `"_d"`. And so now it's easy to check that the outputs of both addition laws are the same:

```
> (QQ ! (X3_o*Z3_d)) eq (QQ ! (Z3_o*X3_d));

 true
```

```
> (QQ ! (Y3_o*T3_d)) eq (QQ ! (T3_o*Y3_d));

 true
```

Next we can check that the output of both addition laws is actually on the curve by evaluating the curve equation on the corresponding outputs and embedding the result in `QQ`. If the result is zero, then the output is on the curve:

```
> QQ ! Evaluate(Ead1,[X3_o,Y3_o,Z3_o,T3_o,1,1,1,1]);

 0

> QQ ! Evaluate(Ead1,[X3_d,Y3_d,Z3_d,T3_d,1,1,1,1]);

 0
```

The last check we want to do is if the system is indeed complete. That is done in the online script TwEdC_AdL_completeness.m, where we use Gröbner basis as explained at the end of Section 3.1.2.

So we study the original addition law, the dual addition law and then the whole system. The process is the same in all three cases so here we will only go through the first case: the original addition law. The first thing to do is generate the Gröbner basis of the ideal generated by the formulate of the addition law:

```
> II_o := ideal<R2 | X3_o,Y3_o,Z3_o,T3_o>;
> GB_o := GroebnerBasis(II_o); GB_o;

 [
     X1*Y1*Z1*Y2^3*Z2^2,
     X1*Y1*X2*Y2,
     X1*T1*X2*T2 - 1/a*Y1*Z1*Y2*Z2,
     X1*T1*Y2*Z2 + Y1*Z1*X2*T2,
     Y1^3*Z1^2*X2*Y2*Z2,
     Y1^3*Z1^2*X2*Y2*T2,
     Y1^2*Z1^3*X2*Z2*T2,
     Y1^2*Z1^3*Y2*Z2*T2,
     Y1^2*Z1*X2^2*T2,
     Y1^2*Z1*Y2^2*Z2,
     Y1*Z1^2*X2*T2^2,
     Y1*Z1^2*Y2*Z2^2,
     Y1*Z1*T1*Y2^2*Z2^3,
     Y1*Z1*X2^2*T2^2 + 1/a*Y1*Z1*Y2^2*Z2^2,
     Z1*T1*Z2*T2
 ]
```

By looking at this basis, we can see that the homogeneous system built from it will not be linear. And so we can't apply a Magma command to solve it. Instead, we have developed the function `bfz` that outputs the list of all solutions of the form $U = V = \cdots = 0$, computed by brute force. And the solutions we obtain are the following:

```
> bfz(GB_o);

 [ X1, Z1 ]
 [ Y1, T1 ]
 [ X2, Z2 ]
 [ Y2, T2 ]
 [ X1, Y1, Z2 ]
 [ X1, Y1, T2 ]
 [ X1, Z2, T2 ]
 [ Y1, Z2, T2 ]
```

```
[ Z1, T1, X2 ]
[ Z1, T1, Y2 ]
[ Z1, X2, Y2 ]
[ T1, X2, Y2 ]
```

The first four solutions imply a projective point of the kind $((0:0),(Y:T))$ or $((X:Z),(0:0))$, which are not points that can happen, so they are not valid solutions. And all the tuples of 3 elements, in different ways, end up implying also a situation like $X = Z = 0$ or $Y = T = 0$ as well. For example, the first of these kind of solutions for the original law is `[ X1, Y1, Z2 ]`. Looking at the projective equation of the curve, if `X1 = Y1 = 0`, then for the point to be on the curve we also need `Z1 = 0` or `T1 = 0`. But that can't happen. So none of the solutions on the list is a valid solution.

The question now is if there are any other solutions that are not included in this list. Suppose there is an extra solution that is not listed above, which had been compiled by looking for what variables equal zero are solutions. So this solution would have to include some variables equal to something other than zero for it not to be on the list.

At the same time, this new solution should make all the elements of the Gröbner basis equal zero, including the monomial ones. Then at least one of the variables should be equal to zero. Let's assume such variable is `V` and so we set `V = 0`. Let's consider now the basis element `X1*T1*X2*T2 - 1/a*Y1*Z1*Y2*Z2`. Evaluating it for `V = 0` gives us one of the following equations to solve: `1/a*Y1*Z1*Y2*Z2 = 0` or `X1*T1*X2*T2 = 0`. But in either case, we have again the case of a monomial equal to zero so the solution for it is another variable equal to zero. Thus the solution is already included in the list above.

The choice of variable for this example does not matter since the polynomial `X1*T1*X2*T2 - 1/a*Y1*Z1*Y2*Z2` actually has all variables we are working with. So we conclude that there are no other solutions aside from the ones already found. And since none of the found solutions is actually a valid one, then we know that the original addition law never evaluates to $((0:0),(0:0))$.

For the other two cases, namely the dual addition law and the whole system, we get an analogous situation. Thus we can conclude two more things: the dual addition law never evaluates to $((0:0),(0:0))$ either and there are no two points P_1, P_2 such that the eight polynomials of the whole addition law system evaluate to zero simultaneously. Therefore the system of addition laws is indeed complete.

Now that we have seen how the system is indeed complete and well-defined, we will build a function to compute the addition of any two given points on an elliptic curve following the twisted Edwards model. Because the function `Ead_AddL` is useful when we want to see how both addition laws for a given curve look like. But if what we want to do is to compute the addition of two points, we need a function that does a few more things. And that's why we have `Ead_AddL_eval` [§A.3,Fn.7], a function that takes as input a basis field `BF`, a list `lst` with the curve parameters a and d and two points `P`, `Q` on the curve over $\mathbb{P}^1 \times \mathbb{P}^1$.

Similarly to its counterpart for the Weierstrass model, this function is also written to be used as widely as possible. So once again we need to include these lines of code in case `FunctionField` is not supported over our input base field:

```
if (Type(BF) eq Type(RealField())) or (Type(BF) eq Type(ComplexField())) then
    fl := true;
else
    fl := false;
end if;

if fl then
    R1<a,d> := PolynomialRing(BF,2);
else
    R1<a,d> := FunctionField(BF,2);
end if;
R2<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(R1,8);
```

The function then goes on to define local variables for the input data and to compute both addition laws. Then the question arises: which addition law should be the one returned as output? We will go

into more detail about why we do it like this at the end of the next section. For now it suffices to say that the function decides this by checking if certain cases of zero coordinates happen:

```
if fl then
    if (Abs(BF!X_o) gt 1E-10 or Abs(BF!Z_o) gt 1E-10) and
    (Abs(BF!Y_o) gt 1E-10 or Abs(BF!T_o) gt 1E-10) then
        X := X_o; Y := Y_o; Z := Z_o; T := T_o;
    elif (Abs(BF!X_d) gt 1E-10 or Abs(BF!Z_d) gt 1E-10) and
    (Abs(BF!Y_d) gt 1E-10 or Abs(BF!T_d) gt 1E-10) then
        X := X_d; Y := Y_d; Z := Z_d; T := T_d;
    else
        return "Error: all are zero";
    end if;
else
    if (X_o ne 0 or Z_o ne 0) and (Y_o ne 0 or T_o ne 0) then
        X := X_o; Y := Y_o; Z := Z_o; T := T_o;
    elif (X_d ne 0 or Z_d ne 0) and (Y_d ne 0 or T_d ne 0) then
        X := X_d; Y := Y_d; Z := Z_d; T := T_d;
    else
        return "Error: all are zero";
    end if;
end if;
```

Here we have again case separation between a rational or finite base field and the rest because of those possibly inaccurate near-zero computations in the case of non-exact fields. If no errors have happened along the way, the function then returns the value of the addition law. The whole function is build in a way that this output can be numerical, if the input are two specific points, or symbolic, if the input contains at least one unknown coordinate.

To illustrate how the function works, here we have a couple of examples:

```
> Ead_AddL_eval(Rationals(),[82,17],[[1,1],[9/4,1]],[[PR|X2,Z2],
  [PR|Y2,0]]);

[
    [
        Y2*Z2,
        153/4*X2*Y2
    ],
    [
        9/4*Y2*Z2,
        -153/4*X2*Y2
    ]
]


> Ead_AddL_eval(RealField(),[82,17],[[1,1],[9/4,1]],[[1,Sqrt(17*82)],
  [Sqrt(82),Sqrt(17)*9/4]]);

[
    [
        788.38446858908883505108217353 7,
        1521.42597585291674288410023685
    ],
    [
        770.27369831281400179793455720 4,
        317.22143907095875855188472079 4
    ]
]
```

The other two functions left in Section A.3 are `ProjCoord` [8], which takes an affine point and embeds it onto $\mathbb{P}^1 \times \mathbb{P}^1$, and `AffCoord` [9], which does the opposite as long as the point is not one at infinity. They come in handy when we want to work with affine coordinates but we want to use the complete system of addition laws.

3.3.3 Exceptional points

While studying the completeness of the system of addition laws in the previous section, we have seen that neither the original addition law nor the dual one have exceptional points such that either law outputs $((0:0),(0:0))$. But that doesn't mean that they don't have exceptional points. Note that a point $((X:Z),(Y:T))$ doesn't have valid coordinates as well if $X = Z = 0$ or $Y = T = 0$. So, following the steps we used in the *Exceptional points* subsection 3.1.2, we will now show the computations to study when does this happen. We will do the process for $(X_3 : Z_3)$, but the complete code can be found in the online file `TwEdC_AdL_exceptionalpts.m`.

We start with the same set up as before: a `PolynomialRing` defined over a `FunctionField`, the original addition law formulae stored in `X3_o`,`Y3_o`,`Z3_o`,`T3_o` and the dual ones in `X3_d`,`Y3_d`,`Z3_d` and `T3_d`. Our first step is to do the variable change. Here `sd` is the square root of d and, regarding the notation from Section 3.1.2, the unknowns are `x` $= \alpha_2$, `y` $= \beta_2$ and, since we assume P_1 is known, `m` $= \alpha_1$, `n` $= \beta_1$ are considered coefficients:

```
> RR2<x1,y1,z1,t1, x2,y2,z2,t2> := FunctionField(R1,8);
> RR3<m,n,x,y> := FunctionField(R1,4);
> h := hom<RR3->RR2|x1/z1,y1/t1,x2/z2,y2/t2>;
> hinv := hom<RR2->RR3| Numerator(m),Numerator(n),Denominator(m),
  Denominator(n), Numerator(x),Numerator(y),Denominator(x),Denominator(y)>;

> hX3_o := hinv(RR2!X3_o); hZ3_o := hinv(RR2!Z3_o);
> printf "%o \t= %o \t\t= 0 \n%o \t= %o \t= 0 \n",X3_o,hX3_o,Z3_o,hZ3_o;

 X1*T1*Y2*Z2 + Y1*Z1*X2*T2       = m*y + n*x             = 0
 sd^2*X1*Y1*X2*Y2 + Z1*T1*Z2*T2  = sd^2*m*n*x*y + 1      = 0
```

We solve the system for `x` and `y`. Since it's not linear, we can't solve it in Magma but for example Matlab can solve it with these lines:

```
>> syms x y m n sa sd
>> [solx,soly] = solve(m*y + n*x == 0,sd^2*m*n*x*y + 1 == 0)

 solx =
 -1/(n*sd)
 1/(n*sd)

 soly =
 1/(m*sd)
 -1/(m*sd)
```

Now we have to undo the variable change and so we get our solutions. Here we divide the solutions by certain coefficients purely to obtain an output consistent with how we wrote the exceptional points previously:

```
> solx := [-1/(n*sd),1/(n*sd)]; soly := [1/(m*sd),-1/(m*sd)];
> solxz := [h(s) : s in solx]; solyt := [h(s) : s in soly];
> solx2 := [Numerator(s) : s in solxz];
> solz2 := [Denominator(s) : s in solxz];
> soly2 := [Numerator(s) : s in solyt];
> solt2 := [Denominator(s) : s in solyt];
> solP2 := [[[R2|solx2[i]/Coefficients(solx2[i])[1],
  solz2[i]/Coefficients(solx2[i])[1]],
  [soly2[i]/Coefficients(soly2[i])[1],solt2[i]/Coefficients(soly2[i])[1]]]
```

```
  : i in [1..#solx2]];
> P2_o1 := solP2_o1[2]; mP2_o1 := solP2_o1[1]; P2_o1; mP2_o1;

 [ [ T1, sd*Y1 ], [ Z1, -sd*X1 ] ]
 [ [ T1, -sd*Y1 ], [ Z1, sd*X1 ] ]
```

Now that we have the solutions to the system, we can do a couple of checks. First: are the solutions points on the curve? Using the quotient ring to define P_1 and P_2 as points on the curve, we can evaluate the curve equation on the solution points and if the output is zero, then the points are indeed on the curve:

```
> I := ideal<R2|Ead1,Ead2>; QI := R2/I;
> QI!Evaluate(Ead1,[T1,Z1,sd*Y1,-sd*X1,1,1,1,1]);
> QI!Evaluate(Ead1,[T1,Z1,-sd*Y1,sd*X1,1,1,1,1]);

 0
 0
```

Another check we can do is to see that the solution points are indeed exceptional, i.e., they make $(X_3 : Z_3)$ evaluate to $(0 : 0)$. And if we evaluate any other pair of law formulae on these exceptional points, we can see that they are indeed only exceptional for $(X_3 : Z_3)$.

```
> QI!Evaluate(X3_o,[X1,Y1,Z1,T1,P2_o1[1][1],
  P2_o1[2][1],P2_o1[1][2],P2_o1[2][2]]);
> QI!Evaluate(X3_o,[X1,Y1,Z1,T1,P2_o1[1][1],
  P2_o1[2][1],P2_o1[1][2],P2_o1[2][2]]);

0
0

> QI!Evaluate(Y3_d,[X1,Y1,Z1,T1,P2_o1[1][1],
  P2_o1[2][1],P2_o1[1][2],P2_o1[2][2]]);
> QI!Evaluate(T3_d,[X1,Y1,Z1,T1,P2_o1[1][1],
  P2_o1[2][1],P2_o1[1][2],P2_o1[2][2]]);

-sa^2*X1^2*T1^2 - Y1^2*Z1^2
2*sd*X1*Y1*Z1*T1
```

Note that most of the exceptional points include some square root of a and/or d. This means that depending on the basis field we are working on and the curve parameters, some of these exceptional points can't happen. When that is the case, the addition law can be complete over said basis field. The only pair of exceptional points for which this doesn't apply is the one of the dual addition law pair (Y_3', T_3'). So we know that the dual addition law always has exceptional points.

For example we have the case of the Edwards curve E-222 over the finite field $K = \mathrm{GF}(2^{222} - 117)$ defined by $a = 1$ and $d = 160102$. Here a is a square over K but d isn't. So the original addition law is complete over K. But the dual addition law still has exceptional cases, as for example point doubling:

```
> a := 1; d := 160102;
> P1 = [[27056910798826810903895890012519629544461773675417114745O\
  2428610129, 1],[28, 1]];
> Evaluate(Y3_d, [P1[1][1],P1[2][1],P1[1][2],P1[2][2],
  P1[1][1],P1[2][1],P1[1][2],P1[2][2]]);
> Evaluate(T3_d, [P1[1][1],P1[2][1],P1[1][2],P1[2][2],
  P1[1][1],P1[2][1],P1[1][2],P1[2][2]]);

0
0
```

On the other hand, if we work for example over the reals and a and d are greater than 0, then all exceptional cases can happen so none of the addition laws is complete and then having a complete system of addition laws comes in really handy. An issue that arises when working over fields whose elements have decimal parts, as the reals, is precision. Let's take for example the curve over $K = \mathbb{R}$ with parameters $a = 82$, $d = 17$. The point $P_1 = ((1 : 1), (9/4 : 1))$ is a point on the curve and together with $P_2 = ((Z_1 : \sqrt{ad}X_1), (\sqrt{a}T_1 : \sqrt{d}Y_1)) = ((1 : 37.33630941), (9.055385138 : 9.276987658))$, they are an exceptional case for the pair (Y_3, T_3) of the original addition law:

```
> a := 82; d := 17; sa := Sqrt(82); sd := Sqrt(17);
> P1 := [[1,1],[9/4, 1]]; P2 := [[1,sa*sd*1],[sa*1,sd*(9/4)]];
> Evaluate(Y3_o, [P1[1][1],P1[2][1],P1[1][2],P1[2][2],
  P2[1][1],P2[2][1],P2[1][2],P2[2][2]]);
> Evaluate(T3_o, [P1[1][1],P1[2][1],P1[1][2],P1[2][2],
  P2[1][1],P2[2][1],P2[1][2],P2[2][2]]);

-5.960464478E-8
2.980232239E-8
```

However, note that the result is not a sharp $(0 : 0)$. This is the reason why in our function `Ead_Addl_eval` these cases need to be treated separately.

Now that we have the exceptional points for each pair of addition law formulae, we could use them to further improve our `Ead_Addl_eval` function. We would do this by checking if any given pair of points to be added is an exceptional pair for either addition law and then only computing the addition law that we need. But implementing these conditions for the exceptional cases involves a considerable amount of lines of code to check each case. We have not studied if that might result in some faster computation of the addition but in many cases it would definitely involve more computations than they are done in the function as it is defined now. Thus we have decided to keep the computation of both addition laws within the function and instead check only when do each addition law yield a point with valid coordinates to decide which output to give.

