

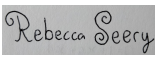
ALGORITHMS FOR RATIONAL CONICS

REBECCA SEERY
SUPERVISOR: NICOLAS MASCOT

DECLARATION CONCERNING PLAGIARISM

I have read and understood the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at <http://www.tcd.ie/calendar>.

I have also read and understood the guide, and completed the ‘Ready Steady Write’ Tutorial on avoiding plagiarism, located at <https://libguides.tcd.ie/academic-integrity/ready-steady-write>.

Signed: 

1. INTRODUCTION

In this project, we take an algorithmic approach to the study of rational conics.

Definition 1.1. A **rational conic** is an equation of the form $ax^2 + bxy + cy^2 + dx + ey + f = 0$, where $a, b, c, d, e, f \in \mathbb{Q}$.

Definition 1.2. A conic $ax^2 + bxy + cy^2 + dx + ey + f = 0$ is **degenerate** if

$$\det \begin{pmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{pmatrix} = 0$$

Over \mathbb{C} , the solution sets of non-degenerate conics take the form of ellipses, parabolas, hyperbolas and circles, while those of degenerate conics can be points, lines, Xs or double lines (two lines on top of one another).

However, over \mathbb{Q} , it is an entirely different story. For example, $x^2 + y^2 + 1 = 0$ is a non-degenerate conic, but it has no solutions over \mathbb{Q} , or even over \mathbb{R} .

Our first problem is to find rational points on rational conics. We will build a program which outputs either a comprehensive list of solutions, or a proof that

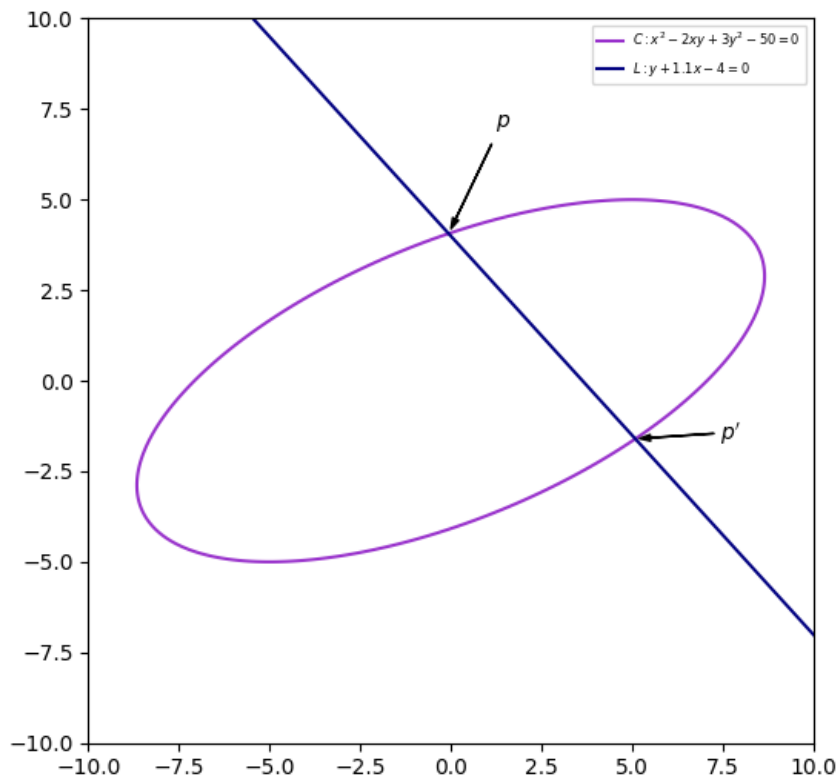
no solutions exist. We will do this using Python and Sage [1].

Degenerate conics are reducible algebraic varieties: they are the union of two lines. Lines are very easy to parameterize so this case is not very interesting. For this reason, we shall simply output a message that the conic is degenerate for any of these special cases.

Why should we place emphasis on conics? Linear Diophantine equations are straightforward to solve using the Smith Normal Form. Conics present a much more interesting challenge, while still having nice properties which higher degree curves lack. An important example of one of these properties is the following theorem:

Theorem 1.3. (*Line-Intersection Property*)

Let C be a conic over \mathbb{Q} . Let L be a line with rational coefficients which intersects C at a rational point p . Then if L intersects C at another point p' , p' will also be a rational point.



To see that this is true, recall that we find the intersection points of a line with a conic using simultaneous equations. We begin with the linear equation, expressing one variable as a linear function of the other. We then substitute it in to the conic

to get a quadratic equation in just one variable. This has rational coefficients because L and C did. Since one of the roots is rational, the other will be too (by Vieta's Formulas). Finally, when we get the other variable back as a linear function of the one we have, it must also be rational.

Conversely, it is clear that we can get every rational point this way as if there is another rational point we can join it to the original with a rational line.

This remarkable property is extremely useful. Once we have one rational point on the conic, we can find every other point by intersecting the conic with rational lines!

Unfortunately, higher degree curves do not have this property, so finding all solutions on them is very difficult. This is one reason we are especially interested in conics. We will later see another property: the Local-Global Principle 7.1.

Example 1.4. (Pythagorean Triples) Searching for (nonzero) Pythagorean Triples $x^2 + y^2 = z^2$ with $x, y, z \in \mathbb{Z}$ is equivalent to solving $(\frac{x}{z})^2 + (\frac{y}{z})^2 = 1$, i.e. finding rational points x and y which satisfy $x^2 + y^2 = 1$. We can easily spot that $(x = 1, y = 0)$ is a rational point. Using the Line-Intersection Property 1.3 on this point we obtain something analogous to Euclid's formulas for Pythagorean triples¹:

$$x = \frac{m^2 - n^2}{m^2 + n^2} \quad y = \frac{2mn}{m^2 + n^2}$$

Example 1.5. (Circle of Radius $\sqrt{3}$) We want to find rational points on the conic $x^2 + y^2 = 3$. There are no solutions.

A solution to this corresponds to an integer solution to the homogenized conic $x^2 + y^2 = 3z^2$ (obtained by multiplying across by the lcm of the denominators of x and y) where $z \neq 0$.

If we had a solution (x, y, z) , we could divide across by the gcd to get another solution. Hence assume that x, y and z are coprime.

Any solution will also be a solution modulo 4. $x^2 + y^2 \equiv 0, 1$ or $2 \pmod{4}$. $3z^2 \equiv 0$ or 3 . However, the only way to have both sides be 0 is if x, y and z are all even. But this is impossible since we assumed that they were coprime. Therefore there are no solutions.

We want to create a program which solves rational conics. We expect our program to output either a parametrization of all solutions (like in 1.4) or a proof that no

¹Finding rational points on the circle of radius 1 actually corresponds to finding **primitive** Pythagorean triples (because we can multiply the top and bottom of the fraction by a nonzero scalar to get the same rational point). Euclid's formulas give us all of the rational points, but in order to obtain all Pythagorean triples, we must use the formulas $x = k(m^2 - n^2)$, $y = 2mnk$, $z = k(m^2 + n^2)$.

solutions exist (as in 1.5). Thanks to the Line-Intersection Property 1.3, we only need to find one point on a conic with solutions to get a parametrization of all of them. Our focus now shifts to finding that initial rational point, or proving none exist. To do this, we will need to set up quite a bit of theory, which we will do over the next few sections.

2. ELEMENTARY NUMBER THEORY

We will be building algorithms for conics from the ground up. To do this, we need some basic ideas from number theory. Even if the reader has taken a course in number theory already, they may still want to stick around as the flavour of this section is more algorithmic than they are likely used to.

We begin with a very simple algorithm which most people will already be familiar with: the Extended Euclidean Algorithm. This algorithm computes the greatest common divisor (gcd) of two numbers a and b , and also computes u and v for a Bézout relation $au + bv = \gcd(a, b)$.

This works by applying the familiar Euclidean Algorithm to compute the gcd, which uses the fact that $\gcd(a, b) = \gcd(b, a \bmod b)$ for any $a \geq b$. The algorithm stops when $b = 0$, at which point the gcd will be a . To obtain u and v , we simply keep track of the change of variables via recursion, and work backwards from the last step: $a = d$, $b = 0$, $u = 1$, $v = 0$, $1 \cdot d + 0 \cdot 0 = d$, where $d = \gcd(a, b)$.

```

Function extended_euclidean( $a, b$ ):
  Data: Integers  $a$  and  $b$ 
  Result: GCD  $d$  and coefficients  $u, v$  such that  $au + bv = d$ 
  if  $b > a$  :
    | ( $d, u, v$ )  $\leftarrow$  extended_euclidean( $b, a$ )
    | return ( $d, v, u$ )
  else if  $b = 0$  :
    | return ( $a, 1, 0$ )
  else:
    | ( $d, u, v$ )  $\leftarrow$  extended_euclidean( $b, a \bmod b$ )
    | return ( $d, v, u - \lfloor a/b \rfloor \cdot v$ )

```

Algorithm 1: Extended Euclidean Algorithm

See page 32 for the code.

Example 2.1. One important use of the Extended Euclidean Algorithm is computing inverses mod n . To compute the inverse of a mod n , we can get a Bézout relation $au + nv = \gcd(a, n)$. The inverse only exists if a and n are coprime, so then we have $au + nv = 1$, or $au = 1 - nv$, so $au \equiv 1 \pmod{n}$, i.e. u is the inverse of a mod n .

Changing variables via recursion like in the Extended Euclidean Algorithm above will be a very useful tool for us when we write an algorithm to solve conics, which can be found here: [7](#).

Another fundamental tool for us is the Chinese Remainder Theorem. It is particularly useful for reducing cases mod N to cases mod n_i , where n_i are simpler. We will need it later for calculating square roots mod n .

The Chinese Remainder Theorem was originally studied in the 3rd century AD by Chinese mathematician Sunzi. Since then, it has been reformulated and abstracted into many different versions, including the one we state here.

Although one can make the theorem much more general, this version is adequate for our purposes.

Theorem 2.2. (*Chinese Remainder Theorem*)

Let $N \in \mathbb{Z}^+$. Let $N = n_1 n_2 \dots n_k$, where n_i are pairwise coprime integers and $n_i \geq 1 \forall i$. Then:

$$x \bmod N \mapsto (x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k)$$

$$\mathbb{Z}/N\mathbb{Z} \rightarrow \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \dots \times \mathbb{Z}/n_k\mathbb{Z}$$

defines a ring isomorphism.

We can turn this theorem into an algorithm quite easily. The relevant part is going backwards from a set of congruences $x_i \bmod n_i$ to obtain the original $x \bmod N$. We would like to simply multiply each x_i by $\frac{N}{n_i}$, but this could ruin the congruence $x \equiv x_i \bmod n_i$, since $\frac{N}{n_i} \not\equiv 1 \bmod n_i$ in general. However, if we can find an inverse v_i of $\frac{N}{n_i} \bmod n_i$, we will be able to obtain x by $x = \sum_{i=1}^{\# \text{congruences}} x_i \cdot \frac{N}{n_i} \cdot v_i$. We can compute the inverse using the Extended Euclidean Algorithm above ([2.1](#)). We do just that in the following algorithm.

```

Function chinese_remainder_theorem(congruences) :
  Data: List of tuples congruences containing pairs  $(x_i, n_i)$ 
  Result: Integer  $x$  such that  $x \equiv x_i \pmod{n_i}$  for all congruences
            $(x_i, n_i)$ 
   $N \leftarrow \prod n_i$ 
   $x \leftarrow 0$ 
  for  $(x_i, n_i)$  in congruences do
     $N_i \leftarrow \frac{N}{n_i}$ 
     $(d, u, v) \leftarrow \text{extended\_euclidean}(n_i, N_i)$     //  $v$  is an inverse
    of  $N_i \pmod{n_i}$ .

    if  $d \neq 1$  :
      | ValueError("The moduli must be pairwise coprime.")

     $x \leftarrow x + x_i \cdot N_i \cdot v$ 

  return  $x \pmod{N}$ 

```

Algorithm 2: Chinese Remainder Theorem

See page [32](#) for the code.

We also define the notion of a norm from algebraic number theory. We require this to prove the Hasse-Minkowski Theorem [7.1](#) later. My source for this was [\[7\]](#). The section on Norms is the section ‘Continued Fractions Attached to Quadratic Irrationals’ and they are defined on slide 45.

Definition 2.3. Let $d \in \mathbb{Z}$ not square. Let $\alpha = a + b\sqrt{d}$ be any element in $\mathbb{Q}(\sqrt{d})$.

The **conjugate** of α is $\bar{\alpha} = a - b\sqrt{d}$.

The **(ANT) norm** of α (in the sense of Algebraic Number Theory) is $N(\alpha) = \alpha\bar{\alpha} = a^2 - db^2$.

Note that α and $\bar{\alpha}$ are Galois conjugates in the field extension $\mathbb{Q}(\sqrt{d})/\mathbb{Q}$.

Lemma 2.4. $N(\alpha)$ is multiplicative, i.e. $N(\alpha\beta) = N(\alpha)N(\beta)$. This means that the space of norms is closed under multiplication (and nonzero division).

3. THE P-ADIC NUMBERS

We now study the p -adic numbers. These are a very nice way of packaging congruences mod p^v for p prime. We need them to make sense of the Local Global Principle when we write an algorithm to solve conics.

Definition 3.1. Let $n \in \mathbb{Z} \setminus \{0\}$, and let p be a prime. The **p -adic valuation** of n is the highest power of p which divides n . It is denoted $v_p(n)$.

We also define $v_p(0) = +\infty$.

For $x = \frac{a}{b} \in \mathbb{Q} \setminus \{0\}$, we define $v_p(x) = v_p(a) - v_p(b)$.

Note that the latter is well-defined as if the numerator and denominator are both multiplied by nonzero k , we get $v_p(x) = v_p(ka) - v_p(kb) = v_p(k) + v_p(a) - v_p(k) - v_p(b) = v_p(a) - v_p(b)$.

Lemma 3.2. *We have: $v_p(xy) = v_p(x) + v_p(y)$.*

Definition 3.3. Let F be a field. A map $||\cdot|| : F \rightarrow \mathbb{R}$ is a **norm** if the following conditions are satisfied:

- (1) $||x|| \geq 0$ and $||x|| = 0$ iff $x = 0$ (Positive Definiteness)
- (2) $||xy|| = ||x|| ||y||$ (Multiplicativity)
- (3) $||x + y|| \leq ||x|| + ||y||$ (Triangle Inequality)

This notion of norm is effectively a special case of a vector space norm, where the vector space is the field is viewed as a 1-dimensional vector space over itself. Then the Homogeneity condition becomes a Multiplicativity condition. However, be wary that this notion of norm is completely different to the norms in Algebraic Number Theory from 2.3. We will do our best to avoid confusion by using the notation $N(x)$ for an ANT norm and $|x|$ or $||x||$ for a vector space norm, or by simply stating which norm we are referring to explicitly.

Definition 3.4. Let p be any prime. The **p -adic norm** is a norm on \mathbb{Q} defined by $|x|_p = p^{-v_p(x)}$.

We often denote the usual norm on \mathbb{Q} (which is just the absolute value on \mathbb{R} restricted to \mathbb{Q}) by $|x|_\infty$.

We can view \mathbb{Q} as a metric space where the metric is induced by $|x|_p$. We may then complete this metric space to obtain the field \mathbb{Q}_p of p -adic numbers.

We also adopt the convention that $\mathbb{Q}_\infty = \mathbb{R}$.

Recall that two norms p and q are equivalent if $\exists c, C > 0 : \forall x \quad cq(x) \leq p(x) \leq Cq(x)$.

Theorem 3.5. (*Ostrowski's Theorem*)

Every norm on \mathbb{Q} is equivalent to exactly one of:

- (1) The trivial norm (the norm $|x| = 0$ if $x = 0$, otherwise $|x| = 1$).
- (2) The usual norm $|\cdot|_\infty$.
- (3) One of the norms $|\cdot|_p$ (which are not equivalent to one another).

This theorem is relevant to the Local-Global Principle 7.1 because it tells us that \mathbb{Q}_p and \mathbb{R} are the only sensible completions of the field \mathbb{Q} .

Limits of sequences in \mathbb{Q}_p are much nicer than in \mathbb{R} : A sequence (a_n) converges in \mathbb{Q}_p if and only if $|a_{n+1} - a_n| \rightarrow 0$ as $n \rightarrow \infty$. We don't get sequences like the Harmonic Series, which diverges even though the differences between terms go to 0.

Definition 3.6. The *p-adic integers* are the subring of \mathbb{Q}_p defined by $\mathbb{Z}_p = \{x \in \mathbb{Q}_p \mid |x|_p \leq 1\}$.

Note that $|x|_p \leq 1$ iff $v_p(x) \geq 0$. Also, invertible elements of \mathbb{Z}_p are exactly those with $v_p(x) = 0$.

Theorem 3.7. (*Representation of p-adic Numbers*)

Let $x \in \mathbb{Q}_p$. Then $x = p^v u$ where $u \in \mathbb{Z}_p^\times$.

Furthermore, we may write u as an infinite sequence $a_0 + a_1p + a_2p^2 + a_3p^3 + \dots$ where $a_i \in \{0, \dots, p-1\}$ are uniquely determined by u . The sequence of partial sums is Cauchy in \mathbb{Q} w.r.t. $|\cdot|_p$, and converges to u in \mathbb{Q}_p . Conversely, given any sequence of $a_i \in \{0, \dots, p-1\}$ with $a_0 \neq 0$, the sequence of partial sums given by $S_n = a_0 + a_1p + a_2p^2 + a_3p^3 + \dots + a_np^n$ converges to a unique $u \in \mathbb{Z}_p^\times$.

Remark 3.8. We don't have to use $\{0, \dots, p-1\}$: we can choose any set of representatives of $\mathbb{Z}/p\mathbb{Z}$.

Theorem 3.9. (*Hensel's Lemma*)

Let $f \in \mathbb{Z}_p[x]$ be a polynomial. Let $a_0 \in \mathbb{Z}_p$ be such that $f(a_0) \equiv 0 \pmod{p}$, and $f'(a_0) \not\equiv 0 \pmod{p}$.

Define a sequence a_n by $a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)}$. Then a_n converges to a unique $a \in \mathbb{Z}_p$ such that $f(a) = 0$ and $a \equiv a_0 \pmod{p}$.

This sequence converges very quickly. If $f(a_n) \equiv 0 \pmod{p^v}$, then $f(a_{n+1}) \equiv 0 \pmod{p^{2v}}$.

Proof.

- Let's begin with the lift from a_0 to a_1 .

- Consider the Taylor Series expansion of $f(x)$ at a_0 :

$$f(a_0 + pt) = f(a_0) + f'(a_0)pt + \frac{1}{2}f''(a_0)p^2t^2 + \dots$$

- We want to find a t such that $f(a_0 + pt) \equiv 0 \pmod{p^2}$.
- Note $f(a_0 + pt) \equiv f(a_0) + f'(a_0)pt \pmod{p^2}$.²
- Since $f(a_0) \equiv 0 \pmod{p}$, we can write $f(a_0) = \alpha p$, for some $\alpha \in \mathbb{Z}_p$.
- Making $f(a_0) + f'(a_0)pt \equiv 0 \pmod{p^2}$ is equivalent to making $\alpha + f'(a_0)t \equiv 0 \pmod{p}$.
- We need $t = \frac{-\alpha}{f'(a_0)}$. Note that $f'(a_0)$ is invertible mod p since it is nonzero.
- This means $a_1 = a_0 - \frac{\alpha p}{f'(a_0)} = a_0 - \frac{f(a_0)}{f'(a_0)}$.
- Now for the lift from a_n to a_{n+1} . Suppose that $f(a_n) \equiv 0 \pmod{p^v}$, and $f'(a_n) \equiv 0 \pmod{p}$ (Note that $f'(a_n) \equiv f'(a_0) \pmod{p}$, so this property is preserved when lifting).
- Consider the Taylor Series expansion of $f(x)$ at a_n :

$$f(a_n + p^v t) = f(a_n) + f'(a_n)p^v t + \frac{1}{2}f''(a_n)p^{2v}t^2 + \dots$$

- We want to find a t such that $f(a_n + p^v t) \equiv 0 \pmod{p^{2v}}$.
- Write $f(a_n) = \alpha p^v$ for $\alpha \in \mathbb{Z}_p$. We can do this since $f(a_n) \equiv 0 \pmod{p^v}$.
- We need $f(a_n + p^v t) \equiv f(a_n) + f'(a_n)p^v t \equiv 0 \pmod{p^{2v}}$. This is equivalent to $\alpha + f'(a_n)t \equiv 0 \pmod{p^v}$.
- Invertibles mod p^v are those which are coprime with p , so $f'(a_n)$ is invertible mod p^v since it is nonzero mod p .
- Hence we can set $t = -\frac{\alpha}{f'(a_n)}$, which makes $a_{n+1} = a_n + p^v t = a_n - \frac{\alpha p^v}{f'(a_n)} = a_n - \frac{f(a_n)}{f'(a_n)}$. Then we indeed have $f(a_{n+1}) \equiv 0 \pmod{p^{2v}}$.
- Since we have shown that the differences between terms a_n and a_{n+1} is divisible by higher and higher values of p , we have $|a_{n+1} - a_n|_p \rightarrow 0$ as $n \rightarrow \infty$. Therefore the sequence converges (since sequences in \mathbb{Q}_p converge iff this difference tends to 0). Uniqueness of a is guaranteed since \mathbb{Q}_p is a metric space.

□

²For $p = 2$, we must show $\frac{1}{2}f''(a_0)p^2t^2$ is divisible by 4. This is true because differentiating the polynomial f twice will multiply each coefficient c_n of x^n by $n(n-1)$ for $n \geq 2$. But since $n(n-1)$ is always even, $f''(a_0)$ will be even too.