3.3.4 Relation to Edwards model

The maps between the Edwards model and the twisted Edwards one are rather trivial. We have nevertheless made a script for them in Magma as well, for the sake of completeness.

Here we will use `b` to denote the parameter of the Edwards curve since we can't use `d` in two places at the same time. The variables `sa`,`sd`,`sb` denote the square roots of `a`,`d` and `b` respectively. We also include in the list of coefficients two points we will assume to be known. In the case of the twisted Edwards model, these points will be `(x1,y1)`, `(x2,y2)` and for the Edwards case we will use `(u1,v1)`, `(u2,v2)`.

With all this information, we define the corresponding algebraic structures where we will be working:

```
> BF := Rationals();
> // For the twisted Edwards model
> AD<a,d,sa,sd,x1,y1,x2,y2> := FunctionField(BF,8);
> Rxy<x,y> := PolynomialRing(AD,2);

> // For the Edwards model
> B<b,sb,u1,v1,u2,v2> := FunctionField(BF,6);
> Ruv<u,v> := PolynomialRing(B,2);
```

Since the only fractions are in the maps of the parameters, we can use Magma 's homomorphisms to build the maps. Starting with the direction Edwards model to twisted Edwards, we define two maps: one for the parameters, `hb`, and another one for the unknowns, `h`:

```
> hb := hom<B -> AD | d/a,sd/sa,sa*x1,y1,sa*x2,y2>;
> h := hom<Ruv -> Rxy | hb,sa*x,y>;
```

To go from the twisted Edwards model to the Edwards one, the identity map would suffice, taking into account the parameter change `b = d/a`. But in order to illustrate the inverse of `h`, we have written a bit of a different map:

```
> gad := hom<AD -> C | Denominator(b),Numerator(b),Denominator(sb),
  Numerator(sb),u1/Denominator(sb),v1,u2/Denominator(sb),v2>;
> g := hom<Rxy -> Ruv | gad, u/Denominator(sb),v>;
```

Applying these maps to one of the curve equations yields the other one in any case. Another point we can check is if addition laws are preserved. To do this we define `(x3,y3)` as the addition of `(x1,y1) + (x2,y2)` with the twisted Edwards curve addition law. Then we check that `(h(x1),h(y1)) + (h(x2),h(y2)) = (h(x3), h(y3))`, using the Edwards addition this time . And do the analogous process for the other direction:

```
> x3 := (x1*y2 + x2*y1)/(1 + d*x1*x2*y1*y2);
  y3 := (y1*y2 - a*x1*x2)/(1 - d*x1*x2*y1*y2);
> u3 := (u1*v2 + u2*v1)/(1 + b*u1*u2*v1*v2);
  v3 := (v1*v2 - u1*u2)/(1 - b*u1*u2*v1*v2);

> Evaluate(x3, [1,hb(b),1,hb(sb),h(u1),h(v1),h(u2),h(v2)]) eq h(u3);
> Evaluate(y3, [1,hb(b),1,hb(sb),h(u1),h(v1),h(u2),h(v2)]) eq h(v3);
true
true

> Evaluate(u3, [g(d)/g(a),g(sd)/g(sa),g(x1),g(y1),g(x2),g(y2)]) eq g(x3);
> Evaluate(v3, [g(d)/g(a),g(sd)/g(sa),g(x1),g(y1),g(x2),g(y2)]) eq g(y3);
true
true
```

The online script corresponding to what has been shown in this section is EdC_TwEdC_coord_change.m.

3.3.5 Relation to Weierstrass model: birational equivalence

We have expanded the numerical functions presented in the previous chapter to implement the birational equivalence maps from the twisted Edwards model to the Weierstrass one. Thus we have `Ew2Ead_P4_eval` [Fn.18] and `Ead2Ew_P4_eval` [Fn. 19] for the Φ maps and `Ew2Ead_eval` [Fn.20] and `Ead2Ew_eval` [Fn.21] for the Ψ maps. They all work in a pretty similar way to the ones from the previous chapter, except for a few details.

For example the fact that `Ew2Ead_P4_eval` and `Ew2Ead_eval` take as extra paramenter `a := 1`, in case we want a curve in twisted Edwards form with $a \neq 1$ as output. Additionally, for the Edwards to twisted Edwards isomorphism part to always work, our functions need a little bit extra, i.e., they need to have the chance to work over $K(\sqrt{a})$ if necessary. For that purpose, these lines are added to the beginning of all four functions:

```
tf, sqrt_a := IsSquare(BF!a);
if not tf then
    BF := AlgebraicClosure(BF);
    tf, sqrt_a := IsSquare(BF!a);
end if;
```

This means that the output of the function might be over a new basis field: the algebraic closure of the input basis field. In case that happens, in order to be able to do further computations with the obtained numerical results, the output of these functions also includes `BF`.

Of course we have tested our functions and the birational equivalence properties with a few examples. And next here we will show how addition is preserved through Ψ with an example over a finite field. But for more one can check the online scripts in the /Maps_between_models/WC_TwEdC directory.

Let's consider the curve in Weierstrass form with parameters $a_2 = 12$, $a_4 = 16$ over $\mathbb{F}_{23}$. Two points on the curve are `p_w := [1, 12]` and `q_w := [15, 17]`, which adds to:

```
> pq_w := (Cw!p_w)+(Cw!q_w); pq_w := [pq_w[1], pq_w[2]]; pq_w;

[ 22, 15 ]
```

We are going to apply the Ψ map to these points for which we will use he function `Ew2Ead_eval`. And we aim at a curve in twisted Edwards form with $a = 11$. These are the results:

```
> p_ad := Ew2Ead_iso_eval(BF,lst_w,p_w: a := 11);
> q_ad := Ew2Ead_eval(BF,lst_w,q_w: a := 11)[3];
> pq_ad := Ew2Ead_eval(BF,lst_w,pq_w: a := 11)[3];
> printf "a = %o, d = %o", p_ad[2][1], p_ad[2][2];
> printf "p_ad = [%o, %o] \n", p_ad[3][1], p_ad[3][2];
> printf "q_ad = [%o, %o] \n", q_ad[1], q_ad[2];
> printf "pq_ad = [%o, %o] \n", pq_ad[1], pq_ad[2];

a = 11, d = 9
p_ad = [8*r1, 6]
q_ad = [13*r1, 8]
pq_ad = [12*r1, 4]
```

Note that these output points all contain a product by `r1`. This is because 11 is not a square in $\mathbb{F}_{23}$ and so `Ew2Ead_eval` had to work over $\bar{\mathbb{F}}_{23}$ and it assigned the value of $\sqrt{11}$ to `r1`.

Defining `Cad` to be the curve with parameters $a = 11$, $d = 9$, we can easily check that `p_ad`, `q_ad` and `pq_ad` are indeed on the output curve:

```
> printf "p_ad -> %o, q_ad -> %o, pq_ad -> %o \n",
    p_ad in Cad, q_ad in Cad, pq_ad in Cad;

p_ad -> true, q_ad -> true, pq_ad -> true
```

Lastly, we can use the addition law for the twisted Edwards model to add `p_ad`, `q_ad` and so we'll see if the result matches `pq_ad`:

```
> r_ad := Ead_AddL_eval(BF,lst_ad, ProjCoord(BF,p_ad), ProjCoord(BF,q_ad));
> r_ad := AffCoord(BF,r_ad);
> printf "r_ad = [%o, %o] \n", r_ad[1], r_ad[2];

r_ad = [12*r1, 4]
```

And thus we have that in this example the Ψ birational equivalence has preserved the addition over $\bar{\mathbb{F}}_{23}$.

3.3.6 Relation to Weierstrass model: 2-isogeny

Regarding the isogenous relation between the twisted Edwards and the Weierstrass models, we have implemented the isogenies into two functions to work numerically with them. Afterwards we move on to check that the image of each isogeny is indeed a point in the codomain curve and we show as well the addition preservation and the doubling property of the isogenies.

We can start by taking a look at the function `Ew2Ead_iso_eval` [Fn. 22], which takes as input the base field `BF`, the list `lst` of the five parameters of an elliptic curve in $\mathcal{E}_w$ form and a point `p` on the curve with affine coordinates. Additionally, we can specify a couple of parameters: a point `PP4` that is the double of a point of order 4 from $\mathcal{E}_w(K)$ and the u_i as the parameter `u_lst` on the function. The default value for `PP4` is `[0,0]` and we might need to change this if the input is in complete Weierstrass form. If no `u_lst` is provided then the default value is `[0,0,0]` and we obtain the u_i by solving the corresponding polynomial, as can be seen in the snipped of the function's code below. Note that we check if the polynomial has 3 different solutions in K and, if it doesn't, we move on to work on $\bar{K}$:

```
if u_lst eq [0,0,0] then
    PR<s> := PolynomialRing(BF);
    if a1 ne 0 or a3 ne 0 then
        a6 := 0;
    end if;
```

```
        rts := Roots(s^3 + a2*s^2 + a4*s + a6);
        if #rts lt 3 then
            BF := AlgebraicClosure(BF);
            PR<s> := PolynomialRing(BF);
            rts := Roots(s^3 + BF!a2*s^2 + BF!a4*s + BF!a6);
        end if;
        rts_abs := [Abs(rts[i][1]): i in [1..3]];
        ParallelSort(~rts_abs,~rts);
        u0 := rts[1][1]; u1 := rts[2][1] - u0; u2 := rts[3][1] - u0;
    else
        u0 := BF!u_lst[1]; u1 := BF!u_lst[2]; u2 := BF!u_lst[3];
    end if;
```

While all possible outputs are valid and isogenous to the input curve, sometimes we'd like to work with a specific curve. And that's the use of specifying `u_lst`: make sure the three possible values for the u_i are assigned in the correct order when one has a specific curve in mind, since different order means different output curve.

Note that we define `u1` and `u2` already substracting `u0` from them, as a way to implicitly implement the change of coordinates $(u', v) \to (u, v)$ for the u_i. This needs to be done as well for the coordinates of the input point, together with any possible adjustments for the cases where $a_1 \neq 0 \neq a_3$:

```
 u := BF!p[1] + u2p - u0; v := BF!p[2] + (a1*u + a3)/2;
```

The only thing left to do now is doing the computations to actually apply the isogeny. This is done in two parts to avoid trying to compute divisions by zero: first the denominators are computed and if either of them happens to be zero, an error is returned. Otherwise the function moves on to do the last computations:

```
denom_x := u1*u2 - u^2; denom_y := v^2 + (u1-u2)*u^2;
if denom_x eq 0 then
    return "Error: isogeny not defined for this point since (denom_x eq 0).";
elif denom_y eq 0 then
    return "Error: isogeny not defined for this point since (denom_y eq 0).";
else
    a := 4*u1; d := 4*u2;
    c_x := v/denom_x; c_y := (v^2 - (u1-u2)*u^2)/denom_y;

    return [* BF, [a,d],[c_x,c_y], u0 *];
end if;
```

As it can be seen above, the output of the function consists of a basis field (in case the function operated over $\bar{K}$), the parameters for an isogenous curve in $\mathcal{E}_{a,d}$ form, the corresponding isogenous point to `p` and the value of `u0`, since it will be needed if one wishes to retrieve the exact original input curve later.

In a similar way we have the dual isogeny implemented in the function `Ead2Ew_iso_eval` [Fn. 23]. The input for this one consists of the base field `BF`, the list `lst` consisting of the two parameters of an elliptic curve in $\mathcal{E}_{a,d}$ form and a point `p` on the curve with affine coordinates. The general isogenous curve in $\mathcal{E}_w$ form will always be one such that $u_0 = 0$, i.e., $a_6 = 0$. On account of the possibility that one wants a different output, `u0` can be provided as a function parameter. And in case we want a curve in complete Weierstrass form as an output, the function also counts with parameters for `a1`, `a3` and `PP4`.

With all this input information, the dual isogeny is computed in the same way as before, checking the denominators first:

```
denom_u := 4*x^2; denom_v := 32*x*(1+y)*(1-y);
if denom_u eq 0 then
    return "Error: isogeny not defined for this point since (denom_u eq 0).";
elif denom_v eq 0 then
    return "Error: isogeny not defined for this point since (denom_v eq 0).";
```

```
else
    u1 := a/4 + u0; u2 := d/4 + u0;
    a2 := -(u0+u1+u2); a4 := u0*u1 + u0*u2 + u1*u2; a6 := -u0*u1*u2;
    c_u := 1/denom_u +u0; c_v := (a-d)*((1-y)^2-(1+y)^2)/denom_v;

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;
    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

    return [*[a1,a2,a3,a4,a6],[u0,u1,u2],[c_u,c_v]*];
end if;
```

Note that here the formulae might not exactly match the ones in the theory since here we don't assume `u0` to be 0 so some adjustments were needed. Plus some extra computations are carried out for the complete Weierstrass case. And so we get the output, which in this case consists of the a_i parameters of the isogenous curve in $\mathcal{E}_w$ form, the corresponding u_i and the isogenous point to `p`.

With these functions in hand, we can try some examples and see the addition preservation and the doubling property of the isogenies. Here we will show some examples over the rationals, but the functions also work over other base fields. Some examples for finite fields can be found online in WC_TwEdC_iso_num.m.

First we can consider a curve with Weierstrass form $C_w : y^2 = x^3 - (5347/4)x^2 - (13770055/8)x+$ $+2301120900$. Two points on this curve are $p_w = (0, 47970)$, $q_w = (253223267168/246395809,$ $55479853577957508/3867675013873)$. And so we can add these points and, applying `Ew2Ead_iso_eval` to these three points, we have the following:

```
> BF := Rationals(); R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
> R2<u,v> := AffineSpace(R1,2); R3<x,y> := AffineSpace(R1,2);

> a1 := 0; a2 := -5347/4; a3 := 0; a4 := -13770055/8; a6 := 2301120900;
> Cw := EllipticCurve([a1,a2,a3,a4,a6]);
> p_w := [0,47970];
> q_w := [253223267168/246395809, 55479853577957508/3867675013873];
> pq_w := (Cw!p_w)+(Cw!q_w); pq_w := [pq_w[1], pq_w[2]];

> out_p_ad := Ew2Ead_iso_eval(BF,[a1,a2,a3,a4,a6],p_w); out_p_ad;

[*
    Rational Field,
    [ 10513, 10578 ],
    [ 144/15697, 5329/5201 ],
    -1312
*]

> out_q_ad := Ew2Ead_iso_eval(BF,[a1,a2,a3,a4,a6],q_w);
> out_pq_ad := Ew2Ead_iso_eval(BF,[a1,a2,a3,a4,a6],pq_w);

> Cad := Curve(R3, 1 + 10578*x^2*y^2 - 10513*x^2 - y^2);
> p_ad := R3!out_p_ad[3]; p_ad in Cad;
> q_ad := R3!out_q_ad[3]; q_ad in Cad;
> pq_ad := R3!out_pq_ad[3]; pq_ad in Cad;

true (144/15697, 5329/5201)
true (125297302150944/12894129196796737, 1100285933947201/436081679542081)
true (29217509337826509121924406982885 60/30342784411173763014660804928735\
 9441, -48196275389508609779308584030224209/42047593363740412523602033206\
 854959)
```

As we have seen, p_ad, q_ad and pq_ad are indeed on the isogenous $\mathcal{E}_{a,d}$ curve. Now let's add p_ad and q_ad using the addition law for $\mathcal{E}_{a,d}$ to see if the result matches pq_ad:

```
> r_ad := Ead_AddL_eval(BF,lst_ad, ProjCoord(R1,p_ad), ProjCoord(R1,q_ad));
> r_ad := AffCoord(BF,r_ad); r_ad;

[ 29217509337826509121924406982885 60/30342784411173763014660804928735 9441,
-48196275389508609779308584030224209/42047593363740412523602033206854959 ]
```

As we can see, pq_ad = r_ad and so we have an example of how the isogeny preserves the addition law. Note that if now we wanted to use Ead2Ew_iso_eval on p_ad without specifying a u0, we would also get an isogenous curve and a valid point, just not the curve we started with:

```
> out_p_w := Ead2Ew_iso_eval(BF,[10513,10578],p_ad); out_p_w;

[*
    [ 0, -21091/4, 0, 55603257/8, 0 ],
    [ 0, 10513/4, 5289/2 ],
    [ 246395809/82944, -435060076913/23887872 ]
*]

> "Cw eq Cw2?", [a1,a2,a3,a4,a6] eq out_p_w[1];
> Cw2 := EllipticCurve(out_p_w[1]); p_w2 := out_p_w[3];
> "p_w2 in Cw?", p_w2 in Cw; "p_w2 in Cw2?", p_w2 in Cw2;

Cw eq Cw2? false
p_w2 in Cw? false
p_w2 in Cw2? true (246395809/82944 : -435060076913/23887872 : 1)
```