Hensel's Lemma is very similar to the Newton-Rapherson Iteration. It is actually much nicer, since Hensel's Lemma is guaranteed to converge to a root, whereas Newton-Rapherson is not.

There is a stronger version of Hensel's Lemma, which the above version is required to prove.

Theorem 3.10. (*Strong Hensel*)

Let $f \in \mathbb{Z}_p[x]$ be a polynomial. Let $a_0 \in \mathbb{Z}_p$ be such that $v_p(f(a_0)) > 2v_p(f'(a_0))$. Define a sequence a_n by $a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)}$. Then a_n converges to a root of f in \mathbb{Z}_p .

It will be more convenient for us later when we study Hilbert Symbols to use the multiplicative group $\{1, -1\}$ in place of $\mathbb{Z}/2\mathbb{Z}$. Note that there is an obvious isomorphism between the two, namely $1 \mapsto 0$ and $-1 \mapsto 1$. We use the multiplicative group $\{1, -1\}$ in the following theorem.

Theorem 3.11. (*Squares in \mathbb{Q}_p*)

Let $x \in \mathbb{Q}_p$. Then $x = p^v u$ where $v = v_p(x)$, $u \in \mathbb{Z}_p^\times$.

- If $p \neq 2$, let $\pi : \mathbb{Z}_p \rightarrow \mathbb{Z}/p\mathbb{Z}$ be the projection $u \mapsto u \bmod p$. Then x is square iff v is even and $\pi(u)$ is square in $\mathbb{Z}/p\mathbb{Z}$.
Also, $\mathbb{Q}_p^\times / (\mathbb{Q}_p^\times)^2 \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ via $x = p^v u \mapsto ((-1)^v, \left(\frac{\pi(u)}{p}\right))$.
- If $p = 2$, let $\pi : \mathbb{Z}_2 \rightarrow \mathbb{Z}/8\mathbb{Z}$ be the projection $u \mapsto u \bmod 8$. Note that u is odd since its even component is in p^v , so π is actually a map to $(\mathbb{Z}/8\mathbb{Z})^\times$. Then x is square iff v is even and $\pi(u)$ is square in $\mathbb{Z}/8\mathbb{Z}$, i.e. $\pi(u) \equiv 1 \bmod 8$.
Also, $\mathbb{Q}_2^\times / (\mathbb{Q}_2^\times)^2 \cong \mathbb{Z}/2\mathbb{Z} \times (\mathbb{Z}/8\mathbb{Z})^\times \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ via $x = p^v u \mapsto ((-1)^v, (-1)^c, (-1)^d)$ where $c, d \in \{0, 1\}$ are such that $\pi(u) \equiv 3^c 5^d \bmod 8$ (since $(\mathbb{Z}/8\mathbb{Z})^\times$ is generated by 3 and 5).

This theorem will be very useful when we study Hilbert symbols later on.

For more information on p -adic Numbers, an excellent reference is [2]. Most of what we've built up here (with the exception of Strong Hensel's Lemma 3.10 and the characterisation of squares in \mathbb{Q}_p 3.11) is covered in Chapter I of this book.

4. SQUARE ROOTS MOD N

We now have all the tools we need in order to derive the Rube-Goldberg machine of an algorithm which computes square roots modulo n . This algorithm is a critical component for solving conics using the Hasse-Minkowski Theorem 7.1.

```

Function tonelli_shanks( $a, p$ ) :
  Data: Integers  $a$  and  $p$  where  $p$  is prime
  Result: Square root of  $a$  modulo  $p$  if it exists, otherwise returns No Solutions

  if  $p = 2$  :
    | return  $a$            //  $a$  is its own square root mod 2
  if  $a = 0$  :
    | return 0
  if not  $is\_square(a, p)$  :
    | return No Solutions

  factor  $p - 1 = q \cdot 2^e$ 
  while  $is\_square(z, p)$  do
    |  $z \leftarrow$  random integer in the range  $[2, p - 1]$ 
   $b \leftarrow z^q$ 
   $r \leftarrow a^{(q+1)/2}$ 
   $t \leftarrow a^q$ 

  if  $t = 1$  :
    | return  $r$ 

   $m \leftarrow e - 1$ 
  while  $m > 0$  do
    | if  $t^{2^{(m-1)}} = -1$  :
      | |  $r \leftarrow r \cdot b$ 
      | |  $t \leftarrow t \cdot b^2$ 
      |  $b \leftarrow b^2$ 
      |  $m \leftarrow m - 1$ 

  return  $r$ 

```

Algorithm 3: Tonelli-Shanks Algorithm

Proof. First we deal with the edge cases $p = 2$ and $a = 0$. For $p = 2$, we note that $0^2 = 0$ and $1^2 = 1$, so a is its own square root mod 2. When $a = 0$, it has only one square root, namely 0 itself. It is easier to treat these cases separately than trying to incorporate them into the algorithm. Finally, we add a check to see if a is actually a square mod p .

With these edge cases out of the way, we may assume that p is odd and $a \in (\mathbb{Z}/p\mathbb{Z})^*$ is nonzero and square mod p . Factor $p - 1 = 2^e \cdot q$, where q is odd.

Find a quadratic non-residue z . Do this randomly, choosing numbers between 2 and $p - 1$ and checking if they are not square mod p . This step is usually very fast because half of the numbers between 1 and $p - 1$ are not square mod p .

We can check if a number is square using Euler's Criterion ($a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$) or Legendre symbols, but actually computing roots requires this algorithm.

Set $r = a^{\frac{q+1}{2}}$. Then $r^2 = a^{q+1} = a^q \cdot a$.

Set $t = a^q$, so $r^2 = ta$.

Set $m = e - 1$.

Then $t^{2^m} = t^{2^{e-1}} \equiv (a^q)^{2^{e-1}} = a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) = 1$.

So t is a 2^m -th root of 1.

We use induction to reduce m by finding new values of r and t .

On each step of the induction, we will maintain two conditions:

- (1) t is a 2^m -th root of 1.
- (2) $r^2 = ta$.

These conditions hold at the beginning, and we will ensure that they hold on each step of the induction.

Base Case $m = 0$: Then t is a 2^0 -th root of 1, so $t = 1$.

But $r^2 = ta$, so $r^2 = a$, so $r = \sqrt{a} \pmod{p}$, and we have found our root.

Inductive step: If t is an 2^{m-1} -th root of 1 already, we have done this step of induction, and can proceed to the next step without changing t and r .

Otherwise, $t^{2^{m-1}} = -1$, since -1 is the only other root of 1 besides 1.

We would like to multiply r by a factor b (which we will determine later) in order to satisfy the next pair of conditions.

To maintain the second condition, we must multiply t by b^2 .

To achieve the first condition with $m - 1$, we must have that b^2 is a 2^{m-1} -th root of -1 , i.e. that b is a 2^m -th root of -1 .

By Euler's Criterion, $z^{\frac{p-1}{2}} = z^{q2^{e-1}} \equiv -1 \pmod{p}$, so z^q is a 2^{e-1} -th root of -1 .

Hence, we may repeatedly square z^q to obtain a sequence of 2^i -th roots of -1 in reverse. The one we need is $b = (z^q)^{e-m}$, which is exactly what is computed by the algorithm.

It is straightforward to see that the new t and r satisfy both conditions for $m-1$, so we are done by the inductive hypothesis. □

See [32](#) for the code. For more information on this algorithm, see [\[3\]](#). The section on the Tonelli-Shanks algorithm is 1.5.1 on page 32.

Something is wrong, which you may have already noticed. At the beginning of the section, I promised we would compute square roots modulo n , for any number $n \in \mathbb{Z}^+$. However, the Tonelli-Shanks Algorithm above only works for square roots modulo prime numbers p .

I intend to keep my promise, however, and now I will show how solutions mod p may be extended to solutions mod n using the theory we have built up in the previous sections.

Firstly, we would like to use Hensel's Lemma [3.9](#) to lift from squares mod p to squares mod p^v . In order to do this, we need to find a root of the polynomial $f(x) = x^2 - b$ in \mathbb{Q}_p . Our a_0 is just the root mod p that we obtain from Tonelli-Shanks.

We must check that $f'(a_0) \not\equiv 0 \pmod{p}$. Indeed, $f'(a_0) = 2a_0 \not\equiv 0$ unless $a_0 \equiv 0$ (in which case we are computing the square root of 0, which is just 0) or $2 \equiv 0$, so we may use Hensel's Lemma for any $p \neq 2$.

We can stop computing as soon as we have all the terms before p^v , i.e. after $\lceil \log_2(v) \rceil$ iterations. The resulting number will satisfy $f(a) \equiv 0 \pmod{p^v}$, i.e. a will be a square root of b mod p^v .

For $p = 2$, all is not lost, as we can use Strong Hensel's Lemma 3.10 instead. Note that $v_2(2x) = 1$ when $2 \nmid x$. In order to make the inequality $v_p(f(a_0)) > 2v_p(f'(a_0))$ work, we need to look for square roots modulo 8. Square roots mod 2 and 4 are easy to code by hand. Only numbers which are 0 (impossible) or $\equiv 1 \pmod{8}$ have square roots: We can choose either 1, 3, 5, or 7 for the latter.³ From there, Strong Hensel does the trick, and we may compute square roots modulo 2^v .

We must work out how many times to perform Hensel Lifting when $p = 2$. Suppose $x_n^2 \equiv a \pmod{2^u}$ for some u , i.e. $x_n = a + 2^u t$ for some t . Then $x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}$ so:

$$\begin{aligned}
x_{n+1}^2 - a &= \left(x_n - \frac{x_n^2 - a}{2x_n} \right)^2 - a \\
&= \left(\frac{x_n + \frac{a}{x_n}}{2} \right)^2 - a \\
&= \frac{1}{4} \left(x_n^2 + 2a + \frac{a^2}{x_n^2} \right) - a \\
&= \frac{1}{4} \left((a + 2^u t)^2 + 2a + \frac{a^2}{(a + 2^u t)} \right) - a \quad \dots (\text{since } x_n^2 = a + 2^u t) \\
&= \frac{1}{4} \left(3a + 2^u t + a \frac{1}{1 + 2^u t/a} \right) - a \\
&= \frac{1}{4} \left(3a + 2^u t + a \left(1 - \frac{2^u t}{a} + \frac{4^u t^2}{a^2} + \dots \right) \right) - a \quad \dots (\text{by Geometric Series}) \\
&= \frac{1}{4} \left(3a + 2^u t + a - 2^u t + a \left(\frac{4^u t^2}{a^2} + \dots \right) \right) - a \\
&= 2^{2u-2} \frac{t^2}{a} + \dots
\end{aligned}$$

So $x_{n+1} \equiv a \pmod{2^{2u-2}}$.

³Note that \mathbb{Q}_2 is a field, so although we have 4 choices for initial points, we only get 2 square roots in \mathbb{Q}_2 (which will be negatives of one another).