Last but not least, we have an example starting with the $\mathcal{E}_{a,d}$ model: the curve $C_{a,d} : 82x^2 + y^2 = 1 + 17x^2y^2$. A point on this curve is $p_{tw} = (1, 9/4)$ and so we apply both isogenies to it:

```
> a := 82; d := 17; p_ad := [1, 9/4];
> out_w := Ead2Ew_iso_eval(BF,[a,d],p_ad); out_w;

[*
    [ 0, -99/4, 0, 697/8, 0 ],
    [ 0, 41/2, 17/4 ],
    [ 1/4, 9/2 ]
*]

> out_ad := Ew2Ead_iso_eval(BF,out_w[1],out_w[3]); out_ad;

[*
    Rational Field,
    [ 17, 82 ],
    [ 72/1393, 1361/1231 ],
    0
*]

> "Cad eq Cad1?", [a,d] eq out_ad[2];
> Cad1 := Curve(R3, 1 + out_ad[2][2]*x^2*y^2 - out_ad[2][1]*x^2 - y^2);
> "q_ad in Cad1?", out_ad[3] in Cad1;

Cad eq Cad1? false
q_ad in Cad1? true (72/1393, 1361/1231)
```

Although everything computes, there seems to have been a problem since after applying both `Ead2Ew_iso_eval` and `Ew2Ead_iso_eval`, the output curve is not $C_{a,d}$. This is due to the fact that `Ew2Ead_iso_eval` has computed the correct u_i but obtained them in a different order. So specifying the `u_lst` fixes this and allows us to see how p_{tw} is doubled through the isogenies:

```
> out_ad2 := Ew2Ead_iso_eval(BF,out_w[1],out_w[3]: u_lst := out_w[2]);
> out_ad2;

[*
    Rational Field,
    [ 82, 17 ],
    [ 72/1393, 1231/1361 ],
    0
*]

> "Cad eq Cad2?", [a,d] eq out_ad2[2];
> Cad2 := Curve(R3, 1 + out_ad2[2][2]*x^2*y^2 - out_ad2[2][1]*x^2 - y^2);
> q_ad2 := out_ad2[3]; "q_ad2 in Cad2?", q_ad2 in Cad2;
> p_ad_db := AffCoord(BF,Ead_AddL_eval(BF,[a,d],
    [[p_ad[1],1],[p_ad[2],1]],[[p_ad[1],1],[p_ad[2],1]]));
> "q_ad2 = 2p_ad?", q_ad2 eq p_ad_db;

Cad eq Cad2? true
q_ad2 in Cad2? true (72/1393, 1231/1361)
q_ad2 = 2p_ad? true
```

Chapter 4

Other models

In the previous chapters we have studied in depth the Weierstrass, the Edwards and the twisted Edwards models, their respective addition laws and how they are all related. We have shown how far we have got studying all that using Magma so further research could be done finding ways to fully prove all the results using only computer algebra. But of course another possible direction for follow up work would be to look into the many other elliptic curve models out there. For completeness' sake, but without going into too much detail, we will present in this chapter some more models as a way of indeed pointing a direction into the future work that could be done to keep understanding how all different elliptic curve models are connected.

If one were to simply compile a list of names for all the models out there that define elliptic curves, they would soon find themselves submerged in an ocean of new and old models with several generalizations, twists and modifications to improve different computational specifics. Such is the case for example of the doubling-oriented and the tripling-oriented Doche–Icart–Kohel forms: they are particular cases of the Weierstrass model of special interest thanks to their faster multiplication methods [17]. In order to avoid getting lost in too many specifics for the models to come, here we will focus on just trying to paint an overview of how several other families of elliptic curves look like in as general terms as possible.

In that spirit, we will start by talking about the Montgomery and Hessian models, followed by the Jacobi intersections and the Jacobi quartic. At the end we will also mention the family of Huff models. For each model an addition law will be provided and a projective form, together with a general idea of how do they relate to at least one other model.

4.1 Montgomery model

In 1987 P. L. Montgomery published his paper *Speeding the Pollard and elliptic curve methods of factorization* [31], where he studied ways of speeding up said factorization methods. In that paper he introduced a different parametrization of elliptic curves, later known as the Montgomery curves, and the Montgomery ladder, a new method of computing the n-th addition of a point P to itself, i.e., nP. As it happens, the computation of the x coordinate of nP is at the heart of the ECM factorization method, as well as the Diffie-Hellman key exchange in the context of elliptic curve cryptography. And Montgomery's ladder offers an efficient way to compute said coordinate when working with an elliptic curve given in Montgomery's form.

In [14] C. Costello and B. Smith elaborate on these topics, referring as well to their cryptographic applications. From their paper we have the following definitions:

Definition 16. Let K be a field and $\mathcal{C}$ an elliptic curve over K, then its Montgomery form is $\mathcal{E}_{A,B}\colon By^2 = x^3 + Ax^2 + x$, where $A, B \in K$ are such that $B(A^2 - 4) \neq 0$.

Let K be a field, $\mathcal{C}$ an elliptic curve over K and (x_1, y_1), (x_2, y_2) two points on $\mathcal{C}$. Fix A, B to be such that $A, B \in K$, $B(A^2 - 4) \neq 0$. Then the addition law of $\mathcal{C}$ on its Montgomery form $\mathcal{E}_{A,B}$ is as follows:

if $P_1 = -P_2 = (x_2, -y_2)$, *then:*

$$P_1 + P_2 = P_1 - P_1 = \mathcal{O}$$

otherwise:

$$P_1 + P_2 = (x_3, y_3) = \left(B\lambda^2 - (x_1 + x_2) - A, \lambda(x_1 - x_3) - y_1\right)$$

where:

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P_1 \neq \pm P_2 \\ \dfrac{3x_1^2 + 2Ax_1 + 1}{2By_1}, & \text{if } P_1 = P_2 \end{cases}$$

The zero of this addition law is $\mathcal{O} = (0 : 1 : 0)$ and the opposite of a point $P = (x, y)$ is $-P = (x, -y)$. The geometric interpretation of the addition law is the same chord and tangent method used for the Weierstrass' model addition law.

We get the following projective formulations from [28], where they also present a complete system of addition laws for this model. The projective curve equation for $(X : Y : Z) \in \mathbb{P}^2$ is:

$$BY^2Z = X^3 + AX^2Z + XZ^2$$

And the addition law, given two points on the curve $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2) \in \mathbb{P}_2$, is:

$$P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3) = \big(Bu^2vZ_1Z_2 - (AZ_1Z_2 + X_1Z_2 + X_2Z_1)v^3 :$$
$$-Bu^3Z_1Z_2 + (AZ_1Z_2 + 2X_1Z_2 + X_2Z_1)uv^2 - Y_1Z_2v^3 \ : \ v^3Z_1Z_2\big)$$

where:

$$(u, v) = \begin{cases} (Y_2Z_1 - Y_1Z_2, X_2Z_1 - X_1Z_2), & \text{if } P_1 \neq P_2 \\ (3X_1^2 + 2AX_1Z_1 + Z_1^2, 2BY_1Z_1), & \text{if } P_1 = P_2 \end{cases}$$

Regarding the connection of the Montgomery form to other models, as mentioned before in Section 3.2.3, it is isomorphic to the Weierstrass model and birationally equivalent to the twisted Edwards one. In [§3, [3]] and in [§2, [14]] one can find more details about said relations.

4.2 Hessian model

The speed of arithmetic calculations on any given model for elliptic curves is a very relevant quality. Elliptic curves given in different Hessian forms are highly valued thanks to their fast addition formulae. Already in 1986 D. V. Chudnovsky and G. V. Chudnovsky studied the efficient performance of the Hessian model addition in [12], corroborated by N.P. Smart in [38] and later by M. Joye and J.-J. Quisquater in [26], where they also mention its superior speed in elliptic curve scalar multiplication as well as its memory usage efficiency. Other versions of this model are the generalized Hessian form [20] and the twisted Hessian model [4], which also stand out for their cost efficient arithmetics.

We will start by presenting the Hessian model. Looking at different articles, one might find two slightly different ways to define this model. This is due to the fact that it is derived from the Weierstrass form, as shown in [38], and depending how the parameter D is defined in said derivation, the model can be defined by the equation $x^3 + y^3 + 1 = 3Dxy$ with the condition that $D^3 \neq 1$, or be defined as follows:

Definition 17. Let K be a field and $\mathcal{C}$ an elliptic curve over K, then its Hessian form is $\mathcal{E}_D \colon x^3 + y^3 + 1 = Dxy$, where $D \in K$ is such that $D^3 \neq 27$.

The derivation from the Weierstrass model mentioned above shows that this model is birationally equivalent to the Weierstrass one. In projective form, given by the usual projective map $x = X/Z$, $y = Y/Z$, the curve equation is the following:

$$\bar{\mathcal{E}}_D \colon X^3 + Y^3 + Z^3 = DXYZ$$

And the addition law in projective coordinates, derived from the chord and tangent method, is given by this system [38]:

Let K be a field and fix $D \in K$ to be such that $D^3 \neq 27$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\bar{\mathcal{E}}_D$ in $\mathbb{P}^2$. Also let $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ be two points on the curve. Then the complete set of addition laws of $\mathcal{C}$ is:

$$P_1 + P_2 = \begin{cases} P_3 &= (X_3 : Y_3 : Z_3) \\ P'_3 &= (X'_3 : Y'_3 : Z'_3) \end{cases} =$$

$$= \begin{cases} \big(Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1 \ : \ X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1 \ : \ Z_1^2 Y_2 X_2 - Z_2^2 Y_1 X_1\big), & \text{if } P_1 \neq P_2 \\ \big(Y_1(Z_1^3 - X_1^3) \ : \ X_1(Y_1^3 - Z_1^3) \ : \ Z_1(X_1^3 - Y_1^3)\big), & \text{if } P_1 = P_2 \end{cases}$$

The neutral element for this addition law is $(1 : -1 : 0)$ and the opposite of a point $P = (X : Y : Z)$ is $-P = (Y : X : Z)$. Note that this addition law does not depend on the curve parameter.

Let's move on to take a quick glance at the generalized and the twisted modifications of this model, both of which include more isomorphism classes of elliptic curves.

4.2.1 Generalized Hessian model

R. R. Farashahi and M. Joye introduce this model for elliptic curve in [20] and they study different addition laws for it. The generalized Hessian form for elliptic curves is defined as follows:

Definition 18. Let K be a field and $\mathcal{C}$ an elliptic curve over K, then its projective generalized Hessian form in $\mathbb{P}^2$ is $\bar{\mathcal{E}}_{C,D}\colon X^3 + Y^3 + CZ^3 = DXYZ$, where $C, D \in K$ such that $C(27C - D^3) \neq 0$.

This model doesn't have a unique point at infinity. Instead, it has the points $(1 : -\omega : 0)$ such that $\omega^3 = 1$ at infinity. Based on the addition laws from the Hessian model, the following are the addition laws for this generalized form:

Let K be a field and fix $C, D \in K$ to be such that $C(27C - D^3) \neq 0$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\bar{\mathcal{E}}_{C,D}$ in $\mathbb{P}^2$. Also let $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ be two points on the curve. Then the complete set of addition laws of $\mathcal{C}$ is:

$$P_1 + P_2 = \begin{cases} P_3 &= (X_3 : Y_3 : Z_3) \\ P'_3 &= (X'_3 : Y'_3 : Z'_3) \end{cases} =$$

$$= \begin{cases} \big(Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1 \ : \ X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1 \ : \ Z_1^2 Y_2 X_2 - Z_2^2 Y_1 X_1\big), & \text{if } P_1 \neq P_2 \\ \big(Y_1(CZ_1^3 - X_1^3) \ : \ X_1(Y_1^3 - CZ_1^3) \ : \ Z_1(X_1^3 - Y_1^3)\big), & \text{if } P_1 = P_2 \end{cases}$$

Here the zero of the addition is also $(1 : -1 : 0)$ and given a point $P = (X : Y : Z)$, $-P$ is $(Y : X : Z)$.

In [20] they further work on other addition law options, striving to find a unified addition law. And so they present the following to be a complete addition law for any elliptic curve in generalized Hessian form, as long as C is not a cube in K:

$$P_1 + P_2 = P_3 \ = \ (X_3 : Y_3 : Z_3) \ = \ \big(CY_2Z_2Z_1^2 - X_1Y_1X_2^2 \ : \ X_2Y_2Y_1^2 - CX_1Z_1Z_2^2 \ : \ X_2Z_2X_1^2 - Y_1Z_1Y_2^2\big)$$

Note that the condition for completeness for this addition law implies that this addition is not complete over arbitrary extensions of K, in particular $\bar{K}$.

They also study the relation to the Weierstrass model, determining that every elliptic curve in $\mathcal{E}_w$ form over a field K with a point of order 3 in $\mathcal{E}(K)$ is isomorphic over $\bar{K}$ to a curve in generalized Hessian form.

4.2.2 Twisted Hessian model

D. J. Bernstein, T. Lange et al. present in [4] a twist over the Hessian model so it looks like this:

Definition 19. Let K be a field and $\mathcal{C}$ an elliptic curve over K, then its projective twisted Hessian form in $\mathbb{P}^2$ is $\bar{\mathcal{E}}_{A,D}\colon AX^3 + Y^3 + Z^3 = DXYZ$, where $A, D \in K$ such that $A(27A - D^3) \neq 0$.

The point at infinity for this model is $(0 : -1 : 1)$ and, as it can be easily seen, it is similar to the generalized Hessian model, up to the order of the coordinates. In spite of their similarities, both models present different advantages as for example: the generalized Hessian is fully symmetric on X and Y and the twisted Hessian has a finite point as neutral element.

The system of addition laws for this model is the following:

Let K be a field and fix $A, D \in K$ to be such that $A(27A - D^3) \neq 0$ and A is not a cube in K. Let $\mathcal{C}$ be an elliptic curve over K defined by $\bar{\mathcal{E}}_{A,D}$ in $\mathbb{P}^2$. Also let $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ be two points on the curve. Then the complete set of addition laws of $\mathcal{C}$ is:

$$P_1 + P_2 = \begin{cases} P_3 &= (X_3 : Y_3 : Z_3) \\ P_3' &= (X_3' : Y_3' : Z_3') \end{cases} =$$

$$= \begin{cases} \left(X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1 \ : \ Z_1^2 X_2 Y_2 - Z_2^2 X_1 Y_1 \ : \ Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1\right), & \text{if defined} \\ \left(Z_2^2 X_1 Z_1 - Y_1^2 X_2 Y_2 \ : \ Y_2^2 Y_1 Z_1 - A X_1^2 X_2 Z_2 \ : \ A X_2^2 X_1 Y_1 - Z_1^2 Y_2 Z_2\right), & \text{if defined} \end{cases}$$

Furthermore, it holds that:

$$(X_3 : Y_3 : Z_3) = (0 : 0 : 0) \Leftrightarrow P_2 = (\omega^2 X_1 : \omega Y_1 : Z_1) \text{ for some } \omega \in K \text{ such that } w^3 = 1$$

$$(X_3' : Y_3' : Z_3') = (0 : 0 : 0) \Leftrightarrow P_2 = (Z_1 : \gamma^2 X_1 : \gamma Y_1) \text{ for some } \gamma \in K \text{ such that } \gamma^3 = a$$

In [Thm. 4.5, [4]] they prove that $(X_3 : Y_3 : Z_3) = (0 : 0 : 0) = (X_3' : Y_3' : Z_3')$ never happens as long as A is not a cube in K. So in that case, the formulae above define a complete system of addition laws. The neutral element for this addition law system is $(0 : -1 : 1)$ and the opposite of a point $P = (X : Y : Z)$ is $-P = (X : Z : Y)$. The twisted Hessian and the Weierstrass models are both isomorphic and isogenous as it is shown in [§5, [4]].

4.3 Jacobi quadric intersections

Aside from speed, another relevant aspect of an addition law of an elliptic curve is if it is unified, i.e., if the same formulae can be used for point doubling and for adding two different points. That is due to the fact that using addition laws that are not unified for elliptic curve systems in cryptography makes said systems susceptible to certain side-channel attacks [30]. As it turns out, the Jacobi models have a unified addition law so these models became of great interest in the early 2000s. Here we will show the model consisting of a quadric intersection and in the next section the Jacobi quartic will be explained.

One way to embed an elliptic curve given by a short Weierstrass equation into the projective space $\mathbb{P}^3$ is by the following intersection of quadrics:

$$\begin{cases} X^2 - TZ = 0 \\ Y^2 - aXZ - bZ^2 - TX = 0 \end{cases}$$

The embedding is given by the map $(x, y) \mapsto (X : Y : Z : T) = (x : y : 1 : x^2)$. In [30] P.-Y. Liardet and N.P. Smart give a unified addition law for this model, but they already point out that the formulae are too complicated to be of use in any real life implementation. And so they move on to give another intersection of quadrics with simpler yet still unified addition formulae:

$$\begin{cases} X^2 + Y^2 = T^2 \\ kX^2 + Z^2 = T^2 \end{cases}$$

The curve defined by this intersection is parametrized by a Jacobi elliptic function [pg. 413, [12]] and that is where this model gets its name. Furthermore, said curve corresponds to an elliptic curve in short Weierstrass form with three points of order two. More specifically, it corresponds to an elliptic curve given by the following equation via the map $(x, y) \mapsto (-2y, x^2 - \lambda, x^2 + 2x\lambda + \lambda, x^2 + 2x + \lambda)$, where $\lambda = 1 - k$:

$$y^2 = x(x + 1)(x + \lambda)$$

This special case of the Weierstrass model is known as the Legendre form. As it is explained in [§3, [30]], any curve with three points of order 2 given by a short Weierstrass equation can be written in Legendre form. Note that we have also worked with this special case of the Weierstrass model when deriving the isogeny between this model and the twisted Edwards one in Section 3.2.3.

Now that we know where it comes from, let us give the formal definition for the Jacobi model:

Definition 20. Let K be a field such that $\text{char}(K) \neq 2, 3$ and $\mathcal{C}$ an elliptic curve over K, then its projective Jacobi form $\bar{\mathcal{E}}_k$ is given by the intersection of $X^2 + Y^2 - T^2 = 0$ and $kX^2 + Z^2 - T^2 = 0$, where $k \in K \backslash \{0, \pm 1\}$.

The unified addition law for this model is the following [§3, [30]], [§2, [7]]:

Let K be a field such that $char(K) \neq 2, 3$ and fix $k \in K$ to be such that $k \neq 0, \pm 1$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\bar{\mathcal{E}}_k$ in $\mathbb{P}^3$. Also let $P_1 = (X_1 : Y_1 : Z_1 : T_1)$, $P_2 = (X_2 : Y_2 : Z_2 : T_2)$ be two points on the curve. Then the addition law of $\mathcal{C}$ is:

$$P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3 : T_3) =$$

$$= \big(T_1\, Y_2 X_1\, Z_2 + Z_1\, X_2\, Y_1\, T_2 : T_1\, Y_2\, Y_1\, T_2 - Z_1\, X_2 X_1\, Z_2 :$$

$$T_1\, Z_1\, T_2 Z_2 - k^2 X_1\, Y_1 X_2\, Y_2 : (T_1\, Y_2)^2 + (Z_1 X_2)^2\big)$$

The neutral element of this addition law is $(0 : 1 : 1 : 1)$ and the inverse of a point $P = (X : Y : Z : T)$ is $-P = (-X : Y : Z : T)$. And the three points of order 2 are $(0 : -1 : 1 : 1)$, $(0 : 1 : -1 : 1)$ and $(0 : 1 : 1 : -1)$.

There is a twisted version of this model by R. Feng, M. Nie and H. Wu given in [21]. In their paper they introduce the twisted Jacobian intersections, which have the model defined above as a special case. Furthermore, they provide a new addition law independent of the curve parameters and with faster performance.

4.4 Extended Jacobi quartic

In the spirit of finding faster and more memory efficient arithmetics on the Jacobi form studied by [30], O. Bille and M. Joye propose in [7] to work with a quartic instead of an intersection of quadrics. This allows to work in the projective space $\mathbb{P}^2$ instead of $\mathbb{P}^3$, which reduces the number of variables and therefore makes the addition computations more efficient.

One of the quartics Jacobi studied is the following, where $k \neq 0, \pm 1$:

$$y^2 = (1 - x^2)(1 - k^2 x^2)$$

This elliptic curve can be parametrized by the same elliptic functions that the Jacobi intersection from the previous section. From its parametrization, an addition law can also be derived as it is shown in [§2, [7]]. However, in said paper they propose to work with a more general equation, which defines the extended Jacobi quartic model:

Definition 21. Let K be a field such that $\text{char}(K) \neq 2, 3$ and $\mathcal{C}$ an elliptic curve over K, then its projective extended Jacobi quartic form is $\bar{\mathcal{E}}_{\epsilon,\delta}\colon Y^2 = \epsilon X^4 - 2\delta X^2 Z^2 + Z^4$, where $\epsilon, \delta \in K$ such that $\epsilon \neq 0, 1$ and $\delta \neq 1/2, 1$.

Note that the Jacobi quartic shown before is included in this extended version as the special case when $\epsilon = k^2$, $\delta = (1 + k^2)/2$. The addition law for this extended model is as follows:

Let K be a field such that $char(K) \neq 2, 3$ and fix $\epsilon, \delta \in K$ to be such that $\epsilon \neq 0, 1$ and $\delta \neq 1/2, 1$. Let $\mathcal{C}$ be an elliptic curve over K defined by $\bar{\mathcal{E}}_{\epsilon,\delta}$ in $\mathbb{P}^2$. Also let $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ be two points on the curve. Then the addition law of $\mathcal{C}$ is:

$$P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3) =$$
$$= \Big(X_1 Z_1 Y_2 + Y_1 X_2 Z_2 \ : \ \big((Z_1 Z_2)^2 + \epsilon (X_1 X_2)^2\big)\big(Y_1 Y_2 - 2\delta X_1 X_2 Z_1 Z_2\big) +$$
$$+ \ 2\epsilon X_1 X_2 Z_1 Z_2 (X_1^2 Z_2^2 + Z_1^2 X_2^2) \ : \ (Z_1 Z_2)^2 - \epsilon (X_1 X_2)^2 \Big)$$

This addition law is unified, as the one for the Jacobi intersection model. Its neutral element is $(0 : 1 : 1)$ and the opposite of a point $P = (X : Y : Z)$ is $-P = (-X : Y : Z)$. In [§3, [7]] they show the relation between this quartic model and the Jacobi form from the previous section. They also explain how any elliptic curve in short Weierstrass form with a point of order 2 can be expressed in $\bar{\mathcal{E}}_{\epsilon,\delta}$ form, where the point of order 2 is $(0 : -1 : 1)$. Furthermore, we can find a detailed study of how the Weierstrass model and the extended Jacobi one are connected in [39].

H. Hisil, K. Wong, et al. study this model further in [23], where they work with the homogeneous version of the Jacobi and the extended Jacobi quartics. They also show the pertinent arithmetics for them and study how much faster they are, as compared to other models like the Weierstrass and the Edwards ones.

4.5 Huff model

G. B. Huff published in 1948 an article where, in order to study a diophantine problem, he introduced a model for elliptic curves [25]. But it wouldn't be until 2010 that this model would be studied as such by M. Joye, M. Tibouchi and D. Vergnaud [27]. They begin their paper by presenting the Huff model for elliptic curves and a unified addition law for it, both of which we show here:

Definition 22. Let K be a field such that $\mathrm{char}(K) \neq 2$ and $\mathcal{C}$ an elliptic curve over K, then its projective Huff form in $\mathbb{P}^2$ is $\bar{\mathcal{E}}_{a,b}\colon aX(Y^2 - Z^2) = bY(X^2 - Z^2)$, where $a, b \in K\backslash\{0\}$ such that $a^2 \neq b^2$.

Let K be a field such that $\mathrm{char}(K) \neq 2$, $\mathcal{C}$ an elliptic curve over K and $(X_1 : Y_1 : Z_1)$, $(X_2 : Y_2 : Z_2)$ two points on $\mathcal{C}$. Fix a, b to be such that $a, b \in K\backslash\{0\}$, $a^2 \neq b^2$. Then the addition law of $\mathcal{C}$ on its Huff form $\mathcal{E}_{a,b}$ is as follows:

$$(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3) =$$
$$= \big((X_1 Z_2 + X_2 Z_1)(Y_1 Y_2 + Z_1 Z_2)^2 (Z_1 Z_2 - X_1 X_2) \ :$$
$$(Y_1 Z_2 + Y_2 Z_1)(X_1 X_2 + Z_1 Z_2)^2 (Z_1 Z_2 - Y_1 Y_2) \ :$$
$$(Z_1^2 Z_2^2 - X_1^2 X_2^2)(Z_1^2 Z_2^2 - Y_1^2 Y_2^2)\big)$$

This addition law is also derived via the chord and tangent method explained for the Weierstrass model, but using the point $(0 : 0 : 1)$ instead of the infinity point as neutral element. So the zero of this addition law is $(0 : 0 : 1)$, and the inverse of a point $P = (X : Y : Z)$ is $-P = (X : Y : -Z)$. Two points P_1, P_2 on the curve form an exceptional pair iff $X_1 X_2 = \pm Z_1 Z_2$ or $Y_1 Y_2 = \pm Z_1 Z_2$. Note that, as in the Hessian model case, this addition law does not depend on the curve parameters. In [Thm. 2, [27]] they study the relation of the Huff model to other elliptic curves.

The paper also includes different generalizations and extensions on this model, for which they also provide addition laws. Furthermore, they also include formulae on how to compute Tate pairings using this model. This paper is also the starting point of the study of this model and its variants. As examples of follow-up work that has been done in this matter there are the generalization studied by H. Wu and R. Feng [40] that extends the number of isomorphism curve classes covered by the model and the follow-up work in [33] by N. G. Orhon and H. Hisil to speed up several arithmetics on the model to make it competitive with respect to other models. There are also different studies on the binary case and the elliptic curve cryptography architectures related to it, like [11] and [34].

Appendix: functions in Magma

A Addition laws

A.1 Functions for symbolic checks

```
Ew_AddL_b2calc := function(BF,lst,abc_lst)

  //abc_lst --> case 1: [0,0,1]; case 2: [0,1,0]; case 3: [1,0,0];

  AAA<a1,a2,a3,a4,a6, A200200, A200110, A200101, A200011, A200020,
  A200002, A110200, A110110, A110101, A110011, A110020, A110002, A101200, A101110, A101101, A101011,
     A101020, A101002, A011200, A011110,
  A011101, A011011, A011020, A011002, A020200, A020110, A020101, A020011, A020020, A020002, A002200,
     A002110, A002101, A002011, A002020,
  A002002 > :=   FunctionField(BF, 41);

    if #lst eq 2 then
        a1 := AAA ! 0; a2 := AAA ! 0; a3 := AAA ! 0;
        a4 := AAA ! lst[1]; a6 := AAA ! lst[2];
    elif #lst eq 5 then
        a1 := AAA ! lst[1]; a2 := AAA ! lst[2]; a3 := AAA ! lst[3];
        a4 := AAA ! lst[4]; a6 := AAA ! lst[5];
    else
        return "Error: wrong number of coeffs";
    end if;

  RRR<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(AAA,6);
  I := ideal< RRR |
       Y1^2*Z1+a1*X1*Y1*Z1+a3*Y1*Z1^2-X1^3-a2*X1^2*Z1-a4*X1*Z1^2-a6*Z1^3,
       Y2^2*Z2+a1*X2*Y2*Z2+a3*Y2*Z2^2-X2^3-a2*X2^2*Z2-a4*X2*Z2^2-a6*Z2^3 >;
  Q<x1,y1,z1,x2,y2,z2> := quo<RRR | I>;

  F := A200200*X1^2*X2^2 + A200110*X1^2*X2*Y2 + A200101*X1^2*X2*Z2 + A200011*X1^2*Y2*Z2 + A200020*X1^2*Y2
     ^2 + A200002*X1^2*Z2^2 + A110200*X1*Y1*X2^2 + A110110*X1*Y1*X2*Y2 + A110101*X1*Y1*X2*Z2 + A110011*X1*
     Y1*Y2*Z2 + A110020*X1*Y1*Y2^2 + A110002*X1*Y1*Z2^2 + A101200*X1*Z1*X2^2 + A101110*X1*Z1*X2*Y2 +
     A101101*X1*Z1*X2*Z2 + A101011*X1*Z1*Y2*Z2 + A101020*X1*Z1*Y2^2 + A101002*X1*Z1*Z2^2 + A011200*Y1*Z1*
     X2^2 + A011110*Y1*Z1*X2*Y2 + A011101*Y1*Z1*X2*Z2 + A011011*Y1*Z1*Y2*Z2 + A011020*Y1*Z1*Y2^2 + A011002
     *Y1*Z1*Z2^2 + A020200*Y1^2*X2^2 + A020110*Y1^2*X2*Y2 + A020101*Y1^2*X2*Z2 + A020011*Y1^2*Y2*Z2 +
     A020020*Y1^2*Y2^2 + A020002*Y1^2*Z2^2 + A002200*Z1^2*X2^2 + A002110*Z1^2*X2*Y2 + A002101*Z1^2*X2*Z2 +
      A002011*Z1^2*Y2*Z2 + A002020*Z1^2*Y2^2 + A002002*Z1^2*Z2^2;

    monF := Monomials(F);

  B<b1,b2,b3,b4,b6> := FunctionField(BF,5);
  HHH<
  H200200, H200110, H200101, H200011, H200020, H200002,
  H110200, H110110, H110101, H110011, H110020, H110002,
  H101200, H101110, H101101, H101011, H101020, H101002,
  H011200, H011110, H011101, H011011, H011020, H011002,
  H020200, H020110, H020101, H020011, H020020, H020002,
  H002200, H002110, H002101, H002011, H002020, H002002> := PolynomialRing(B, 36);

  h := hom< AAA->HHH | b1,b2,b3,b4,b6,
  H200200, H200110, H200101, H200011, H200020, H200002,
  H110200, H110110, H110101, H110011, H110020, H110002,
  H101200, H101110, H101101, H101011, H101020, H101002,
  H011200, H011110, H011101, H011011, H011020, H011002,
  H020200, H020110, H020101, H020011, H020020, H020002,
  H002200, H002110, H002101, H002011, H002020, H002002 >;
```

```
  binv := hom<B->AAA |a1,a2,a3,a4,a6>;
  hinv := hom< HHH->RRR | binv,
  X1^2*X2^2, X1^2*X2*Y2, X1^2*X2*Z2, X1^2*Y2*Z2, X1^2*Y2^2, X1^2*Z2^2, X1*Y1*X2^2, X1*Y1*X2*Y2, X1*Y1*X2*
     Z2, X1*Y1*Y2*Z2, X1*Y1*Y2^2, X1*Y1*Z2^2, X1*Z1*X2^2, X1*Z1*X2*Y2, X1*Z1*X2*Z2, X1*Z1*Y2*Z2, X1*Z1*Y2
     ^2, X1*Z1*Z2^2, Y1*Z1*X2^2, Y1*Z1*X2*Y2, Y1*Z1*X2*Z2, Y1*Z1*Y2*Z2, Y1*Z1*Y2^2, Y1*Z1*Z2^2, Y1^2*X2^2,
      Y1^2*X2*Y2, Y1^2*X2*Z2, Y1^2*Y2*Z2, Y1^2*Y2^2, Y1^2*Z2^2, Z1^2*X2^2, Z1^2*X2*Y2, Z1^2*X2*Z2, Z1^2*Y2
     *Z2, Z1^2*Y2^2, Z1^2*Z2^2>;

  V := VectorSpace(B, 36);

  kappa := (Y1*Z2+Y2*Z1+a1*X2*Z1+a3*Z1*Z2)/(X1*Z2-X2*Z1);
  mu := -(Y1*X2+Y2*X1+a1*X1*X2+a3*X1*Z2)/(X1*Z2-X2*Z1);
  sXZ := kappa^2+a1*kappa-(X1*Z2+X2*Z1)/(Z1*Z2)-a2;
  sYZ := -(kappa+a1)*sXZ-mu-a3;
  lambda := (Y1*Z2-Y2*Z1)/(X1*Z2-X2*Z1);
  nu := -(Y1*X2-Y2*X1)/(X1*Z2-X2*Z1);
  f := lambda^2+a1*lambda-(X1*Z2+X2*Z1)/(Z1*Z2)-a2;
  g := -(lambda+a1)*f-nu-a3;
  Z0 := (X1*Z2-Z1*X2)^3/(Z1*Z2);
  a := RRR!abc_lst[1]; b := RRR!abc_lst[2]; c := RRR!abc_lst[3];

  ABC := ( RRR ! Q ! (RRR ! (a*Numerator(sXZ*Z0) + b*Numerator(sYZ*Z0) + c*Z1*Z2*Numerator(Z0))) div (Z1
     ^2*Z2^2));
  Z3 := -ABC;

  coefZ3 := Coefficients(Z3);
  monZ3 := Monomials(Z3);
  Z3_ci := [<coefZ3[i],Index(monF,monZ3[i])>:i in [1..#coefZ3]];

  Nx3 := (RRR ! Q ! (Numerator(f)*ABC)) div (Z1*Z2);
  Fx3 := F*(X1*Z2-Z1*X2)^2 + Nx3;
  Qx3 := Q ! Fx3;
  Cx3 := Coefficients(Qx3);
  Ex3 := [ [B | Coefficient(h(Cx3[i]),j,1) :j in [1..36]] : i in [1..88] ];
  Mx3 := Matrix(B,88,36, Ex3);

  Wx3 := [ h(Cx3[i])-&+[Ex3[i][j]*HHH.j : j in [1..36]] : i in [1..88]];
  Vx3 := Vector(B, 88, Wx3);

  Sx3, Kx3 := Solution(Transpose(Mx3), Vx3);
  X3 := -hinv(&+[ Sx3[i]*HHH.i : i in [1..36]]);

  coefX3 := Coefficients(X3);
  monX3 := Monomials(X3);
  X3_ci := [<coefX3[i],Index(monF,monX3[i])>:i in [1..#coefX3]];

  Ny3 := RRR ! Q ! (Numerator(g)*ABC) div (Z1*Z2);
  Fy3 := F*(X1*Z2-Z1*X2)^3 + Ny3;
  Qy3 := Q ! Fy3;
  Cy3 := Coefficients(Qy3);

  Ey3 := [ [B | Coefficient(h(Cy3[i]),j,1) :j in [1..36]] : i in [1..88] ];
  My3 := Matrix(B,88,36, Ey3);

  Wy3 := [ h(Cy3[i])-&+[Ey3[i][j]*HHH.j : j in [1..36]] : i in [1..88]];
  Vy3 := Vector(B, 88, Wy3);

  Sy3, Ky3 := Solution(Transpose(My3), Vy3);
  Y3 := -hinv(&+[ Sy3[i]*HHH.i : i in [1..36]]);

  coefY3 := Coefficients(Y3);
  monY3 := Monomials(Y3);
  Y3_ci := [<coefY3[i],Index(monF,monY3[i])>:i in [1..#coefY3]];

  return [*[X3,Y3,Z3],[X3_ci,Y3_ci,Z3_ci], monF*];
end function;
```