From here, going to the general case is not hard, and can be done using the Chinese Remainder Theorem 2.2. We decompose n into a product of prime powers: $n = \prod_{i=1}^r p_i^{v_i}$. We may then compute square roots mod $p_i^{v_i}$ using Tonelli-Shanks and Hensel Lifting, as described above. Finally, we can recombine these into a square root modulo n . If any of these square roots do not exist, there is no square root mod n either.

We can now set up a complete algorithm for square roots mod n . You will notice that most of the algorithm is just dealing with different special cases. We have already done the main steps in the Tonelli-Shanks Algorithm 3 and the Chinese Remainder Theorem Algorithm 2. Surprisingly, Hensel Lifting only takes up a couple of lines.

```

Function sqrt_2_adic( $a, v$ ):
  Data: Integer  $a$  and precision  $v$ 
  Result: 2-adic square root of  $a$  with precision  $v$ 

  if  $a \equiv 0 \pmod{p^v}$  :
    | return 0
  // We must compute roots mod 2 and 4 by hand:
  if  $v = 1$  :
    | return 1
  else if  $v = 2$  :
    | if  $a \equiv 1 \pmod{4}$  :
    |   | return 1
    | else:
    |   | return No Solutions

  if  $a \not\equiv 1 \pmod{8}$  :
    | // There are no solutions mod 8, so there are no
    |   solutions mod higher powers of 2 either.
    | return No Solutions

   $x \leftarrow 1$  // 1 is our root mod 8 to lift mod  $2^v$ 
   $u \leftarrow 3$  // Variable to keep track of precision

  while  $u < v$  do
    |  $x \leftarrow x - (x^2 - a)/2x$  // Hensel Lifting
    |  $u \leftarrow 2u - 2$ 

  return  $x$ 

```

Algorithm 4: Square Roots mod 2^v

See page 33 for the code.

```

Function sqrt_p_adic( $a, p, v$ ):
  Data: Integer  $a$ , prime  $p$ , and precision  $v$ 
  Result:  $p$ -adic square root of  $a$  with precision  $v$ 

  if  $a \equiv 0 \pmod{p^v}$  :
    | return 0
  if  $p = 2$  :
    | return sqrt_2_adic( $a, v$ )

  if not is_square( $a, p$ ) :
    | return No Solutions

   $x \leftarrow \text{tonelli\_shanks}(a, p)$ 
  for  $n = 1, \dots, \lceil \log_2(v) \rceil$  do
    |  $x \leftarrow x - (x^2 - a)/2x \pmod{p^{2^n}}$  // Hensel Lifting

  return  $x$ 

```

Algorithm 5: Square Roots mod p^v

The code for this can be found on page 34.

```

Function mod_n_sqrt( $a, n$ ):
  Data: Integers  $a$  and  $n$ 
  Result: Square root of  $a$  modulo  $n$  if it exists, otherwise No Solutions

  Factor  $n = \prod_{i=0}^k p_i^{v_i}$ 
  congruences  $\leftarrow []$ 
  for  $i = 0, \dots, k$  do
    |  $x \leftarrow \text{sqrt\_p\_adic}(a, p_i, v_i)$ 
    | if  $x$  is No Solutions :
      | | return No Solutions
    | congruences.append( $(x, p_i^{v_i})$ )
  return chinese_remainder_theorem(congruences)

```

Algorithm 6: Square Roots mod n

See page 35 for the code.

5. PROJECTIVE SPACE

There is a connection between conics and quadratic forms (defined in the next section: 6.1). We need to use projective space in order to fully understand this connection.

Definition 5.1. Let k be a field. The **projective space** of dimension n over k is $k\mathbb{P}^n = (k^{n+1} \setminus \{(0, \dots, 0)\}) / \sim$, where $(x_1, \dots, x_{n+1}) \sim (\lambda x_1, \dots, \lambda x_{n+1})$ for any $\lambda \in k^*$.

Given a rational conic $ax^2 + bxy + cy^2 + dx + ey + f = 0$ and a solution (x, y) , we may reduce x and y to have a common denominator. So we then have $(\frac{x}{z}, \frac{y}{z})$ as the same solution. If we multiply across by z , we **homogenize** the conic to obtain $ax^2 + bxy + cy^2 + dxz + eyz + fz^2 = 0$. Finding rational points on the original conic amounts to finding integer points on the homogenized conic.

However, note that if we have a solution (x, y, z) to the homogenized conic $ax^2 + bxy + cy^2 + dxz + eyz + fz^2 = 0$, then $(\lambda x, \lambda y, \lambda z)$ is another solution, for any $\lambda \in \mathbb{Q}^*$. Therefore solutions to the homogenized conic live in $\mathbb{Q}\mathbb{P}^2$. Finding integer solutions to the homogenized conic is the same as finding rational solutions, since we may simply multiply across by the common denominator.

This makes the conic much easier to work with, as it deals with a lot of special cases at once.

However, there is a slight problem which we shall have to deal with. If we find a point (x, y, z) on the homogenized conic which has $z = 0$, this will not correspond to any point on the original conic. It is a **point at infinity**: a point in $\mathbb{Q}\mathbb{P}^2$ which is not represented by the non-homogeneous conic. If we decide to work with the homogenized conic instead, we will have to deal with these points before outputting the final solution. However, the price of introducing these points is well worth the simplification.

In projective space, the notions of ellipses, parabolas and hyperbolas are essentially equivalent. A parabola is just an ellipse which is tangent to the line at infinity $z = 0$: it touches exactly one point at infinity. A hyperbola is an ellipse which goes through infinity and comes back on the other side. Depending on how you dehomogenize from projective space back to Euclidean space, a conic may end up being either an ellipse, a parabola or a hyperbola.

6. QUADRATIC FORMS

We can simplify a given conic significantly using the theory of quadratic forms. We will need to do this in order to find rational points on conics.

Definition 6.1. Let V be a finite dimensional vector space over a field k . A **quadratic form** is a map $Q : V \rightarrow k$ which satisfies the following conditions:

- (1) $Q(av) = a^2Q(v)$.
- (2) $Q(u + v) - Q(u) - Q(v)$ is a bilinear form.

Definition 6.2. Let $Q : V \rightarrow k$ be a quadratic form. Let $\{e_1, \dots, e_n\}$ be a basis of V . The **matrix of Q** w.r.t. $(e_i)_{i=1, \dots, n}$ is the matrix A whose entries are given by:

$$a_{i,j} = \frac{1}{2}(Q(e_i + e_j) - Q(e_i) - Q(e_j))$$

Note that every homogenized conic $ax^2 + bxy + cy^2 + dxz + eyz + fz^2$ is a quadratic form on \mathbb{Q}^3 . Finding nontrivial solutions (x, y, z) to the equation amounts to finding nonzero elements of the kernel of the quadratic form.

Theorem 6.3. (*Diagonalization of Quadratic Forms over any Field*)

Let $Q : V \rightarrow k$ be a quadratic form. Then there exists a basis $\{e_1, \dots, e_n\}$ such that the matrix of Q w.r.t. $(e_i)_{i=1, \dots, n}$ is diagonal.

Proof. Induction on n .

Base Case: $n = 0$ is trivial since an empty matrix is diagonal.

Inductive Step: Choose an element $e_n \in V$ such that $Q(e_n) \neq 0$.

If no such element exists, the form is identically zero, so any basis of V will make the matrix of Q be the zero matrix, which is diagonal.

Consider the subspace generated by e_n , denoted ke_n . This subspace has an orthogonal complement (orthogonal with respect to the direct product induced by Q) $H = (ke_n)^\perp$, which is a hyperplane of dimension $n - 1$.

By the Inductive Hypothesis, there is a basis (e_1, \dots, e_{n-1}) of H such that the matrix of Q restricted to H is diagonal. Then, since $V = H \oplus ke_n$, we have that (e_1, \dots, e_n) is a basis of V which makes the matrix of Q diagonal. \square

This is an important theorem for us, as it means we can linearly change variables to turn our conic $ax^2 + bxy + cy^2 + dxz + eyz + fz^2 = 0$ into a much simpler one: $ax^2 + by^2 + cz^2 = 0$. Note that the change of variables may lead to $a, b, c \in \mathbb{Q} \setminus \mathbb{Z}$. This is not an issue: we may simply multiply across by the common denominator

without affecting the solution set.

Unfortunately, this is as far as we can go. Although we can obtain an *orthogonal* basis over any field (which makes the matrix diagonal), we cannot obtain an *orthonormal* basis without introducing square roots. Over \mathbb{R} , we are limited only by Sylvester's Law of Inertia, which says that we can make the entries on the diagonal of the matrix into 1s, -1 s and 0s (if the conic is degenerate), and the number of each is determined by the quadratic form alone. Over \mathbb{C} , we can obtain the identity matrix for any non-degenerate quadratic form.

Definition 6.4. Let $Q : V \rightarrow k$ be a quadratic form. Let $\lambda \in k$ be a scalar. We say Q **represents** λ if $\exists x \in V : Q(x) = \lambda$.

See chapter IV of [5] (on page 27) for more information on Quadratic Forms.

7. THE HASSE-MINKOWSKI THEOREM

Suppose we have a rational conic $ax^2 + bxy + cy^2 + dx + ey + f = 0$. We may homogenize to obtain a quadratic form, and then diagonalize this form using 6.3 to obtain an equation $ax^2 + by^2 + cz^2 = 0$. By multiplying across by denominators, we may assume $a, b, c \in \mathbb{Z}$.

It is finally time to solve this conic. Before the main theorem of this chapter, however, there is a further simplification that we must do.

Firstly, observe that if any of $a, b, c = 0$, we have a solution: $(1, 0, 0)$ or $(0, 1, 0)$ or $(0, 0, 1)$. Note that if the solution has $z = 0$ we will have a point at infinity, but this is still perfectly adequate for using the line intersection property 1.3 in $\mathbb{Q}\mathbb{P}^2$ to obtain solutions to the original rational conic which are not at infinity.

On the other hand, if all $a, b, c > 0$ or $a, b, c < 0$, then it is clear that there are no solutions over \mathbb{Q} .

After dealing with these cases, we have either two of a, b, c positive and one negative, or two of a, b, c negative and one is positive. We may assume the former case by multiplying across by -1 . We may also change variables so that a and b are positive.

We may also make it so that a, b , and c are square-free by changing variables. (We will have to keep track of this change as it affects the solution set.)

Now suppose that (x, y, z) is a solution to the equation $-acx^2 - bcy^2 - z^2 = 0$.

In this case, it is clear that $c|z^2$, since it divides all the other terms in the equation. But we changed variables so that c is square-free, so $c|z$.

We have already dealt with the case $c = 0$ above so we may assume that it is nonzero. Dividing $-acx^2 - bcy^2 - z^2 = 0$ across by $-c$ yields $ax^2 + by^2 + \frac{z^2}{c} = 0$. When we plug $(x, y, \frac{z}{c})$ into $ax^2 + by^2 + cz^2 = 0$ we get $ax^2 + by^2 + \frac{z^2}{c} = 0$. This means that $(x, y, \frac{z}{c})$ is a solution to $ax^2 + by^2 + cz^2 = 0$ whenever (x, y, z) is a solution to $-acx^2 - bcy^2 - z^2 = 0$.

By this reasoning, we have reduced $ax^2 + by^2 + cz^2 = 0$ to $ax^2 + by^2 = z^2$ via $a \leftarrow -ac$, $b \leftarrow -bc$. Note that a and b are still positive because c was negative.

We have now reduced our original conic to the equation $ax^2 + by^2 = z^2$. Finally, we may change variables one last time to ensure that $|a| \leq |b|$.

Theorem 7.1. (*Hasse-Minkowski Theorem: Local-Global Principle Version*)

Let C be a rational conic. Then C has a nontrivial solution over \mathbb{Q} if and only if it has a nontrivial solution over \mathbb{Q}_p for all primes p and for $\mathbb{Q}_\infty = \mathbb{R}$.

Proof. It is clear that having a solution over \mathbb{Q} means that we have one over \mathbb{Q}_p , since $\mathbb{Q} \subset \mathbb{Q}_p$.

The other direction is much more interesting. Assume that C has a nontrivial solution over \mathbb{Q}_p for all places p (including $\mathbb{Q}_\infty = \mathbb{R}$).

We may perform the reduction described at the start of this section to obtain an equation $ax^2 + by^2 = z^2$, with a and b square-free, nonzero integers and $|a| \leq |b|$. There is one problematic step, where we said that all $a, b, c > 0$ or $a, b, c < 0$ means that there are no solutions over \mathbb{Q} . However, in this case, there are no solutions over \mathbb{R} either, which contradicts our assumptions.

We now use induction on $m = |a| + |b|$. (Although we reduced to the case where a and b are positive in the above construction, we cannot assume this for the induction.)

The smallest case is $m = 2$, since a and b are nonzero. In this case, we have $\pm x^2 \pm y^2 = z^2$. If both a and b are -1 , there are no solutions over $\mathbb{Q}_\infty = \mathbb{R}$, which contradicts our assumptions. Otherwise, at least one of a and b is 1 , so either $(1, 0, 1)$ or $(0, 1, 1)$ will be a solution.

For $m > 2$, begin by factoring $b = p_1 \dots p_k$. b is square-free, so these primes are all distinct.

We would like to prove that a is square mod p_i for any one of the primes dividing b . If $a \equiv 0 \pmod{p_i}$, then we are done, since $a = 0^2$.

Otherwise, by assumption there is a nontrivial solution (x, y, z) to $ax^2 + by^2 = z^2$ in \mathbb{Q}_{p_i} . If x, y and z are all divisible by p_i , we may divide across by p_i to obtain a new solution. Therefore assume that they are not all divisible by p_i .

Since $p_i | b$, we have $ax^2 \equiv z^2 \pmod{p_i}$. If $x^2 \equiv 0 \pmod{p_i}$, then $x \equiv 0 \pmod{p_i}$ also. But then also $z \equiv 0 \pmod{p_i}$, and since $ax^2 + by^2 = z^2$ with b square-free, this means that $p_i | y$. However, this contradicts our assumption that x, y and z are not all divisible by p_i .

Therefore, we may legally divide $ax^2 \equiv z^2 \pmod{p_i}$ by x^2 to obtain $a \equiv \frac{z^2}{x^2} \pmod{p_i}$, which tells us that a is square mod p_i .

We may apply the same reasoning to all the $p_i | b$. We may then apply the Chinese Remainder Theorem to show that a is square mod b .

Therefore, we may find a t such that $t^2 \equiv a \pmod{b}$, i.e. $t^2 = a + bb'$ for some b' . We may choose our representative t of \sqrt{a} in $\mathbb{Z}/b\mathbb{Z}$ such that $|t| \leq \frac{|b|}{2}$.

Observe that bb' is the norm (2.3) of $t + \sqrt{a}$. If $ax^2 + by^2 = z^2$ has a nontrivial solution, then y cannot be 0 or a would have to be square, which contradicts our assumption that a was square-free (unless $a = 1$, in which case $(1, 0, 1)$ is a solution). Hence b is a norm of $\frac{z}{y} + \frac{x}{y}a$.

But the space of norms is closed under multiplication (2.4). So b is a norm if and only if b' is a norm, i.e. $ax^2 + by^2 = z^2$ has a nontrivial solution (over either \mathbb{Q}_p or \mathbb{Q}) if and only if $ax^2 + b'y^2 = z^2$ has a nontrivial solution (over \mathbb{Q}_p or \mathbb{Q} respectively).

However, we have:

$$\begin{aligned}
 |b'| &= \left| \frac{t^2 - a}{b} \right| \\
 &\leq \left| \frac{\frac{|b|^2}{4} - b}{b} \right| \quad \dots (\text{since } |t| \leq \frac{|b|}{2} \text{ and } |a| \leq |b|) \\
 &\leq \frac{|b|}{4} + 1 \\
 &< |b| \quad \dots (\text{since } |b| \geq 2).
 \end{aligned}$$

We must change variables so that b' is square-free and $|a| \leq |b'|$, and then we can finally apply the induction hypothesis.

□

The more general version of this is the **Local-Global Principle**, which says that an equation has a nontrivial ‘global’ solution over a number field if and only if it has a nontrivial ‘local’ solution for every completion of that number field. It is a ‘Principle’ and not a theorem because it does not always hold. For example, it does not hold for cubics.

See [6] for a more algebraic geometry flavoured approach to this theorem. It is Theorem 1.1 on page 2.

Although we only proved for conics in 2 variables, the Hasse-Minkowski Theorem actually holds for any number of variables. See [5] for a proof of this. It is Theorem 8 in section 3 of chapter IV (on page 41). In fact, the proof of this more general version can also be made constructive to find rational points on conics in an arbitrary number of variables (or prove that none exist). Unfortunately, this is beyond the scope of this project.

We are particularly interested in the proof, since it can be made constructive to actually find a solution. Once we have one solution, we can use the Line-Intersection property to find all solutions. We can then do all of our previous steps in reverse to get solutions to the original conic over \mathbb{Q} .

In order to turn the induction step into a recursive algorithm, we need to explicitly determine the change of variables to go backwards from a solution (p, q, r) of $ap^2 + b'q^2 = r^2$ to a solution (x, y, z) of the equation from the previous step $ax^2 + by^2 = z^2$.

Recall that $ap^2 + b'q^2 = r^2$ makes b' a norm of $\frac{r}{q} + \frac{p}{q}\sqrt{a}$, while $ax^2 + by^2 = z^2$ makes b a norm of $\frac{z}{y} + \frac{x}{y}\sqrt{a}$. Also remember that bb' was the norm of $t + \sqrt{a}$.

$$\begin{aligned}
b &= \frac{bb'}{b'} \\
&= \frac{N(t + \sqrt{a})}{N\left(\frac{r}{q} + \frac{p}{q}\sqrt{a}\right)} \\
&= N\left(q \frac{t + \sqrt{a}}{r + p\sqrt{a}}\right) \\
&= N\left(q \frac{(t + \sqrt{a})(r - p\sqrt{a})}{(r + p\sqrt{a})(r - p\sqrt{a})}\right) \\
&= N\left(q \frac{tr - pa + r\sqrt{a} - tp\sqrt{a}}{r^2 - p^2a}\right) \\
&= N\left(\frac{tr - pa + r\sqrt{a} - tp\sqrt{a}}{qb'}\right) \quad \dots(\text{since } r^2 - p^2a = q^2b') \\
&= \left(\frac{1}{qb'}\right)^2 N((tr - pa) + (r - tp)\sqrt{a})
\end{aligned}$$

$$a(r - tp)^2 + b(b'q)^2 = (-ap + tr)^2$$

Therefore the change of variables is $(p, q, r) \mapsto (r - tp, b'q, -ap + tr)$.

```

Function hasse_minkowski ( $a, b$ ) :
  Data: Integers  $a$  and  $b$ 
  Result: Solution  $(p, q, r)$  or No Solutions
   $a, b$  should be made square-free. Keep track of this transformation in a
  matrix  $T$ 
  if  $|a| > |b|$  :
     $a, b \leftarrow b, a$ 
     $T \leftarrow \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot T$ 

  // Terminating Conditions:
  if  $a = 1$  :
    return  $T^{-1} \cdot \text{vector}(\mathbb{Q}, [1, 0, 1])$ 
  else if  $b = 1$  :
    return  $T^{-1} \cdot \text{vector}(\mathbb{Q}, [0, 1, 1])$ 
  else if  $|a| + |b| \leq 2$  :
    return No Solutions

  if not is_square ( $a \bmod b$ ) :
    return No Solutions
   $t \leftarrow \text{mod\_n\_sqrt}(a, b)$ 
  Ensure  $-|b|/2 \leq t \leq |b|/2$ 

   $b' \leftarrow (t^2 - a)/b$ 
   $\text{solution} \leftarrow \text{hasse\_minkowski}(a, b')$ 
  if  $\text{solution}$  is No Solutions :
    return No Solutions
   $[p, q, r] \leftarrow \text{solution}$ 
  return  $T^{-1} \cdot \begin{pmatrix} -t \cdot p + r \\ b' \cdot q \\ -a \cdot p + t \cdot r \end{pmatrix}$ 

```

Algorithm 7: Hasse-Minkowski Algorithm

See page 38 for the code.

Example 7.2. We test our code on the conic $83x^2 + 32xy - 2y^2 + \frac{1}{2}x + 25y + 74 = 0$. Our program finds the initial point $(x = 652088813, y = -2722553580, z = -379326094)$. We get the very nasty parametrization:

$$x = \frac{96509144324m^2 + 9353476489873170318mn + 226630137121767097769936636n^2}{191615852099m^2 + 18571031587203766512mn + 449966965722038916702679352n^2}$$

$$y = \frac{-402937929840m^2 - 39051951812812396527mn - 946209594239363097256153500n^2}{191615852099m^2 + 18571031587203766512mn + 449966965722038916702679352n^2}$$

Example 7.3. The Hasse-Minkowski theorem does not work for higher degree curves. For example, Selmer showed that $3x^3 + 4y^3 + 5z^3 = 0$ has nontrivial solutions over \mathbb{R} and over \mathbb{Q}_p for all primes p , but no nontrivial solutions over \mathbb{Q} .

There is another version of the theorem, which is less suited to actually computing solutions. However, it is slightly better if one simply wants to test if a conic has solutions, and isn't concerned about finding them. We need to perform the simplification at the beginning of this section in order to get the conic into a form where we can apply this theorem.

Theorem 7.4. (*Hasse-Minkowski Theorem: Legendre's Version*)

Let a and b be positive square-free integers with $|a| \leq |b|$. Then $ax^2 + by^2 = z^2$ has a nontrivial solution over \mathbb{Z} if and only if:

- (1) a is a square mod b
- (2) b is a square mod a
- (3) $-\frac{ab}{d}$ is a square mod d , where $d = \gcd(a, b)$

This version of the theorem was originally discovered by Legendre. See [4] for a proof of this theorem. It is Proposition 17.3.1 on page 273.