Function 1: `Ew_AddL_b2calc`, discussed in 1.2.2

```
Ew_AddL := function(BF,lst)
    R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
    R2<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(R1,6);
```

```
4
5     if #lst eq 2 then
6         a1 := R1 ! 0; a2 := R1 ! 0; a3 := R1 ! 0;
7         a4 := R1 ! lst[1]; a6 := R1 ! lst[2];
8     elif #lst eq 5 then
9         a1 := R1 ! lst[1]; a2 := R1 ! lst[2]; a3 := R1 ! lst[3];
10        a4 := R1 ! lst[4]; a6 := R1 ! lst[5];
11    else
12        return "Error: wrong number of coeffs";
13    end if;
14
15  X3_001 := R2!(-a2*X1^2*X2*Z2 + a1*X1^2*Y2*Z2 - a4*X1^2*Z2^2 + 2*X1*Y1*Y2*Z2 + a3*X1*Y1*Z2^2 + a2*X1*Z1*
       X2^2 + X1*Z1*Y2^2 + 2*a3*X1*Z1*Y2*Z2 - 3*a6*X1*Z1*Z2^2 - Y1^2*X2*Z2 - a1*Y1*Z1*X2^2 - 2*Y1*Z1*X2*Y2 -
        2*a3*Y1*Z1*X2*Z2 + a4*Z1^2*X2^2 - a3*Z1^2*X2*Y2 + 3*a6*Z1^2*X2*Z2);
16
17  Y3_001 := R2!(-3*X1^2*X2*Y2 + (a1*a2 - 3*a3)*X1^2*X2*Z2 + (-a1^2 - a2)*X1^2*Y2*Z2 + (a1*a4 - a2*a3)*X1
       ^2*Z2^2 + 3*X1*Y1*X2^2 + 2*a2*X1*Y1*X2*Z2 - 2*a1*X1*Y1*Y2*Z2 + a4*X1*Y1*Z2^2 + (-a1*a2 + 3*a3)*X1*Z1*
       X2^2 - 2*a2*X1*Z1*X2*Y2 + (-2*a1*a3 - 2*a4)*X1*Z1*Y2*Z2 + (3*a1*a6 - a3*a4)*X1*Z1*Z2^2 - Y1^2*Y2*Z2 +
        (a1^2 + a2)*Y1*Z1*X2^2 + 2*a1*Y1*Z1*X2*Y2 + (2*a1*a3 + 2*a4)*Y1*Z1*X2*Z2 + Y1*Z1*Y2^2 + (a3^2 + 3*a6
       )*Y1*Z1*Z2^2 + (-a1*a4 + a2*a3)*Z1^2*X2^2 - a4*Z1^2*X2*Y2 + (-3*a1*a6 + a3*a4)*Z1^2*X2*Z2 + (-a3^2 -
       3*a6)*Z1^2*Y2*Z2);
18
19  Z3_001 := R2!(3*X1^2*X2*Z2 + a2*X1^2*Z2^2 - a1*X1*Y1*Z2^2 - 3*X1*Z1*X2^2 + a4*X1*Z1*Z2^2 - Y1^2*Z2^2 -
       a3*Y1*Z1*Z2^2 - a2*Z1^2*X2^2 + a1*Z1^2*X2*Y2 - a4*Z1^2*X2*Z2 + Z1^2*Y2^2 + a3*Z1^2*Y2*Z2);
20
21
22  X3_010 := R2!(-a1*a2*X1^2*X2^2 - a2*X1^2*X2*Y2 + (-a1^2*a3 - 2*a1*a4)*X1^2*X2*Z2 + (-a1*a3 - a4)*X1^2*Y2
       *Z2 + (-a1*a3^2 - 3*a1*a6)*X1^2*Z2^2 + (a1^2 - a2)*X1*Y1*X2^2 + 2*a1*X1*Y1*X2*Y2 - 2*a4*X1*Y1*X2*Z2 +
        X1*Y1*Y2^2 + (-a3^2 - 3*a6)*X1*Y1*Z2^2 + (-a1*a4 - a2*a3)*X1*Z1*X2^2 - 2*a4*X1*Z1*X2*Y2 + (-2*a1*a3
       ^2 - 6*a1*a6 - 2*a3*a4)*X1*Z1*X2*Z2 + (-2*a3^2 - 6*a6)*X1*Z1*Y2*Z2 + (-a1^3*a6 + a1^2*a3*a4 - a1*a2*
       a3^2 - 4*a1*a2*a6 + a1*a4^2 - a3^3 - 3*a3*a6)*X1*Z1*Z2^2 + a1*Y1^2*X2^2 + Y1^2*X2*Y2 + a3*Y1^2*X2*Z2
       + (a1*a3 - a4)*Y1*Z1*X2^2 + 2*a3*Y1*Z1*X2*Y2 - 6*a6*Y1*Z1*X2*Z2 + (-a1^2*a6 + a1*a3*a4 - a2*a3^2 - 4*
       a2*a6 + a4^2)*Y1*Z1*Z2^2 - a3*a4*Z1^2*X2^2 - 3*a6*Z1^2*X2*Y2 + (-a3^3 - 6*a3*a6)*Z1^2*X2*Z2 + (-a1^2*
       a6 + a1*a3*a4 - a2*a3^2 - 4*a2*a6 + a4^2)*Z1^2*Y2*Z2 + (-a1^2*a3*a6 + a1*a3^2*a4 - a2*a3^3 - 4*a2*a3*
       a6 + a3*a4^2)*Z1^2*Z2^2);
23
24  Y3_010 := R2!((-a2^2 + 3*a4)*X1^2*X2^2 + (a1^2*a4 - 2*a1*a2*a3 - a2*a4 + 3*a3^2 + 9*a6)*X1^2*X2*Z2 + (3*
       a1^2*a6 - 2*a1*a3*a4 + a2*a3^2 + 3*a2*a6 - a4^2)*X1^2*Z2^2 + (a1*a2 - 3*a3)*X1*Y1*X2^2 + (2*a1*a4 -
       2*a2*a3)*X1*Y1*X2*Z2 + (3*a1*a6 - a3*a4)*X1*Y1*Z2^2 + (-a2*a4 + 9*a6)*X1*Z1*X2^2 + (6*a1^2*a6 - 4*a1*
       a3*a4 + 2*a2*a3^2 + 12*a2*a6 - 4*a4^2)*X1*Z1*X2*Z2 + (a1^4*a6 - a1^3*a3*a4 + a1^2*a2*a3^2 + 5*a1^2*a2
       *a6 - a1^2*a4^2 - a1*a2*a3*a4 - a1*a3^3 - 3*a1*a3*a6 + a2^2*a3^2 + 4*a2^2*a6 - a2*a4^2 - a3^2*a4 - 3*
       a4*a6)*X1*Z1*Z2^2 + a1*Y1^2*X2*Y2 + Y1^2*Y2^2 + a3*Y1^2*Y2*Z2 + (a1*a4 - a2*a3)*Y1*Z1*X2^2 + (6*a1*a6
        - 2*a3*a4)*Y1*Z1*X2*Z2 + (a1^3*a6 - a1^2*a3*a4 + a1*a2*a3^2 + 4*a1*a2*a6 - a1*a4^2 - a3^3 - 3*a3*a6)
       *Y1*Z1*Z2^2 + (3*a2*a6 - a4^2)*Z1^2*X2^2 + (a1^2*a2*a6 -  a1*a2*a3*a4 + 3*a1*a3*a6 + a2^2*a3^2 + 4*a2
       ^2*a6 - a2*a4^2 - 2*a3^2*a4 - 3*a4*a6)*Z1^2*X2*Z2 + (a1^3*a3*a6 - a1^2*a3^2*a4 + a1^2*a4*a6 + a1*a2*
       a3^3 + 4*a1*a2*a3*a6 - 2*a1*a3*a4^2 + a2*a3^2*a4 + 4*a2*a4*a6 - a3^4 - 6*a3^2*a6 - a4^3 - 9*a6^2)*Z1
       ^2*Z2^2);
25
26  Z3_010 := R2!(3*a1*X1^2*X2^2 + 3*X1^2*X2*Y2 + (a1^3 + 2*a1*a2)*X1^2*X2*Z2 + (a1^2 + a2)*X1^2*Y2*Z2 + (a1
       ^2*a3 + a1*a4)*X1^2*Z2^2 + 3*X1*Y1*X2^2 + (2*a1^2 + 2*a2)*X1*Y1*X2*Z2 + 2*a1*X1*Y1*Y2*Z2 + (2*a1*a3 +
        a4)*X1*Y1*Z2^2 + (a1*a2 + 3*a3)*X1*Z1*X2^2 + 2*a2*X1*Z1*X2*Y2 + (2*a1^2*a3 + 2*a1*a4 + 2*a2*a3)*X1*
       Z1*X2*Z2 + (2*a1*a3 + 2*a4)*X1*Z1*Y2*Z2 + (2*a1*a3^2 + 3*a1*a6 + a3*a4)*X1*Z1*Z2^2 +  a1*Y1^2*X2*Z2 +
        Y1^2*Y2*Z2 + a3*Y1^2*Z2^2 + a2*Y1*Z1*X2^2 + (2*a1*a3 + 2*a4)*Y1*Z1*X2*Z2 + Y1*Z1*Y2^2 + 2*a3*Y1*Z1*
       Y2*Z2 + (2*a3^2 + 3*a6)*Y1*Z1*Z2^2 + a2*a3*Z1^2*X2^2 + a4*Z1^2*X2*Y2 + (a1*a3^2 + 2*a3*a4)*Z1^2*X2*Z2
        + (a3^2 + 3*a6)*Z1^2*Y2*Z2 + (a3^3 + 3*a3*a6)*Z1^2*Z2^2);
27
28    return [[X3_001,Y3_001,Z3_001],[X3_010,Y3_010,Z3_010]];
29  end function;
```

Function 2: `Ew_AddL`, discussed in §1.2.3

```
1   Ead_AddL := function(BF,lst : except := false)
2
3     if except then
4         R1<a,d,sa,sd> := FunctionField(BF,4);
5     else
6         R1<a,d> := FunctionField(BF,2);
7     end if;
8
9       R2<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(R1,8);
10      if #lst eq 2 then
11          a := R1 ! lst[1]; d := R1 ! lst[2];
12      else
```

```
        return "Error: wrong number of coeffs";
    end if;

    X3_o := R2 ! X1*Y2*Z2*T1 + X2*Y1*Z1*T2;
    Z3_o := R2 ! Z1*Z2*T1*T2 + d*X1*X2*Y1*Y2;
    Y3_o := R2 ! Y1*Y2*Z1*Z2 - a*X1*X2*T1*T2;
    T3_o := R2 ! Z1*Z2*T1*T2 - d*X1*X2*Y1*Y2;

    X3_d := R2 ! X1*Y1*Z2*T2 + X2*Y2*Z1*T1;
    Z3_d := R2 ! a*X1*X2*T1*T2 + Y1*Y2*Z1*Z2;
    Y3_d := R2 ! X1*Y1*Z2*T2 - X2*Y2*Z1*T1;
    T3_d := R2 ! X1*Y2*Z2*T1 - X2*Y1*Z1*T2;

    return [[[X3_o, Z3_o], [Y3_o, T3_o]],[[X3_d, Z3_d], [Y3_d, T3_d]]];
end function;
```

Function 3: `Ead_AddL`, discussed in 3.3.2

A.2 Functions for point addition: based on geometric interpretation

```
Ew_AddL_geom := function(BF,lst,P,Q)

  R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
  R2<X,Y,Z> := ProjectiveSpace(R1,2);
  O := R2![0,1,0];

  //Get the coeffs
  if #lst eq 2 then
      a1 := R1 ! 0; a2 := R1 ! 0; a3 := R1 ! 0;
      a4 := R1 ! lst[1]; a6 := R1 ! lst[2];
  elif #lst eq 5 then
      a1 := R1 ! lst[1]; a2 := R1 ! lst[2]; a3 := R1 ! lst[3];
      a4 := R1 ! lst[4]; a6 := R1 ! lst[5];
  else
      return "Error: wrong number of coeffs";
  end if;

  // Get the coordinates
  Xp := R1 ! P[1]; Yp := R1 ! P[2]; Zp := R1 ! P[3]; P := R2![Xp,Yp,Zp];
  Xq := R1 ! Q[1]; Yq := R1 ! Q[2]; Zq := R1 ! Q[3]; Q := R2![Xq,Yq,Zq];

  // Define the Ew
  C := Curve(R2, (Y^2)*Z + a1*X*Y*Z + a3*Y*Z^2 - X^3-a2*(X^2)*Z-a4*X*Z^2 - a6*Z^3);

  // Compute the coeffs of the line and define it
  // We define the line as a curve because it makes the intersection easier
  if (P eq Q) and (Type(Rationals()) eq Type(BF)) and (a1 eq 0) and (a3 eq 0) then

      L := Curve(R2,X*Zp - Xp*Z);

  elif P eq Q then

      l := (3*Xp^2+2*a2*Xp*Zp+a4*Zp^2-a1*Yp*Zp)/(2*Yp*Zp+a1*Xp*Zp+a3*Zp^2);
      n := (-Xp^3+a4*Xp*Zp^2+2*a6*Zp^3-a3*Yp*Zp^2)/(2*Yp*Zp^2+a1*Xp*Zp^2+a3*Zp^3);
      L := Curve(R2,Y - l*X - n*Z);

  elif (Xp eq Xq) and (Zp eq Zq) and ((Yp eq - Yq-a1*Xq-a3*Zq) or (Yq eq - Yp-a1*Xp-a3*Zp)) then

      L := Curve(R2,X*Zp - Xp*Z);

  elif P eq O then

      L := Curve(R2,X*Zq - Xq*Z);

  elif Q eq O then

      L := Curve(R2,X*Zp - Xp*Z);

  else
      l := (Zp*Yq - Yp*Zq)/(Xq*Zp-Xp*Zq); n := (Yp*Xq - Yq*Xp)/(Xq*Zp-Xp*Zq);
      L := Curve(R2,Y - l*X - n*Z);
  end if;

```

```
  // Get the points in the intersection
  pts := Points(Intersection(C,L));

  // This should only contain R
  if P eq Q then
    R_lst := pts diff Seqset([P,Q,O]);
  else
    R_lst := pts diff Seqset([P,Q]);
  end if;

  // In case we have a special situation, we need this.
  if #R_lst ne 1 then
    nmrs := [IntersectionNumber(L,C,pts[i]) : i in [1..#pts]];
    i := Index(nmrs,2);
    if i ne 0 then
      R := pts[i];
    else
      return "Error: something went wrong in the intersection.";
    end if;
  else
    R := R_lst[1];
  end if;

  if R eq O then
    return R;
  else
    Xr := R[1]; Zr := R[3];
    Lp := Curve(R2,X*Zr - Xr*Z);
    pts := Points(Intersection(C,Lp));
    Rp_lst := pts diff Seqset([R,O]);
    if #Rp_lst ne 1 then
      nmrs := [IntersectionNumber(Lp,C,pts[i]) : i in [1..#pts]];
      i := Index(nmrs,2);
      if i ne 0 then
        Rp := pts[i];
      else
        return "Error: something went wrong in the intersection.";
      end if;
    else
      Rp := Rp_lst[1];
    end if;
  end if;

  return Rp;

end function;
```