8. HILBERT SYMBOLS

Definition 8.1. Let k be a field. Let $a, b \in k^\times$. We define the **Hilbert symbol** (a, b) to be 1 if $ax^2 + by^2 - z^2 = 0$ has a nontrivial solution (x, y, z not all 0), and -1 otherwise.

We are mainly interested in the case $k = \mathbb{Q}_p$ for p prime or ∞ . Let us denote the Hilbert symbol over these fields by $(a, b)_p$.

Lemma 8.2. (*Properties of the Hilbert Symbol*)

The Hilbert symbol satisfies the following properties:

- (1) $(c^2a, b) = (a, b)$ (*Invariance under Squares*)
- (2) $(a, b) = (b, a)$ (*Symmetry*)
- (3) $(a, c^2) = 1$
- (4) $(a, -a) = 1$
- (5) $(a, 1 - a) = 1$
- (6) For \mathbb{Q}_p or \mathbb{R} , $(aa', b) = (a, b)(a', b)$ (*Bilinearity*)

Proof.

- (1) $(c^2a, b) = 1$ if $c^2ax^2 + by^2 = z^2$ has a nontrivial solution else -1 . However, we can change variables $x_1 = \frac{x}{c}$ to get $ax_1^2 + by^2 = z^2$. This gives a bijection between solutions of $c^2ax^2 + by^2 = z^2$ and solutions of $ax_1^2 + by^2 = z^2$. Hence $c^2ax^2 + by^2 = z^2$ has a nontrivial solution iff $ax_1^2 + by^2 = z^2$ has a nontrivial solution. Therefore the Hilbert symbols are equal: $(c^2a, b) = (a, b)$.
- (2) It is clear that $ax^2 + by^2 = z^2$ has a nontrivial solution iff $bx^2 + ay^2 = z^2$ has a nontrivial solution.
- (3) $(a, c^2) = 1$ iff $ax^2 + c^2y^2 = z^2$ has a nontrivial solution. One can verify that $x = 0, y = 1, z = c$ is a solution.
- (4) $(a, -a) = 1$ iff $ax^2 - ay^2 = z^2$ has a nontrivial solution. One such solution is $x = 1, y = 1, z = 0$.
- (5) $(a, 1 - a) = 1$ iff $ax^2 + (1 - a)y^2 = z^2$ has a solution. $x = 1, y = 1, z = 1$ satisfies this equation.
- (6) The proof of this is quite long. See [5], Chapter III, Section 1.2, Theorem 2 (on page 20).

□

By (1) Invariance under Squares, multiplying either a or b by a square has no effect on the Hilbert symbol. Hence the Hilbert symbol gives us a map $k^\times/(k^{\times 2}) \times k^\times/(k^{\times 2}) \rightarrow \{1, -1\}$.

Recall in Theorem 3.11 we discovered that $\mathbb{Q}_p^\times/\mathbb{Q}_p^{\times 2}$ could be realized as a vector space over $\mathbb{Z}/2\mathbb{Z}$, which had dimension 2 for $p \neq 2$ and dimension 3 for $p = 2$. However, since we realized $\mathbb{Z}/2\mathbb{Z}$ as the multiplicative group $\{1, -1\}$, “addition” in our vector space corresponds to elementwise multiplication, and “scalar multiplication” by an element in $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ corresponds to taking powers of each component.

Note that $\mathbb{Q}_\infty^\times/\mathbb{Q}_\infty^{\times 2} = \mathbb{R}^\times/\mathbb{R}^{\times 2}$ can also be realized as a vector space over $\mathbb{Z}/2\mathbb{Z}$. An element of \mathbb{R}^\times is square iff it is positive, so we realize $\mathbb{R}^\times/\mathbb{R}^{\times 2} \cong \{1, -1\}$ via $x \mapsto \text{sign}(x)$. Hence this is a vector space of dimension 1.

Since the Hilbert symbol gives us a map $k^\times/(k^{\times 2}) \times k^\times/(k^{\times 2}) \rightarrow \{1, -1\}$, and by 8.2 (2) and (6), for $k = \mathbb{Q}_p$ or \mathbb{R} , the Hilbert symbol is a symmetric bilinear form! Hence there exists a symmetric matrix A such that $(a, b) = aAb$ where a and b are written in their vector form in the vector space $k^\times/(k^{\times 2})$. It is easier to work in vector spaces over the additive field $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ and then take powers of -1 .

Lemma 8.3. *We may write $(a, b)_p = (-1)^{aAb}$ for p prime or ∞ , and a and b are written as vectors in the (additive) vector space $\mathbb{Q}_p^\times/\mathbb{Q}_p^{\times 2}$ over $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ (with the same bases as in 3.11 only additive now, i.e. 1 replaced with 0 and -1 replaced with 1). The matrix A is given by:*

- $A = (1)$ if $p = \infty$

- $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ if $p \equiv 1 \pmod{4}$
- $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ if $p \equiv 3 \pmod{4}$
- $A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ if $p = 2$

This lemma gives us a formula for computing Hilbert symbols over \mathbb{Q}_p . See 25 for the code.

One might also wonder how to compute them over \mathbb{Q} . By the Hasse-Minkowski Theorem 7.1, we have $(a, b) = 1$ over \mathbb{Q} iff $(a, b)_p = 1$ for all p prime or ∞ . However, checking all of the infinitely many prime numbers is not really possible. Fortunately, by the following theorem, we only need to check finitely many primes to get Hilbert symbols over \mathbb{Q} .

Theorem 8.4. *We have $(a, b)_p = 1$ for all primes p with $p \nmid 2ab$ (note that ∞ is not included among these). Hence the number of places p for which the conic $ax^2 + by^2 = z^2$ fails to have solutions is finite.*

Proof.

- Suppose $p \nmid 2ab$.
- We firstly look for a nontrivial solution to the dehomogenized conic $ax^2 + by^2 = 1$ in $\mathbb{Z}/p\mathbb{Z}$. If either a or b is 0 the conic is degenerate, which is not allowed. Hence assume that they are nonzero.
- $\#(\mathbb{Z}/p\mathbb{Z})^\times = p - 1$. $\#(\mathbb{Z}/p\mathbb{Z})^{\times 2} = \frac{p-1}{2}$. $\#(\mathbb{Z}/p\mathbb{Z})^2 = \frac{p-1}{2} + 1 = \frac{p+1}{2}$ since 0 is also a square in $\mathbb{Z}/p\mathbb{Z}$.
- Set $A = \{ax^2 | x \in \mathbb{Z}/p\mathbb{Z}\}$, $B = \{1 - by^2 | y \in \mathbb{Z}/p\mathbb{Z}\}$. Since a and b are invertible (being nonzero) we have $\#A = \#B = \frac{p+1}{2}$.
- By the Pigeon Hole Principle, A and B have an element t in common. Then $t = ax^2 = 1 - by^2$, so $ax^2 - (1 - by^2) = t - t = 0$, i.e. $ax^2 + by^2 = 1$. So we have found a solution mod p . Note also that x and y cannot both be 0.
- We now wish to use Hensel lifting 3.9 to get a solution in \mathbb{Q}_p . Let x_0, y_0 denote the solution we found mod p . If x_0 is 0, we may swap the roles of x_0 and y_0 (since they cannot both be 0), so assume $x_0 \not\equiv 0 \pmod{p}$.
- Fix a representative y of y_0 in \mathbb{Q}_p . We then have $f(x_n) = ax_n^2 + by^2 - 1$, and $f'(x_0) = 2ax_0$. Since $p \nmid 2a$ and $x_0 \not\equiv 0 \pmod{p}$, $f'(x_0) \not\equiv 0 \pmod{p}$, so Hensel's Lemma applies.

□

Given a conic which does not have solutions over \mathbb{Q} (e.g. the Hasse-Minkowski algorithm returned *No Solutions*), we can use the above theorem to find the values of p for which the conic fails to have solutions. We first diagonalize our conic

to obtain $ax^2 + by^2 = cz^2$, and divide by c to get $\frac{a}{c}x^2 + \frac{b}{c}y^2 = z^2$, which has solutions over \mathbb{Q}_p iff $(\frac{a}{c}, \frac{b}{c})_p = 1$. Since Hilbert symbols are invariant over squares, $(\frac{a}{c}, \frac{b}{c})_p = (ac, bc)_p$. We only need to check p with $p = \infty$ or $p|2abc$: all other p give $(ac, bc)_p = 1$.

See [30](#) for the code.

Example 8.5. The conic:

$$174839507349205743885342098x^2 + 100924631243586773741746381789y^2 \\ + 320478392574389207819834921048921048 = 0$$

is not locally solvable over \mathbb{R} or \mathbb{Q}_p with $p = 2, 7, 953, 5737, 13745419, 26606347, 94481687, 1196385979, 8304347251$

Interestingly, the number of places will always be even. For a proof of this, see Theorem 3 in Section 2.1 of Chapter III in [\[5\]](#).

9. ISOMORPHIC QUADRATIC FORMS AND CONICS

We would now like to consider the question of when (nondegenerate) quadratic forms and conics are \mathbb{Q} -isomorphic to one another.

Definition 9.1. Let k be a field. Let Q and Q' be quadratic forms defined on n -dimensional vector spaces over k and let A and A' be their respective matrices. We say Q and Q' are **k -isomorphic** if there exists a matrix $T \in GL_n(k)$ such that $A' = TAT^{-1}$.

Definition 9.2. Let k be a field. Let C and C' be conics defined over k and let A and A' be matrices in $PGL_n(k)$ for each of them. We say C and C' are **k -isomorphic** if there exists a matrix $T \in GL_n(k)$ such that $A' = TAT^{-1}$. \mathbb{Q} -isomorphic rational conics are sometimes called **equivalent**.

All (nondegenerate) conics are \mathbb{C} -isomorphic to one another. In addition, the proof of the Hasse-Minkowski Theorem [7.1](#) reduces all nondegenerate conics which have nontrivial solutions in \mathbb{Q} to $x^2 + y^2 = z^2$, so they are all \mathbb{Q} -isomorphic to one another. The other cases are less obvious. In fact, there are infinitely many isomorphism classes of nondegenerate conics which do not have solutions over \mathbb{Q} .

We begin with the case of isomorphic quadratic forms.

Definition 9.3. Let Q be a nondegenerate quadratic form on a vector space V over \mathbb{R} , and let A be the matrix of Q with respect to some basis. The **inertia** of Q is the pair (n_+, n_-) of the number of positive and negative eigenvalues of A .

Sylvester's Law of Inertia tells us that the inertia is independent of the choice of basis. Conversely, two quadratic forms over a real vector space are \mathbb{R} -isomorphic iff they have the same inertia.

Definition 9.4. Let Q be a nondegenerate, rational quadratic form on a vector space V over a field k . Let A be the matrix of Q corresponding to any basis of V . We define the **discriminant** $d(Q)$ to be the determinant of the matrix A in $k^\times/k^{\times 2}$.