Function 4: `Ew_AddL_geom`, mentioned in §1.2.1

```
Ead_AddL_geom := function(BF,lst,P,Q)

  R1<a,d> := FunctionField(BF,2);
  R2<X,Y,Z> := ProjectiveSpace(R1,2);
  Op := R2 ! [0,-1,1]; O1 := R2![1,0,0]; O2 := R2![0,1,0];

  // Get the coeffs
  if #lst eq 2 then
    a := R1!lst[1]; d := R1!lst[2];
  else
    return "Error: wrong number of coeffs";
  end if;

  // Get the coordinates
  Xp := R1 ! P[1]; Yp := R1 ! P[2]; Zp := R1 ! P[3]; P := R2![Xp,Yp,Zp];
  Xq := R1 ! Q[1]; Yq := R1 ! Q[2]; Zq := R1 ! Q[3]; Q := R2![Xq,Yq,Zq];

  // Define the Ead curve
  Crv := Curve(R2, (a*(X^2) + (Y^2))*Z^2 - Z^4 - d*(X^2)*(Y^2));

  // Compute the coeffs of the conic and define it
  // We define the conic as a curve because in some cases its polynomial has a degree too low for the
     magma command Conic to work
  if P eq Op then
```

```
    cz := -Xq; cxy := Zq; cxz := Zq;
    Cnc := Curve(R2,cz*(Z^2 + Y*Z)+cxy*X*Y+cxz*X*Z);

  elif Q eq Op then
    cz := -Xp; cxy := Zp; cxz := Zp;
    Cnc := Curve(R2,cz*(Z^2 + Y*Z)+cxy*X*Y+cxz*X*Z);

  elif P eq Q then
    cz := Xp*Zp*(Zp-Yp);
    cxy := d*(Xp^2)*Yp-Zp^3;
    cxz := Zp*(Zp*Yp - a*Xp^2);
    Cnc := Curve(R2,cz*(Z^2 + Y*Z)+cxy*X*Y+cxz*X*Z);

  else
    cz := Xp*Xq*(Yp*Zq-Yq*Zp);
    cxy := Zp*Zq*(Xp*Zq-Xq*Zp + Xp*Yq - Xq*Yp);
    cxz := Xq*Yq*(Zp^2)-Xp*Yp*(Zq^2)+Yp*Yq*(Xq*Zp-Xp*Zq);
    Cnc := Curve(R2,cz*(Z^2 + Y*Z)+cxy*X*Y+cxz*X*Z);

  end if;

  // Get the points in the intersection
  pts := Points(Intersection(Crv,Cnc));

  // This should only contain -P3
  P3_lst := pts diff Seqset([P,Q,Op,O1,O2]);

  // In case we have a special situation, we need this.
  if #P3_lst ne 1 then
    og_pts := [P,Q,Op,O1,O2]; nmrs := [1,1,1,2,2];
    if #pts eq #Seqset(og_pts) then
      if P eq Op then nmrs[3] +:= 1; end if;
      if Q eq Op then nmrs[3] +:= 1; end if;
      for i in [1..2] do
        if IntersectionNumber(Cnc,Crv,og_pts[i]) gt nmrs[i] then
          return [-og_pts[i][1],og_pts[i][2],og_pts[i][3]];
        end if;
      end for;
      for i in [3..5] do
        if IntersectionNumber(Cnc,Crv,og_pts[i]) gt nmrs[i] then
          return og_pts[i];
        end if;
      end for;
    return "Error: something went wrong in the intersection.";
    end if;
  end if;

  // Note that the output is the opposite of the 8th intersection point.
  P3 := [-P3_lst[1][1], P3_lst[1][2], P3_lst[1][3]];

  return P3;

end function;
```

Function 5: `Ead_AddL_geom`, mentioned in §3.3.1

A.3 Functions for point addition: from explicit formulae

```
Ew_AddL_eval := function(BF,lst,P,Q)

  // Things are a bit different for decimal numbers, so we need a flag
  if (Type(BF) eq Type(RealField())) or (Type(BF) eq Type(ComplexField())) then
    fl := true;
  else
    fl := false;
  end if;

  if fl then
    R1<a1,a2,a3,a4,a6> := PolynomialRing(BF,5);
  else
    R1<a1,a2,a3,a4,a6> := FunctionField(BF,5);
  end if;
    R2<X1,Y1,Z1, X2,Y2,Z2> := PolynomialRing(R1,6);
```

```
16
17  //Get the coeffs
18  if #lst eq 2 then
19      a1 := R1 ! 0; a2 := R1 ! 0; a3 := R1 ! 0;
20      a4 := R1 ! lst[1]; a6 := R1 ! lst[2];
21  elif #lst eq 5 then
22      a1 := R1 ! lst[1]; a2 := R1 ! lst[2]; a3 := R1 ! lst[3];
23      a4 := R1 ! lst[4]; a6 := R1 ! lst[5];
24  else
25      return "Error: wrong number of coeffs";
26  end if;
27
28  // Get the points
29  Xp := R2!P[1]; Yp := R2!P[2]; Zp := R2!P[3];
30  Xq := R2!Q[1]; Yq := R2!Q[2]; Zq := R2!Q[3];
31
32
33  // Choose which to use
34  if fl then
35    if (Abs(BF!(Xp - Xq)) gt 1E-10) and (Abs(BF!(Yp - Yq)) gt 1E-10) and (Abs(BF!(Zp - Zq)) gt 1E-10) then
36        //here we use (a,b,c) = (0,1,0)
37
38        X3 := R2!(-a1*a2*Xp^2*Xq^2 - a2*Xp^2*Xq*Yq + (-a1^2*a3 - 2*a1*a4)*Xp^2*Xq*Zq + (-a1*a3 - a4)*Xp^2*
     Yq*Zq + (-a1*a3^2 - 3*a1*a6)*Xp^2*Zq^2 + (a1^2 - a2)*Xp*Yp*Xq^2 + 2*a1*Xp*Yp*Xq*Yq - 2*a4*Xp*Yp*Xq*Zq
      + Xp*Yp*Yq^2 + (-a3^2 - 3*a6)*Xp*Yp*Zq^2 + (-a1*a4 - a2*a3)*Xp*Zp*Xq^2 - 2*a4*Xp*Zp*Xq*Yq + (-2*a1*
     a3^2 - 6*a1*a6 - 2*a3*a4)*Xp*Zp*Xq*Zq + (-2*a3^2 - 6*a6)*Xp*Zp*Yq*Zq + (-a1^3*a6 + a1^2*a3*a4 - a1*a2
     *a3^2 - 4*a1*a2*a6 + a1*a4^2 - a3^3 - 3*a3*a6)*Xp*Zp*Zq^2 + a1*Yp^2*Xq^2 + Yp^2*Xq*Yq + a3*Yp^2*Xq*Zq
      + (a1*a3 - a4)*Yp*Zp*Xq^2 + 2*a3*Yp*Zp*Xq*Yq - 6*a6*Yp*Zp*Xq*Zq + (-a1^2*a6 + a1*a3*a4 - a2*a3^2 -
     4*a2*a6 + a4^2)*Yp*Zp*Zq^2 - a3*a4*Zp^2*Xq^2 - 3*a6*Zp^2*Xq*Yq + (-a3^3 - 6*a3*a6)*Zp^2*Xq*Zq + (-a1
     ^2*a6 + a1*a3*a4 - a2*a3^2 - 4*a2*a6 + a4^2)*Zp^2*Yq*Zq + (-a1^2*a3*a6 + a1*a3^2*a4 - a2*a3^3 - 4*a2*
     a3*a6 + a3*a4^2)*Zp^2*Zq^2);
39
40        Y3 := R2!((-a2^2 + 3*a4)*Xp^2*Xq^2 + (a1^2*a4 - 2*a1*a2*a3 - a2*a4 + 3*a3^2 + 9*a6)*Xp^2*Xq*Zq +
     (3*a1^2*a6 - 2*a1*a3*a4 + a2*a3^2 + 3*a2*a6 - a4^2)*Xp^2*Zq^2 + (a1*a2 - 3*a3)*Xp*Yp*Xq^2 + (2*a1*a4
     - 2*a2*a3)*Xp*Yp*Xq*Zq + (3*a1*a6 - a3*a4)*Xp*Yp*Zq^2 + (-a2*a4 + 9*a6)*Xp*Zp*Xq^2 + (6*a1^2*a6 - 4*
     a1*a3*a4 + 2*a2*a3^2 + 12*a2*a6 - 4*a4^2)*Xp*Zp*Xq*Zq + (a1^4*a6 - a1^3*a3*a4 + a1^2*a2*a3^2 + 5*a1
     ^2*a2*a6 - a1^2*a4^2 - a1*a2*a3*a4 - a1*a3^3 - 3*a1*a3*a6 + a2^2*a3^2 + 4*a2^2*a6 - a2*a4^2 - a3^2*a4
      - 3*a4*a6)*Xp*Zp*Zq^2 + a1*Yp^2*Xq*Yq + Yp^2*Yq^2 + a3*Yp^2*Yq*Zq + (a1*a4 - a2*a3)*Yp*Zp*Xq^2 + (6*
     a1*a6 - 2*a3*a4)*Yp*Zp*Xq*Zq + (a1^3*a6 - a1^2*a3*a4 + a1*a2*a3^2 + 4*a1*a2*a6 - a1*a4^2 - a3^3 - 3*
     a3*a6)*Yp*Zp*Zq^2 + (3*a2*a6 - a4^2)*Zp^2*Xq^2 + (a1^2*a2*a6 - a1*a2*a3*a4 + 3*a1*a3*a6 + a2^2*a3^2 +
      4*a2^2*a6 - a2*a4^2 - 2*a3^2*a4 - 3*a4*a6)*Zp^2*Xq*Zq + (a1^3*a3*a6 - a1^2*a3^2*a4 + a1^2*a4*a6 + a1
     *a2*a3^3 + 4*a1*a2*a3*a6 - 2*a1*a3*a4^2 + a2*a3^2*a4 + 4*a2*a4*a6 - a3^4 - 6*a3^2*a6 - a4^3 - 9*a6^2)
     *Zp^2*Zq^2);
41
42        Z3 := R2!(3*a1*Xp^2*Xq^2 + 3*Xp^2*Xq*Yq + (a1^3 + 2*a1*a2)*Xp^2*Xq*Zq + (a1^2 + a2)*Xp^2*Yq*Zq + (
     a1^2*a3 + a1*a4)*Xp^2*Zq^2 + 3*Xp*Yp*Xq^2 + (2*a1^2 + 2*a2)*Xp*Yp*Xq*Zq + 2*a1*Xp*Yp*Yq*Zq + (2*a1*a3
      + a4)*Xp*Yp*Zq^2 + (a1*a2 + 3*a3)*Xp*Zp*Xq^2 + 2*a2*Xp*Zp*Xq*Yq + (2*a1^2*a3 + 2*a1*a4 + 2*a2*a3)*Xp
     *Zp*Xq*Zq + (2*a1*a3 + 2*a4)*Xp*Zp*Yq*Zq + (2*a1*a3^2 + 3*a1*a6 + a3*a4)*Xp*Zp*Zq^2 + a1*Yp^2*Xq*Zq +
      Yp^2*Yq*Zq + a3*Yp^2*Zq^2 + a2*Yp*Zp*Xq^2 + (2*a1*a3 + 2*a4)*Yp*Zp*Xq*Zq + Yp*Zp*Yq^2 + 2*a3*Yp*Zp*
     Yq*Zq + (2*a3^2 + 3*a6)*Yp*Zp*Zq^2 + a2*a3*Zp^2*Xq^2 + a4*Zp^2*Xq*Yq + (a1*a3^2 + 2*a3*a4)*Zp^2*Xq*Zq
      + (a3^2 + 3*a6)*Zp^2*Yq*Zq + (a3^3 + 3*a3*a6)*Zp^2*Zq^2);
43
44      else
45        //here we use (a,b,c) = (0,0,1)
46        X3 := R2!(-a2*Xp^2*Xq*Zq + a1*Xp^2*Yq*Zq - a4*Xp^2*Zq^2 + 2*Xp*Yp*Yq*Zq + a3*Xp*Yp*Zq^2 + a2*Xp*Zp
     *Xq^2 + Xp*Zp*Yq^2 + 2*a3*Xp*Zp*Yq*Zq - 3*a6*Xp*Zp*Zq^2 - Yp^2*Xq*Zq - a1*Yp*Zp*Xq^2 - 2*Yp*Zp*Xq*Yq
     - 2*a3*Yp*Zp*Xq*Zq + a4*Zp^2*Xq^2 - a3*Zp^2*Xq*Yq + 3*a6*Zp^2*Xq*Zq);
47
48        Y3 := R2!(-3*Xp^2*Xq*Yq + (a1*a2 - 3*a3)*Xp^2*Xq*Zq + (-a1^2 - a2)*Xp^2*Yq*Zq + (a1*a4 - a2*a3)*Xp
     ^2*Zq^2 + 3*Xp*Yp*Xq^2 + 2*a2*Xp*Yp*Xq*Zq - 2*a1*Xp*Yp*Yq*Zq + a4*Xp*Yp*Zq^2 + (-a1*a2 + 3*a3)*Xp*Zp*
     Xq^2 - 2*a2*Xp*Zp*Xq*Yq + (-2*a1*a3 - 2*a4)*Xp*Zp*Yq*Zq + (3*a1*a6 - a3*a4)*Xp*Zp*Zq^2 - Yp^2*Yq*Zq +
      (a1^2 + a2)*Yp*Zp*Xq^2 + 2*a1*Yp*Zp*Xq*Yq + (2*a1*a3 + 2*a4)*Yp*Zp*Xq*Zq + Yp*Zp*Yq^2 + (a3^2 + 3*a6
     )*Yp*Zp*Zq^2 + (-a1*a4 + a2*a3)*Zp^2*Xq^2 - a4*Zp^2*Xq*Yq + (-3*a1*a6 + a3*a4)*Zp^2*Xq*Zq + (-a3^2 -
     3*a6)*Zp^2*Yq*Zq);
49
50        Z3 := R2!(3*Xp^2*Xq*Zq + a2*Xp^2*Zq^2 - a1*Xp*Yp*Zq^2 - 3*Xp*Zp*Xq^2 + a4*Xp*Zp*Zq^2 - Yp^2*Zq^2 -
      a3*Yp*Zp*Zq^2 - a2*Zp^2*Xq^2 + a1*Zp^2*Xq*Yq - a4*Zp^2*Xq*Zq + Zp^2*Yq^2 + a3*Zp^2*Yq*Zq);
51
52      end if;
53  else
54    if (Xp eq Xq) and (Yp eq Yq) and (Zp eq Zq) then
55      //here we use (a,b,c) = (0,1,0)
56
```

```
      X3 := R2!(-a1*a2*Xp^2*Xq^2 - a2*Xp^2*Xq*Yq + (-a1^2*a3 - 2*a1*a4)*Xp^2*Xq*Zq + (-a1*a3 - a4)*Xp^2*Yq
    *Zq + (-a1*a3^2 - 3*a1*a6)*Xp^2*Zq^2 + (a1^2 - a2)*Xp*Yp*Xq^2 + 2*a1*Xp*Yp*Xq*Yq - 2*a4*Xp*Yp*Xq*Zq +
     Xp*Yp*Yq^2 + (-a3^2 - 3*a6)*Xp*Yp*Zq^2 + (-a1*a4 - a2*a3)*Xp*Zp*Xq^2 - 2*a4*Xp*Zp*Xq*Yq + (-2*a1*a3
    ^2 - 6*a1*a6 - 2*a3*a4)*Xp*Zp*Xq*Zq + (-2*a3^2 - 6*a6)*Xp*Zp*Yq*Zq + (-a1^3*a6 + a1^2*a3*a4 - a1*a2*
    a3^2 - 4*a1*a2*a6 + a1*a4^2 - a3^3 - 3*a3*a6)*Xp*Zp*Zq^2 + a1*Yp^2*Xq^2 + Yp^2*Xq*Yq + a3*Yp^2*Xq*Zq
    + (a1*a3 - a4)*Yp*Zp*Xq^2 + 2*a3*Yp*Zp*Xq*Yq - 6*a6*Yp*Zp*Xq*Zq + (-a1^2*a6 + a1*a3*a4 - a2*a3^2 - 4*
    a2*a6 + a4^2)*Yp*Zp*Zq^2 - a3*a4*Zp^2*Xq^2 - 3*a6*Zp^2*Xq*Yq + (-a3^3 - 6*a3*a6)*Zp^2*Xq*Zq + (-a1^2*
    a6 + a1*a3*a4 - a2*a3^2 - 4*a2*a6 + a4^2)*Zp^2*Yq*Zq + (-a1^2*a3*a6 + a1*a3^2*a4 - a2*a3^3 - 4*a2*a3*
    a6 + a3*a4^2)*Zp^2*Zq^2);

      Y3 := R2!((-a2^2 + 3*a4)*Xp^2*Xq^2 + (a1^2*a4 - 2*a1*a2*a3 - a2*a4 + 3*a3^2 + 9*a6)*Xp^2*Xq*Zq + (3*
    a1^2*a6 - 2*a1*a3*a4 + a2*a3^2 + 3*a2*a6 - a4^2)*Xp^2*Zq^2 + (a1*a2 - 3*a3)*Xp*Yp*Xq^2 + (2*a1*a4 -
    2*a2*a3)*Xp*Yp*Xq*Zq + (3*a1*a6 - a3*a4)*Xp*Yp*Zq^2 + (-a2*a4 + 9*a6)*Xp*Zp*Xq^2 + (6*a1^2*a6 - 4*a1*
    a3*a4 + 2*a2*a3^2 + 12*a2*a6 - 4*a4^2)*Xp*Zp*Xq*Zq + (a1^4*a6 - a1^3*a3*a4 + a1^2*a2*a3^2 + 5*a1^2*a2
    *a6 - a1^2*a4^2 - a1*a2*a3*a4 - a1*a3^3 - 3*a1*a3*a6 + a2^2*a3^2 + 4*a2^2*a6 - a2*a4^2 - a3^2*a4 - 3*
    a4*a6)*Xp*Zp*Zq^2 + a1*Yp^2*Xq*Yq + Yp^2*Yq^2 + a3*Yp^2*Yq*Zq + (a1*a4 - a2*a3)*Yp*Zp*Xq^2 + (6*a1*a6
     - 2*a3*a4)*Yp*Zp*Xq*Zq + (a1^3*a6 - a1^2*a3*a4 + a1*a2*a3^2 + 4*a1*a2*a6 - a1*a4^2 - a3^3 - 3*a3*a6)
    *Yp*Zp*Zq^2 + (3*a2*a6 - a4^2)*Zp^2*Xq^2 + (a1^2*a2*a6 - a1*a2*a3*a4 + 3*a1*a3*a6 + a2^2*a3^2 + 4*a2
    ^2*a6 - a2*a4^2 - 2*a3^2*a4 - 3*a4*a6)*Zp^2*Xq*Zq + (a1^3*a3*a6 - a1^2*a3^2*a4 + a1^2*a4*a6 + a1*a2*
    a3^3 + 4*a1*a2*a3*a6 - 2*a1*a3*a4^2 + a2*a3^2*a4 + 4*a2*a4*a6 - a3^4 - 6*a3^2*a6 - a4^3 - 9*a6^2)*Zp
    ^2*Zq^2);

      Z3 := R2!(3*a1*Xp^2*Xq^2 + 3*Xp^2*Xq*Yq + (a1^3 + 2*a1*a2)*Xp^2*Xq*Zq + (a1^2 + a2)*Xp^2*Yq*Zq + (a1
    ^2*a3 + a1*a4)*Xp^2*Zq^2 + 3*Xp*Yp*Xq^2 + (2*a1^2 + 2*a2)*Xp*Yp*Xq*Zq + 2*a1*Xp*Yp*Yq*Zq + (2*a1*a3 +
     a4)*Xp*Yp*Zq^2 + (a1*a2 + 3*a3)*Xp*Zp*Xq^2 + 2*a2*Xp*Zp*Xq*Yq + (2*a1^2*a3 + 2*a1*a4 + 2*a2*a3)*Xp*
    Zp*Xq*Zq + (2*a1*a3 + 2*a4)*Xp*Zp*Yq*Zq + (2*a1*a3^2 + 3*a1*a6 + a3*a4)*Xp*Zp*Zq^2 + a1*Yp^2*Xq*Zq +
    Yp^2*Yq*Zq + a3*Yp^2*Zq^2 + a2*Yp*Zp*Xq^2 + (2*a1*a3 + 2*a4)*Yp*Zp*Xq*Zq + Yp*Zp*Yq^2 + 2*a3*Yp*Zp*Yq
    *Zq + (2*a3^2 + 3*a6)*Yp*Zp*Zq^2 + a2*a3*Zp^2*Xq^2 + a4*Zp^2*Xq*Yq + (a1*a3^2 + 2*a3*a4)*Zp^2*Xq*Zq +
     (a3^2 + 3*a6)*Zp^2*Yq*Zq + (a3^3 + 3*a3*a6)*Zp^2*Zq^2);
   else
      //here we use (a,b,c) = (0,0,1)

      X3 := R2!(-a2*Xp^2*Xq*Zq + a1*Xp^2*Yq*Zq - a4*Xp^2*Zq^2 + 2*Xp*Yp*Yq*Zq + a3*Xp*Yp*Zq^2 + a2*Xp*Zp
    *Xq^2 + Xp*Zp*Yq^2 + 2*a3*Xp*Zp*Yq*Zq - 3*a6*Xp*Zp*Zq^2 - Yp^2*Xq*Zq - a1*Yp*Zp*Xq^2 - 2*Yp*Zp*Xq*Yq
    - 2*a3*Yp*Zp*Xq*Zq + a4*Zp^2*Xq^2 - a3*Zp^2*Xq*Yq + 3*a6*Zp^2*Xq*Zq);

      Y3 := R2!(-3*Xp^2*Xq*Yq + (a1*a2 - 3*a3)*Xp^2*Xq*Zq + (-a1^2 - a2)*Xp^2*Yq*Zq + (a1*a4 - a2*a3)*Xp
    ^2*Zq^2 + 3*Xp*Yp*Xq^2 + 2*a2*Xp*Yp*Xq*Zq - 2*a1*Xp*Yp*Yq*Zq + a4*Xp*Yp*Zq^2 + (-a1*a2 + 3*a3)*Xp*Zp*
    Xq^2 - 2*a2*Xp*Zp*Xq*Yq + (-2*a1*a3 - 2*a4)*Xp*Zp*Yq*Zq + (3*a1*a6 - a3*a4)*Xp*Zp*Zq^2 - Yp^2*Yq*Zq +
     (a1^2 + a2)*Yp*Zp*Xq^2 + 2*a1*Yp*Zp*Xq*Yq + (2*a1*a3 + 2*a4)*Yp*Zp*Xq*Zq + Yp*Zp*Yq^2 + (a3^2 + 3*a6
    )*Yp*Zp*Zq^2 + (-a1*a4 + a2*a3)*Zp^2*Xq^2 - a4*Zp^2*Xq*Yq + (-3*a1*a6 + a3*a4)*Zp^2*Xq*Zq + (-a3^2 -
    3*a6)*Zp^2*Yq*Zq);

      Z3 := R2!(3*Xp^2*Xq*Zq + a2*Xp^2*Zq^2 - a1*Xp*Yp*Zq^2 - 3*Xp*Zp*Xq^2 + a4*Xp*Zp*Zq^2 - Yp^2*Zq^2 -
     a3*Yp*Zp*Zq^2 - a2*Zp^2*Xq^2 + a1*Zp^2*Xq*Yq - a4*Zp^2*Xq*Zq + Zp^2*Yq^2 + a3*Zp^2*Yq*Zq);

   end if;
  end if;

   return [X3,Y3,Z3];
end function;
```

Function 6: `Ew_AddL_eval`, discussed in §1.2.3

```
Ead_AddL_eval := function(BF,lst,P,Q)

  // Things are a bit different for decimal numbers, so we need a flag for it
  if (Type(BF) eq Type(RealField())) or (Type(BF) eq Type(ComplexField())) then
    fl := true;
  else
    fl := false;
  end if;

  if fl then
    R1<a,d> := PolynomialRing(BF,2);
  else
    R1<a,d> := FunctionField(BF,2);
  end if;
  R2<X1,Y1,Z1,T1, X2,Y2,Z2,T2> := PolynomialRing(R1,8);

  // Get the coeffs
  if #lst eq 2 then
      a := R2!lst[1]; d := R2!lst[2];
```

```
  else
      return "Error: wrong number of coeffs";
  end if;

  // Get the points
  Xp := R2 ! P[1][1]; Zp := R2 ! P[1][2]; Yp := R2 ! P[2][1]; Tp := R2 ! P[2][2];
  Xq := R2 ! Q[1][1]; Zq := R2 ! Q[1][2]; Yq := R2 ! Q[2][1]; Tq := R2 ! Q[2][2];

  // Compute the sums
  X_o := Evaluate(R2 ! X1*Y2*Z2*T1+X2*Y1*Z1*T2,[Xp,Yp,Zp,Tp,Xq,Yq,Zq,Tq]);
  Z_o := Evaluate(R2 ! Z1*Z2*T1*T2+d*X1*X2*Y1*Y2,[Xp,Yp,Zp,Tp,Xq,Yq,Zq,Tq]);
  Y_o := Evaluate(R2 ! Y1*Y2*Z1*Z2-a*X1*X2*T1*T2,[Xp,Yp,Zp,Tp,Xq,Yq,Zq,Tq]);
  T_o := Evaluate(R2 ! Z1*Z2*T1*T2-d*X1*X2*Y1*Y2,[Xp,Yp,Zp,Tp,Xq,Yq,Zq,Tq]);

  X_d := Evaluate(R2 ! X1*Y1*Z2*T2+X2*Y2*Z1*T1,[Xp,Yp,Zp,Tp, Xq,Yq,Zq,Tq]);
  Z_d := Evaluate(R2 ! a*X1*X2*T1*T2+Y1*Y2*Z1*Z2,[Xp,Yp,Zp,Tp, Xq,Yq,Zq,Tq]);
  Y_d := Evaluate(R2 ! X1*Y1*Z2*T2-X2*Y2*Z1*T1,[Xp,Yp,Zp,Tp, Xq,Yq,Zq,Tq]);
  T_d := Evaluate(R2 ! X1*Y2*Z2*T1-X2*Y1*Z1*T2,[Xp,Yp,Zp,Tp, Xq,Yq,Zq,Tq]);

  // Choose which to use
  if fl then
      if (Abs(BF!X_o) gt 1E-10 or Abs(BF!Z_o) gt 1E-10) and (Abs(BF!Y_o) gt 1E-10 or Abs(BF!T_o) gt 1E-10)
      then
          X := X_o; Y := Y_o; Z := Z_o; T := T_o;
     elif (Abs(BF!X_d) gt 1E-10 or Abs(BF!Z_d) gt 1E-10) and (Abs(BF!Y_d) gt 1E-10 or Abs(BF!T_d) gt 1E
     -10) then
      X := X_d; Y := Y_d; Z := Z_d; T := T_d;
    else
      return "Error: all are zero";
    end if;
  else
    if (X_o ne 0 or Z_o ne 0) and (Y_o ne 0 or T_o ne 0) then
      X := X_o; Y := Y_o; Z := Z_o; T := T_o;
    elif (X_d ne 0 or Z_d ne 0) and (Y_d ne 0 or T_d ne 0) then
      X := X_d; Y := Y_d; Z := Z_d; T := T_d;
    else
      return "Error: all are zero";
    end if;
  end if;

    return [[X,Z],[Y,T]];
end function;
```

Function 7: `Ead_AddL_eval`, discussed in §3.3.2

```
ProjCoord := function(PR, p)
  return [[PR ! p[1], PR ! 1],[PR ! p[2],PR ! 1]];
end function;
```

Function 8: `ProjCoord`, mentioned in 3.3.2

```
AffCoord := function(BF,P)
  if (P[1][2] eq 0) or (P[2][2] eq 0) then
    return "Error: point at infinity";
  else
    return [BF ! (P[1][1] div P[1][2]), BF ! (P[2][1]div P[2][2])];
  end if;
end function;
```

Function 9: `AffCoord`, mentioned in 3.3.2

B Maps between models

B.1 Edwards - Weierstrass birational equivalences

```
Ew2Ed_P4_symb := function(BF,lst,P4 : PP4 := [0,0])
  // lst := [a1,a2,a3,a4,a6] can be numerical coeffs or letters

  R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
  Rw_var<u,v> := FunctionField(R,2);

  a1 := R ! lst[1]; a2 := (R ! lst[2]); a3 := (R ! lst[3]); a4 := (R ! lst[4]); a6 := (R ! lst[5]);
```

```
  up := (R ! P4[1]); vp := (R ! P4[2]);

  u2p := R ! PP4[1]; v2p := R ! PP4[2];

  up +:= u2p; vp +:= (a1*up + a3)/2;
  u +:= u2p; v +:= (a1*u + a3)/2;

  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
  a2 +:= -3*u2p + (a1^2)/4;

  d := 1-4*(up^3)/(vp^2);
  c_x := (vp*u)/(up*v); c_y := (u-up)/(u+up);

  return [*[d],[c_x,c_y]*];

end function;
```

Function 10: `Ew2Ed_P4_symb`, discussed in §2.3

```
Ed2Ew_P4_symb := function(BF,lst,P4 : PP4 := [0,0], aa1 := 0, aa3 := 0)
  //lst := [d] can be numerical coeff or letter

  R<a1,a2,a3,a4,a6,up,vp,d> := FunctionField(BF,8);
  Rd_var<x,y> := FunctionField(R,2);

  d := R ! lst[1];
  up := R ! P4[1]; vp := R ! P4[2];
  u2p := R ! PP4[1]; v2p := R ! PP4[2];

  a1 := R ! aa1; a2 := 2*(1+d)/(1-d)*up; a3 := R ! aa3; a4 := up^2; a6 := 0;
  c_u := up*(1+y)/(1-y); c_v := vp*(1+y)/(x*(1-y));

  c_u -:= u2p; c_v -:= (a1*(c_u-u2p)+a3)/2;
  a2 -:= -3*u2p + (a1^2)/4;
  a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
  a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

  return [*[a1,a2,a3,a4,a6],[c_u,c_v]*];

end function;
```

Function 11: `Ed2Ew_P4_symb`, discussed in §2.3

```
Ew2Ed_symb := function(BF,lst)
  // lst := [a1,a2,a3,a4,a6] can be numerical coeffs or letters

  R<a1,a2,a3,a4,a6,d> := FunctionField(BF,6);
  Rw_var<u,v> := FunctionField(R,2);

  a1 := R ! lst[1]; a2 := (R ! lst[2]);a3 := (R ! lst[3]);a4 := (R ! lst[4]);a6 := (R ! lst[5]);

  d := (a2/2)-1;
  c_x := -2*u/v; c_y := (v^2-(2+2*d)*u^2 - 2*u^3)/(4*d*u^2-v^2);

  return [*[d],[c_x,c_y]*];

end function;
```

Function 12: `Ew2Ed_symb`, discussed in §2.3

```
Ed2Ew_symb := function(BF,lst)
  //lst := [d] can be numerical coeff or letter

  R<a1,a2,a3,a4,a6,d> := FunctionField(BF,6);
  Rd_var<x,y> := FunctionField(R,2);

  d := R ! lst[1];

  a1 := 0; a2 := 2*(d+1); a3 := 0; a4 := (d-1)^2; a6 := 0;

  A := 2*y - (2*d*y + d +1)*x^2 + 2;
  c_u := A/(x^2); c_v := -2*A/(x^3);

```

```
  return [*[a1,a2,a3,a4,a6],[c_u,c_v]*];

end function;
```

Function 13: `Ed2Ew_symb`, discussed in §2.3

```
Ew2Ed_P4_eval := function(BF,lst,P4,p : PP4 := [0,0])

  if #lst eq 5 then
      a1 := BF!lst[1]; a2 := BF!lst[2]; a3 := BF!lst[3]; a4 := BF!lst[4]; a6 := BF!lst[5];
  elif #lst eq 2 then
      a1 := BF!0; a2 := BF!0; a3 := BF!0; a4 := BF!lst[1]; a6 := BF!lst[2];
  else
      return "Error: wrong number of coeffs";
  end if;
  u2p := BF ! PP4[1];
  up := (BF ! P4[1]) + u2p; vp := (BF ! P4[2]) + (a1*up + a3)/2;
  u := BF!p[1] + u2p; v := BF!p[2] + (a1*u + a3)/2;
  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - (a1^2)*u2p/2; a2 +:= - 3*u2p + (a1^2)/4;

  denom_x := up*v; denom_y := (u+up); denom_d := vp^2;
  if denom_x eq 0 then
      return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
      return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  elif denom_y eq 0 then
      return "Error: birat. eq. not defined for this point since (denom_d eq 0).";
  else
      d := 1-4*(up^3)/(vp^2);
      c_x := vp*u/denom_x; c_y := (u-up)/denom_y;

      return [*[BF|d],[BF|c_x,c_y]*];
  end if;
end function;
```

Function 14: `Ew2Ed_P4_eval`, discussed in §2.3

```
Ed2Ew_P4_eval := function(BF,lst,P4,p : PP4 := [0,0], a1 := 0, a3 := 0)

  up := (BF ! P4[1]) + BF!PP4[1]; vp := (BF ! P4[2]) + (a1*up + a3)/2;
  u2p := BF ! PP4[1]; v2p := BF ! PP4[2];
  d := BF!lst[1]; x := BF!p[1]; y := BF!p[2];

  denom_x := (1-y); denom_y := x*(1-y);
  if denom_x eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  else
    a2 := 2*(1+d)/(1-d)*up; a4 := up^2; a6 := 0;
    c_u := up*(1+y)/denom_x; c_v:= vp*(1+y)/denom_y;

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;

    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

    return [*[a1,a2,a3,a4,a6],[c_u,c_v]*];
  end if;

end function;
```

Function 15: `Ed2Ew_P4_eval`, discussed in §2.3

```
Ew2Ed_eval := function(BF,lst,p: PP4 := [0,0])

  if #lst eq 5 then
      a1 := BF!lst[1]; a2 := BF!lst[2]; a3 := BF!lst[3]; a4 := BF!lst[4]; a6 := BF!lst[5];
  elif #lst eq 2 then
      a1 := BF!0; a2 := BF!0; a3 := BF!0; a4 := BF!lst[1]; a6 := BF!lst[2];
  else
      return "Error: wrong number of coeffs";
```

```
  end if;
  u := BF!p[1]; v := BF!p[2];

  u2p := BF ! PP4[1];
  u := BF!p[1] + u2p; v := BF!p[2] + (a1*u + a3)/2;
  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - (a1^2)*u2p/2; a2 +:= - 3*u2p + (a1^2)/4;

  if a2 ne 0 then
      d := (a2/2)-1;
  elif (a2 eq 0) and (IsSquare(BF!(a4))) then
      d := Sqrt(a4) + 1;
  else
      return "Error: d is a not a square over the basis field";
  end if;

  denom_x := v; denom_y := 2*(a2-2)*u^2-v^2;
  if denom_x eq 0 then
      return "Error: map not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
      return "Error: map not defined for this point since (denom_y eq 0).";
  else
      c_x := -2*u/denom_x; c_y := (v^2-a2*u^2 - 2*u^3)/denom_y;

      return [*[d],[c_x,c_y]*];
  end if;

end function;
```