For $k = \mathbb{Q}_p$, we denote the discriminant of a quadratic form Q by $d_p(Q)$.

The discriminant is well defined because if T is a change of basis matrix, the matrix of Q wrt. the new basis is TAT^t , and this has determinant $\det(T)^2 \det(A) = \det(T)^2 d(Q) = d(Q)$ in $k^\times/k^{\times 2}$. Hence the discriminant is invariant under isomorphism of quadratic forms.

Definition 9.5. Let Q be a nondegenerate, rational quadratic form on a vector space V over \mathbb{Q}_p . Let A be the matrix of Q corresponding to some orthogonal basis of V . Note that A is diagonal here since our basis is required to be orthogonal. Let a_i be the entries along its diagonal. The **invariant** $\varepsilon_p(Q)$ is defined as $\varepsilon_p(Q) = \prod_{i < j} (a_i, a_j)_p$.

For a proof that this does not depend on the choice of basis, see [5] Chapter IV, Section 2.1, Theorem 5 (on page 35).

Remark 9.6. In dimension 3, we have $\varepsilon_p(Q) = (a_1, a_2)_p (a_1, a_3)_p (a_2, a_3)_p$.

Theorem 9.7. Let Q and Q' be two quadratic forms on a vector space V over \mathbb{Q}_p . Then they are \mathbb{Q}_p -isomorphic to one another iff they have the same discriminant d_p and the same invariant ε_p .

Theorem 9.8. Let Q and Q' be two quadratic forms on a vector space V over \mathbb{Q} . Then they are \mathbb{Q} -isomorphic to one another iff they have the same inertia, the same discriminant d and the same invariant ε_p for all p (where we view Q as a quadratic form over \mathbb{Q}_p).

For a proof of the first theorem, see [5] Chapter IV, Section 2.3, Theorem 7 (on page 39). For the second, see [5] Chapter IV, Section 3.3, Theorem 9 and its Corollary (on page 44).

Remark 9.9. In order to test whether two quadratic forms are isomorphic over \mathbb{Q} , we only need to check the ε_p for $p = 2, p \mid \prod a_i$ and $p \mid \prod b_i$ (where a_i and b_i denote the entries on the diagonal of a diagonalized matrix of each of the forms). Indeed, by 8.4, we have $(a_i, a_j)_p = 1$ for $p \nmid 2a_i a_j$ and the ε_p are just products of Hilbert symbols.

We present an algorithm to test if two diagonal quadratic forms of dimension 3 are \mathbb{Q} -isomorphic. The algorithm readily extends to higher dimensions. We can also diagonalize arbitrary forms using 6.3.

```

Function is_isomorphic_as_forms ( $a_1, b_1, c_1, a_2, b_2, c_2$ ) :
  Data: Coefficients  $a_1, b_1, c_1$  and  $a_2, b_2, c_2$  of the diagonalized forms
  Result: True if forms are isomorphic, False otherwise
   $d_1 \leftarrow a_1 \cdot b_1 \cdot c_1$ 
   $d_2 \leftarrow a_2 \cdot b_2 \cdot c_2$ 
  if  $d_1 = 0$  or  $d_2 = 0$  :
    | return False // Forms must be nondegenerate
  if  $\text{inertia}(a_1, b_1, c_1) \neq \text{inertia}(a_2, b_2, c_2)$  :
    | return False
  if not  $\text{is\_square}(\mathbb{Q}(\frac{d_1}{d_2}))$  :
    | return False
  for  $p$  in  $\text{prime\_divisors}(2 \cdot d_1 \cdot d_2) + [\infty]$  do
    | if not  $\varepsilon(a_1, b_1, c_1, p) = \varepsilon(a_2, b_2, c_2, p)$  :
      | return False
  return True

```

Algorithm 8: Isomorphic Quadratic forms

See page 46 for the code.

What about isomorphic conics? The quadratic form corresponding to a conic is defined up to multiplication by a scalar $\lambda \in \mathbb{Q}^\times$. Hence, neither the discriminant nor the invariants ε_p are well defined for a given conic.

Let C and C' be conics. Let Q and Q' be quadratic forms corresponding to C and C' . Then C and C' are \mathbb{Q} -isomorphic to each other iff λQ and Q' are \mathbb{Q} -isomorphic to each other for some scalar $\lambda \in \mathbb{Q}^\times$.

Clearly, we must pick λ so that the discriminants $d(\lambda Q)$ and $d(Q')$ match. Hence $\lambda = \frac{d(Q')}{d(Q)}$. Then we can simply use our algorithm 8 to check if these two quadratic forms are isomorphic.

Theorem 9.10. *Let C and C' be rational conics. Let Q and Q' be any quadratic forms corresponding to C and C' respectively. Then C and C' are \mathbb{Q} -isomorphic to one another iff Q and $\frac{d(Q)}{d(Q')}Q'$ are \mathbb{Q} -isomorphic to one another.*

We now come to a very interesting question: Given two quadratic forms Q and Q' which are isomorphic, how can one actually find an isomorphism between them? Although we have been mostly working with conics in 2 variables/quadratic forms in 3 variables, pretty much all of the theory generalizes to higher dimensions. We will need to make use of this now.

Theorem 9.11. *Let Q and Q' be quadratic forms in n variables over \mathbb{Q} . Then Q and Q' are \mathbb{Q} -isomorphic if and only if they represent (see 6.4) the same set of $\lambda \in \mathbb{Q}$.*

Let Q and Q' be isomorphic quadratic forms over V . Let e_1, \dots, e_n be a basis of V such that $Q(e_i) = \lambda_i \neq 0$ for some $\lambda_1, \dots, \lambda_n$. We can always find such a collection of e_i because the dimension of the set where the conic is 0 (if it exists) is always less than the dimension of the space V .

By the above theorem, since Q is isomorphic to Q' , it must represent the same λ_i , i.e. there is an e'_n such that $Q'(e'_n) = \lambda_n$.

How can we find such an e'_n ? The Hasse-Minkowski theorem! It has a generalization which can be made constructive to find solutions to a conic in any number of variables.

We have Q' represents λ_n iff $\tilde{Q}'(x_1, \dots, x_n, z) = Q'(x_1, \dots, x_n) - \lambda_n z^2$ has a non-trivial solution. We can use the multivariable Hasse-Minkowski theorem to find a solution to this and then set $(e'_n)_i = x_i/z$ ⁴ to obtain a vector $e'_n \in V$ such that $Q'(e'_n) = \lambda_n = Q(e_n)$.

We then restrict the forms to the quotient spaces $V/\text{span}(e_n)$ and $V/\text{span}(e'_n)$ and do the same thing in one dimension lower. We can then use induction to find a basis e'_1, \dots, e'_n of V such that $Q(e_i) = Q(e'_i)$. The isomorphism is the linear change of variables which sends e_i to e'_i .

APPENDIX: CODE

I have omitted most imports for the sake of simplicity, as the file structure is not needed to understand the code. If the reader intends to run the code, putting all of it in a single file will work, but this is not recommended as the file would be very long and hence difficult to manage. When a function makes use of another function from a different file, it will need to be imported. In addition, all of the files require the line:

```
from sage.all import *
```

This is my code for the Extended Euclidean Algorithm 1. It is slightly different from the pseudocode as the latter omits some important special cases, such as a or b being negative. We also return 0 for $\gcd(0, 0)$, although it is not defined in this case.

⁴When we have a nondegenerate conic with nontrivial solutions, not all of them can be at infinity. To see this, note when we parameterize solutions to the homogeneous conic we get $x_i = f_i(m_1, \dots, m_{n-1})$ where f_i is a nondegenerate homogeneous conic. Points at infinity are those with m_i such that $f_n(m_1, \dots, m_{n-1}) = 0$, which cannot be the whole space $\mathbb{Q}\mathbb{P}^{n-1}$ because f_n is nondegenerate. For example, conics in dimension 2 have at most 2 points at infinity. Hence we can always find solutions which do not have $z = 0$.

```
def extended_euclidean(a, b):
    if abs(b) > abs(a):
        d, v, u = extended_euclidean(b, a)
        return d, u, v

    elif b == 0:
        if a == 0:
            return 0, 1, 1
        return abs(a), ZZ(a/abs(a)), 0

    else:
        d, u, v = extended_euclidean(b, a % b)
        if d > 0:
            return d, v, u - floor(a/b) * v
        else:
            return -d, -v, -u + floor(a/b) * v
```

This is the code for the Chinese Remainder Theorem. See [2](#) for an explanation.

```
def crt(congruences):
    """
    Congruences: a list of tuples (n_i, x_i)
    Finds x mod n = n_1n_2...n_k which satisfies all the
    congruences."""
    N = 1
    for _, n_i in congruences:
        N *= n_i

    x = 0
    for x_i, n_i in congruences:
        N_i = N / n_i
        d, u, v = extended_euclidean(n_i, N_i)
        if d != 1:
            raise ValueError('The moduli are required to
                               be pairwise coprime.')
        x += x_i * N_i * v
    return x
```

This is the code for the Tonelli-Shanks Algorithm. See [3](#).

```
import random

def tonelli_shanks(a, p):
    S = IntegerModRing(p)
    a = S(a)
```



```

if p == 2:
    return sqrt_mod_2(a)
elif a == 0:
    return 0
elif kronecker(a, p) == -1:  # Check if a is not
    square mod p
    return None
else:
    e = valuation(p-1, 2)
    q = (p-1) / 2**e

    legendre = 0
    while legendre != -1:
        z = S(random.randrange(2, p))
        legendre = kronecker(z, p)
    b = z**q  # b is in S

    r = a**((q+1)/2)
    t = a**q
    if t == 1:
        return r
    else:
        m = e-1
        while m > 0:
            if t**(2**(m-1)) == -1:
                r *= b
                t *= b**2
            b = b**2
            m -= 1
    return r

```

What follows is the code for square roots mod 2^v . See 4.

```

def sqrt_mod_2(a):
    a = IntegerModRing(4)(a)
    return ZZ(a)  # a is its own square root mod 2
                  # because 0*0 = 0 and 1*1 = 1

def sqrt_mod_4(a):
    a = IntegerModRing(4)(a)
    if a == 0 or a == 1:
        return ZZ(a)
    if a == 2 or a == 3:
        return None

```

```

def sqrt_mod_8(a):
    a = IntegerModRing(8)(a)
    if a == 0 or a == 1:
        return ZZ(a)
    else:
        return None

def sqrt_2_adic(a, v):
    '''Computes the square root of a mod 2**v using
    Strong Hensel's Lemma.'''
    if IntegerModRing(2**v)(a) == 0:
        return 0

    v0 = valuation(a, 2)
    if v0 % 2 == 1:
        return None

    if v == 1:
        return sqrt_mod_2(a)
    elif v == 2:
        return sqrt_mod_4(a)

    a = ZZ(IntegerModRing(2**v)(a))
    R = Zp(2, prec = v+ceil(log(v, 2).n())) # A higher
    precision is needed because precision is lost when
    dividing by 2
    u = R(a / 2**v0)
    S = IntegerModRing(8)

    u0 = sqrt_mod_8(u)
    if u0 is None:
        return None

    xn = R(u0)
    current_precision = 3 # We have a square root mod
    8=2**3
    while current_precision < v:
        xn = xn - R((xn*xn-u) / (2*xn))
        current_precision = 2*current_precision - 2

    return xn * 2**(v0/2)