Function 16: `Ew2Ed_eval`, discussed in §2.3

```
Ed2Ew_eval := function(BF,lst,p: PP4 := [0,0], a1 := 0, a3 := 0)

  d := BF!lst[1]; x := BF!p[1]; y := BF!p[2]; u2p := BF!PP4[1];

  denom_x := x^2; denom_y := x^3;
  if denom_x eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  else
    a2 := 2*(d+1); a4 := (d-1)^2; a6 := 0;
    A := 2*y - (2*d*y + d +1)*x^2 + 2;
    c_u := A/(x^2); c_v:= -2*A/(x^3);

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;

    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;


    return [*[a1,a2,a3,a4,a6],[c_u,c_v]*];
  end if;

end function;
```

Function 17: `Ed2Ew_eval`, discussed in §2.3

B.2 Twisted Edwards - Weierstrass birational equivalences

```
Ew2Ead_P4_eval := function(BF,lst,P4,p : PP4 := [0,0], a := 1)

  // If 'a' is not a square on BF, then we need to work on an extension field
  tf, sqrt_a := IsSquare(BF!a);
  if not tf then
    // QF_int := Integers()!(Numerator(a)*Denominator(a));
    // BF<sqrt_a> := QuadraticField(QF_int);
    BF := AlgebraicClosure(BF);
    tf, sqrt_a := IsSquare(BF!a);
    // set a flag that we changed base field?;
  end if;

  a := BF!a;
```

```
  if #lst eq 5 then
        a1 := BF!lst[1]; a2 := BF!lst[2]; a3 := BF!lst[3]; a4 := BF!lst[4]; a6 := BF!lst[5];
    elif #lst eq 2 then
        a1 := BF!0; a2 := BF!0; a3 := BF!0; a4 := BF!lst[1]; a6 := BF!lst[2];
    else
        return "Error: wrong number of coeffs";
    end if;
  u2p := BF ! PP4[1];
  up := (BF ! P4[1]) + u2p; vp := (BF ! P4[2]) + (a1*up + a3)/2;
  u := BF!p[1] + u2p; v := BF!p[2] + (a1*u + a3)/2;
  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - (a1^2)*u2p/2; a2 +:= - 3*u2p + (a1^2)/4;

  denom_x := up*v; denom_y := (u+up); denom_d := vp^2;
  if denom_x eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_d eq 0).";
  else
    d := (1-4*(up^3)/(vp^2))*a; //<---- is this right?
    c_x := vp*u/(denom_x*sqrt_a); c_y := (u-up)/denom_y; //<---- is this right?

    return [*BF,[BF|a,d],[BF|c_x,c_y]*];
  end if;

end function;
```

Function 18: `Ew2Ead_P4_eval`, mentioned in §3.3.5

```
Ead2Ew_P4_eval := function(BF,lst,P4,p : PP4 := [0,0], a1 := 0, a3 := 0)

  a := BF!lst[1];

  // If 'a' is not a square on BF, then we need to work on an extension field
  tf, sqrt_a := IsSquare(BF!a);
  if not tf then
    // QF_int := Integers()!(Numerator(a)*Denominator(a));
    // BF<sqrt_a> := QuadraticField(QF_int);
    BF := AlgebraicClosure(BF);
    tf, sqrt_a := IsSquare(BF!a);
    // set a flag that we changed base field?;
  end if;

  up := (BF ! P4[1]) + BF!PP4[1]; vp := (BF ! P4[2]) + (a1*up + a3)/2;
  u2p := BF ! PP4[1]; v2p := BF ! PP4[2];
  a := BF!a; d := (BF!lst[2])/a; x := (BF!p[1])*sqrt_a; y := BF!p[2]; //<---- is this right?

  denom_x := (1-y); denom_y := x*(1-y);
  if denom_x eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  else
    a2 := 2*(1+d)/(1-d)*up; a4 := up^2;
    c_u := up*(1+y)/denom_x; c_v:= vp*(1+y)/denom_y;

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;

    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

    return [*BF,[a1,a2,a3,a4,a6],[c_u,c_v]*];
  end if;

end function;
```

Function 19: `Ead2Ew_P4_eval`, mentioned in §3.3.5

```
Ew2Ead_eval := function(BF,lst,p: PP4 := [0,0], a := 1)

  // If 'a' is not a square on BF, then we need to work on the algebraic closure of BF
```

```
    tf, sqrt_a := IsSquare(BF!a);
  if not tf then
    //QF_int := Integers()!(Numerator(a)*Denominator(a));
    //BF<sqrt_a> := QuadraticField(QF_int);
    BF := AlgebraicClosure(BF);
    tf, sqrt_a := IsSquare(BF!a);
    // set a flag that we changed base field?;
  end if;

  a := BF!a;
  if #lst eq 5 then
        a1 := BF!lst[1]; a2 := BF!lst[2]; a3 := BF!lst[3]; a4 := BF!lst[4]; a6 := BF!lst[5];
    elif #lst eq 2 then
        a1 := BF!0; a2 := BF!0; a3 := BF!0; a4 := BF!lst[1]; a6 := BF!lst[2];
    else
        return "Error: wrong number of coeffs";
    end if;
  u := BF!p[1]; v := BF!p[2];

  u2p := BF ! PP4[1];
  u := BF!p[1] + u2p; v := BF!p[2] + (a1*u + a3)/2;
  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - (a1^2)*u2p/2; a2 +:= - 3*u2p + (a1^2)/4;

  if a2 ne 0 then
    d := (a2/2)-1;
  elif (a2 eq 0) and (IsSquare(BF!(a4))) then
    d := Sqrt(a4) + 1;
  else
     return "Error: d is a not a square over the basis field";
  end if;
  d *:= a; //<---- is this right?


  denom_x := v; denom_y := 2*(a2-2)*u^2-v^2;
  if denom_x eq 0 then
    return "Error: map not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: map not defined for this point since (denom_y eq 0).";
  else
    c_x := -2*u/(denom_x*sqrt_a); c_y := (v^2-a2*u^2 - 2*u^3)/denom_y; //<---- is this right?

    return [*BF,[a,d],[c_x,c_y]*];
  end if;

end function;
```

Function 20: `Ew2Ead_eval`, mentioned in §3.3.5

```
Ead2Ew_eval := function(BF,lst,p: PP4 := [0,0], a1 := 0, a3 := 0)

        a := BF!lst[1];
  // If 'a' is not a square on BF, then we need to work on the algebraic closure of BF
        tf, sqrt_a := IsSquare(BF!a);
  if not tf then
    BF := AlgebraicClosure(BF);
    tf, sqrt_a := IsSquare(BF!a);
  end if;

  a := BF!a; d := (BF!lst[2])/a; x := (BF!p[1])*sqrt_a; y := BF!p[2]; u2p := BF!PP4[1];

  denom_x := x^2; denom_y := x^3;
  if denom_x eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: birat. eq. not defined for this point since (denom_y eq 0).";
  else
    a2 := 2*(d+1); a4 := (d-1)^2;
    A := 2*y - (2*d*y + d +1)*x^2 + 2;
    c_u := A/(x^2); c_v:= -2*A/(x^3);

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;

    a2 -:= -3*u2p + (a1^2)/4;
```

```
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;


    return [*BF,[a1,a2,a3,a4,a6],[c_u,c_v]*];
  end if;

end function;
```

Function 21: `Ead2Ew_eval`, mentioned in §3.3.5

B.3 Twisted Edwards - Weierstrass 2-isogenies

```
Ew2Ead_iso_eval := function(BF,lst,p : PP4 := [0,0], u_lst := [0,0,0])

    if #lst eq 5 then
        a1 := BF!lst[1]; a2 := BF!lst[2]; a3 := BF!lst[3]; a4 := BF!lst[4]; a6 := BF!lst[5];
    elif #lst eq 2 then
        a1 := BF!0; a2 := BF!0; a3 := BF!0; a4 := BF!lst[1]; a6 := BF!lst[2];
    else
        return "Error: wrong number of coeffs";
    end if;
  u2p := BF ! PP4[1];
  a4 +:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - (a1^2)*u2p/2; a2 +:= - 3*u2p + (a1^2)/4;

  if u_lst eq [0,0,0] then
    PR<s> := PolynomialRing(BF);
    if a1 ne 0 or a3 ne 0 then
      a6 := 0;
    end if;
    rts := Roots(s^3 + a2*s^2 + a4*s + a6);
    if #rts lt 3 then
      BF := AlgebraicClosure(BF);
      PR<s> := PolynomialRing(BF);
      rts := Roots(s^3 + BF!a2*s^2 + BF!a4*s + BF!a6);
    end if;
    rts_abs := [Abs(rts[i][1]): i in [1..3]];
    ParallelSort(~rts_abs,~rts);
    u0 := rts[1][1]; u1 := rts[2][1] - u0; u2 := rts[3][1] - u0;
  else
    u0 := BF!u_lst[1]; u1 := BF!u_lst[2]; u2 := BF!u_lst[3];
  end if;
  u := BF!p[1] + u2p - u0; v := BF!p[2] + (a1*u + a3)/2;

  denom_x := u1*u2 - u^2; denom_y := v^2 + (u1-u2)*u^2;
  if denom_x eq 0 then
    return "Error: isogeny not defined for this point since (denom_x eq 0).";
  elif denom_y eq 0 then
    return "Error: isogeny not defined for this point since (denom_y eq 0).";
  else
    a := 4*u1; d := 4*u2;
    c_x := v/denom_x; c_y := (v^2 - (u1-u2)*u^2)/denom_y;

    return [* BF,[a,d],[c_x,c_y], u0 *];
  end if;

end function;
```

Function 22: `Ew2Ead_iso_eval`, discussed in §3.3.6

```
Ead2Ew_iso_eval := function(BF,lst,p : PP4 := [0,0], a1 := 0, a3 := 0, u0 := 0)

  u2p := PP4[1];
  if (u2p ne 0 or a1 ne 0 or a3 ne 0) and u0 ne 0 then
    return "Error: either specify u2p,a1,a3 != 0 or u0 != 0";
  end if;

  a := BF!lst[1]; d := BF!lst[2]; x := BF!p[1]; y := BF!p[2];

  denom_u := 4*x^2; denom_v := 32*x*(1+y)*(1-y);
  if denom_u eq 0 then
    return "Error: isogeny not defined for this point since (denom_u eq 0).";
  elif denom_v eq 0 then
```

```
    return "Error: isogeny not defined for this point since (denom_v eq 0).";
  else
    u1 := a/4 + u0; u2 := d/4 + u0;
    a2 := -(u0+u1+u2); a4 := u0*u1 + u0*u2 + u1*u2; a6 := -u0*u1*u2;
    c_u := 1/denom_u +u0; c_v := (a-d)*((1-y)^2-(1+y)^2)/denom_v;

    c_u -:= u2p; c_v -:= (a1*c_u+a3)/2;
    a2 -:= -3*u2p + (a1^2)/4;
    a4 -:= 3*u2p^2 - 2*a2*u2p + (a1*a3)/2 - ((a1^2)*u2p)/2;
    a6 := - u2p^3 - a2*u2p^2 + a4*u2p - (a1*u2p - a3)^2/4;

    return [*[a1,a2,a3,a4,a6],[u0,u1,u2],[c_u,c_v]*];
  end if;

end function;
```

Function 23: `Ead2Ew_iso_eval`, discussed in §3.3.6

Bibliography

[1] C. Arene, T. Lange, M. Naehrig, and C. Ritzenthaler. Faster computation of the Tate pairing. *Journal of number theory*, 131(5):842–857, 2011.

[2] M. Ashraf and Kirlar B. B. On the alternate models of elliptic curves. *International Journal of Information Security Science*, 1(2):49–66, 2012.

[3] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In *International Conference on Cryptology in Africa*, pages 389–405. Springer, June 2008.

[4] D. J. Bernstein, C. Chuengsatiansup, D. Kohel, and T. Lange. Twisted Hessian curves. *Cryptology ePrint Archive*, 2015.

[5] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 29–50. Springer, December 2007.

[6] D. J. Bernstein and T. Lange. A complete set of addition laws for incomplete Edwards curves. *Journal of Number Theory*, 131(5):858–872, 2011.

[7] O. Billet and M. Joye. The Jacobi model of an elliptic curve and side-channel analysis. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 34–42. Springer, May 2003.

[8] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[9] W. Bosma and H. W. Lenstra. Complete systems of two addition laws for elliptic curves. *Journal of Number theory*, 53(2):229–240, 1995.

[10] J. W. S. Cassels and E. V. Flynn. *Prolegomena to a middlebrow arithmetic of curves of genus 2*, volume 230. Cambridge University Press, 1996.

[11] A. Chatterjee and I. Sengupta. High-speed unified elliptic curve cryptosystem on FPGAs using binary Huff curves. In *Progress in VLSI Design and Test: 16th International Symposium on VSLI Design and Test, VDAT 2012, Shipur, India, July 1-4, 2012, Proceedings*, volume 7373, page 243. Springer, 2012.

[12] D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7(4):385–434, 1986.

[13] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.

[14] C. Costello and B. Smith. Montgomery curves and their arithmetic. *Journal of Cryptographic Engineering*, 8(3):227–240, 2018.

[15] R. Crandall and C. Pomerance. *Prime numbers: a computational perspective*. Springer, 2001.

[16] M. R. Dam. Edwards elliptic curves. Bachelor's thesis, Faculty of Science and Engineering, University of Groningen, 2012. Retrieved from: https://fse.studenttheses.ub.rug.nl/10478/1/Marion_Dam_2012_WB_1.pdf.

[17] C. Doche, T. Icart, and D. R. Kohel. Efficient scalar multiplication by isogeny decompositions. In *International Workshop on Public Key Cryptography*, pages 191–206. Springer, 2006.

[18] M. Dubal and A. Deshmukh. On pseudo-random number generation using elliptic curve cryptography. In *International Symposium on Security in Computing and Communication*, pages 77–89. Springer, 2013.

[19] H. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

[20] R. R. Farashahi and M. Joye. Efficient arithmetic on Hessian curves. *Public Key Cryptography–PKC 2010*, page 243, 2010.

[21] R. Feng, M. Nie, and H. Wu. Twisted Jacobi intersections curves. In *International Conference on Theory and Applications of Models of Computation*, pages 199–210. Springer, 2010.

[22] T. Hales. The group law for Edwards curves. arXiv preprint arXiv:1610.05278, 2011.

[23] H. Hisil, K. K. Wong, G. Carter, and E. Dawson. Jacobi quartic curves revisited. In *Australasian Conference on Information Security and Privacy*, pages 452–468. Springer, 2009.

[24] H. Hisil, K.K.H. Wong, G. Carter, and Dawson E. Twisted edwards curves revisited. Cryptology ePrint Archive, Report 2008/522, 2008. Retrieved from: https://eprint.iacr.org/2008/522.

[25] G. B. Huff. Diophantine problems in geometry and elliptic ternary forms. *Duke Mathematical Journal*, 15(2):443–453, 1948.

[26] M. Joye and J.-J. Quisquater. Hessian elliptic curves and side-channel attacks. *Lecture Notes in Computer Science*, 2162:0402–0402, 2001.

[27] M. Joye, M. Tibouchi, and D. Vergnaud. Huff's model for elliptic curves. *International Algorithmic Number Theory Symposium*, pages 234–250, July 2010.

[28] J. Kim, J. H. Park, D.-C. Kim, and W.-H. Kim. Complete addition law for Montgomery curves. In *International Conference on Information Security and Cryptology*, pages 260–277. Springer, 2019.

[29] D. Kohel. Addition law structure of elliptic curves. *Journal of Number Theory*, 131(5):894–919, 2011.

[30] P.-Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 391–401. Springer, 2001.

[31] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[32] D. Nguyen. Correspondence between elliptic curves in Edwards-Bernstein and Weierstrass forms. Master's thesis, Department of Mathematics and Statistics, University of Ottawa, 2017. Retrieved from: https://mysite.science.uottawa.ca/mnevins/papers/NguyenMScProj2017.pdf.

[33] N. G. Orhon and H. Hisil. Speeding up Huff form of elliptic curves. *Designs, Codes and Cryptography*, 86(12):2807–2823, 2018.

[34] M. Rashid, M. Imran, M. Kashif, and A. Sajid. An optimized architecture for binary Huff curves with improved security. *IEEE Access*, 9:88498–88511, 2021.

[35] O. Reyad, M. Karar, and K. Hamed. Random bit generator mechanism based on elliptic curves and secure hash function. In *2019 International Conference on Advances in the Emerging Computing Technologies (AECT)*, pages 1–6. IEEE, 2020.

[36] A. Rice and E. Brown. Why ellipses are not elliptic curves. *Mathematics Magazine*, 85(3):163–176, 2012.

[37] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[38] N. P. Smart. The Hessian form of an elliptic curve. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 118–125. Springer, May 2001.

[39] A. Smeets. ttransformation from Weierstrass curves to Jacobi curves. Bachelor's thesis, Eindhoven University of Technology, 2014. Retrieved from: https://pure.tue.nl/ws/portalfiles/portal/67740154/775251-1.pdf.

[40] H. Wu and R. Feng. Elliptic curves in Huff's model. *Wuhan University Journal of Natural Sciences*, 17(6):473–480, 2012.