```

Here is the code for square roots mod p^v . See 5 for an explanation.

```

def sqrt_p_adic(a, p, v):

```

```

'''Algorithm to compute p-adic square roots using
Hensel's Lemma. v is precision.'''
if IntegerModRing(p**v)(a) == 0:
    return 0

v0 = valuation(a, p)
if v0 % 2 == 1:
    return None
if p == 2:
    return sqrt_2_adic(a, v)

R = Zp(p, prec=v)
u = R(a / p**v0)
S = IntegerModRing(p)
u0 = S(u)
if not is_square(u0):
    return None

xn = R(ZZ(tonelli_shanks(u0, p))) # A conversion to
ZZ is needed to maintain precision
for i in range(ceil(log(v, 2).n())):
    xn = xn - R((xn*xn-u) / (2*xn))

return xn * p**(v0/2)

```

Here is the code for square roots mod n . See 6 for an explanation.

```

def sqrt_mod_n(a, n):
    S = IntegerModRing(n)
    congruences = []
    for p in prime_divisors(n):
        v = valuation(n, p)
        R = IntegerModRing(p**v)
        x = sqrt_p_adic(a, p, v)
        if x is None:
            return None
        x = ZZ(x)
        congruences.append((x, p**v))
    return ZZ(S.crt(congruences))

```

In what follows, we need the lines:

```

var('x, y, z, m, n')
R = QQ['x, y']
x, y = R(x), R(y)

```

This function is used to test if a conic is degenerate 1.2. We are not really concerned with degenerate conics so we use this function to detect those edge cases and output the relevant messages.

```
def test_degenerate(coeffs):
    a, b, c, d, e, f = coeffs
    A = matrix(QQ, [[a, b/2, d/2],
                    [b/2, c, e/2],
                    [d/2, e/2, f]])
    return det(A) == 0
```

This function simply extracts the coefficients from a polynomial.

```
def coefficients(f, diagonal=True):
    if not (f in R or f in S):
        raise ValueError('f should be in either QQ[x,y]
                          or ZZ[x,y,z], not '+str(f))

    a = f.monomial_coefficient(R(x**2))
    b = f.monomial_coefficient(R(x*y))
    c = f.monomial_coefficient(R(y**2))
    d = f.monomial_coefficient(R(x))
    e = f.monomial_coefficient(R(y))
    f = f.monomial_coefficient(R(1))
    return [a, b, c, d, e, f]
```

This function is used to change variables $x \mapsto y = Tx$. In order to have $Q_1(x) = x^t Ax$ equal $Q_2(y) = (Tx)^t B Tx$, we need the new matrix to be $B = (T^{-1})^t A T^{-1}$.

```
def transform_conic(coeffs, T):
    '''Changes variables x -> Tx.
    Returns the coefficients of the conic in the new
    vector space.'''
    a, b, c, d, e, f = coeffs
    A = matrix(QQ, [[a, b/2, d/2],
                    [b/2, c, e/2],
                    [d/2, e/2, f]])
    new_A = T.inverse().transpose() * A * T.inverse()
    a, b, c, d, e, f = new_A[0,0], 2*new_A[0,1], new_A
    [1,1], 2*new_A[0,2], 2*new_A[1,2], new_A[2,2]
    return [a, b, c, d, e, f]
```

This code diagonalizes a conic using Sage's functionality. See 6.3 for an explanation of how this works.

```
def diagonalize(coeffs):
    '''Changes variables to obtain a conic of the form ax
    ^2 + by^2 + c.
    Args:
    f - a conic in the ring QQ['x, y'] of the form ax^2 +
    bxy + cy^2 + dx + ey + f
    '''
    a, b, c, d, e, f = coeffs
    Q = QuadraticForm(QQ, 3, [a/2,b/2,d/2,c/2,e/2,f/2])
    D, T = Q.rational_diagonal_form(return_matrix=True)

    a, b, c = D[0,0], D[1,1], D[2,2]

    return [a,b,c], T
```

This function multiplies a list of numbers by the lcm of their denominators.

```
def remove_denominators(coeffs):
    denominator = lcm([coeff.denominator() for coeff in
        coeffs])
    return [coeff * denominator for coeff in coeffs]
```

This function changes variables so that all the coefficients are square free, which is required by Hasse-Minkowski.

```
def remove_squares(coeffs):
    '''Removes the square component form each item in
    coeffs.
    coeffs - List of coefficients. Should be either [a, b
    , c] or [a, b].
    Returns the new coefficients and the transformation
    matrix.
    '''
    T = matrix.identity(QQ, 3)

    # Remove squares from each of the coefficients:
    for i in range(len(coeffs)):
        for p in prime_divisors(coeffs[i]):
            v = valuation(coeffs[i], p)
            if v > 1:
                if v % 2 == 0:
```

```

        coeffs[i] /= p**v
        T[i,i] *= p**(v/2)
    elif v % 2 == 1:
        coeffs[i] /= p**(v-1)
        T[i,i] *= p**((v-1)/2)
    return coeffs, T

```

This function performs the process to make $c = -1$. See 7.

```

def remove_c(coeffs):
    '''coeffs = [a, b, c] is a list of 3 nonzero
    squarefree integers. They are not all positive or
    all negative.'''
    a, b, c = coeffs

    if [x < 0 for x in coeffs].count(True) == 2:
        a, b, c = -a, -b, -c
        # We do not need to track this transformation
        because it does not affect the solution set

    if c < 0:
        T = matrix.identity(QQ, 3) # We already have a,
        b > 0, c < 0
    elif a < 0:
        a, c = c, a
        T = Matrix(QQ, [[0,0,1], [0,1,0], [1,0,0]])
    elif b < 0:
        b, c = c, b
        T = Matrix(QQ, [[1,0,0], [0,0,1], [0,1,0]])
    else:
        raise ValueError('a, b, and c cannot be all
        positive or all negative.')
    if c != -1:
        a *= -c
        b *= -c
        T = Matrix(QQ, [[1,0,0], [0,1,0], [0,0,c]]) * T

    return [a, b], T

```

This code performs the Hasse-Minkowski Algorithm 7.

```

def hasse_minkowski(a, b):
    [a, b], T = remove_squares([a, b])

```

```

if abs(a) > abs(b):
    a, b = b, a
    T = Matrix([[0,1,0],[1,0,0],[0,0,1]]) * T

# Terminating Conditions:
if a == 1:
    return T.inverse() * vector(QQ, [1,0,1])
elif b == 1:
    return T.inverse() * vector(QQ, [0,1,1])
elif abs(a) + abs(b) <= 2:
    return None

M = IntegerModRing(b)
if not is_square(M(a)):
    return None
t = sqrt_mod_n(a, b)
if t > abs(b)/2:
    t -= b
assert -abs(b)/2 <= t <= abs(b)/2

new_b = (t*t - a) / b

solution = hasse_minkowski(a, new_b)
if solution is None:
    return None
[p,q,r] = solution
return T.inverse() * vector(QQ, [-t*p+r, new_b*q, -a*
    p+t*r])

```

This function is a wrapper for the Hasse-Minkowski Algorithm which performs all of the necessary simplifications and then reverses them to get a point on the input conic.

```

def get_initial_point(coeffs):
    coeffs, T_diag = diagonalize(coeffs)
    a, b, c = remove_denominators(coeffs)

    if a == 0:
        return T.inverse() * vector(QQ, [1, 0, 0])
    elif b == 0:
        return T.inverse() * vector(QQ, [0, 1, 0])
    elif c == 0:
        return T.inverse() * vector(QQ, [0, 0, 1])
    elif all([x > 0 for x in [a, b, c]]) or all([x < 0
        for x in [a, b, c]]):

```

```

    return None

[a, b, c], T = remove_squares([a, b, c])

[a, b], T_ = remove_c([a, b, c])
T = T_ * T

[a, b], T_ = remove_squares([a, b]) # We may need to
    square-free-ify again as we have changed a and b
T = T_ * T

solution = hasse_minkowski(a, b)
if solution is None:
    return None

return list(T_diag * T.inverse() * solution)

```

This function is used to get an integer matrix with integer matrix inverse which sends the point (p, q, r) to $(1, 0, 0)$. We need this for computing parametrizations when our initial point is a point at infinity. Having integer coefficients in both the matrix and its inverse leads to nicer parametrizations.

In order to see that the inverse also has integer coefficients, notice that the determinant of this matrix is 1. Since the inverse of a matrix A is $\frac{1}{\det(A)}\text{adj}(A)$, and the adjoint has integer coefficients since A did, the inverse must have integer coefficients.

```

def integer_inverse_matrix(p, q, r):
    """
    Computes a matrix A which sends (p:q:r) to (1:0:0)
    and has integer coefficients,
    and whose inverse also has integer coefficients.
    The integer coefficients ensure that we get a nice
    parametrization.
    """
    g, u_pq, v_pq = extended_euclidean(p, q)
    d, u_rg, v_rg = extended_euclidean(r, g)
    if d != 1:
        raise ValueError('p, q and r should be coprime')
    if g == 0:
        assert(p == 0 and q == 0 and r == 1)
        A = Matrix(QQ, [[0, 1, 0],
                        [0, 0, 1],
                        [1, 0, 0]])

```



```

else:
    A = Matrix(QQ, [[p, -v_pq, -u_rg*p/g],
                    [q,  u_pq, -u_rg*q/g],
                    [r,  0,    v_rg]])
return A

```

This function gives a parametrization of all the rational points on a given conic.

```

def full_solve(C):
    if not C in R:
        try:
            C = R(C)
        except:
            raise ValueError('C should be a conic in QQ[x
                               ,y], not '+str(C))

    coeffs = coefficients(C)
    coeffs = remove_denominators(coeffs)
    a, b, c, d, e, f = coeffs

    if test_degenerate(coeffs):
        return None

    solution = get_initial_point(coeffs)
    if solution is None:
        return None

    [p,q,r] = solution

    # We want p, q, r to be coprime integers to get a
    nice parametrization.
    # We can multiply by anything nonzero to get another
    solution since the conic is homogenous.
    [p,q,r] = remove_denominators([p,q,r])
    divisor = gcd((p, q, r))
    p, q, r = p/divisor, q/divisor, r/divisor

    if r != 0:
        # This formula comes from the line-intersection
        property.
        # There is a slight speed-up by using this
        instead of the next one.
        # However, it does not work if the initial point
        is at infinity.

```

```

x0 = -b*q*n**2 + c*p*m**2 - 2*q*c*m*n - d*r*n**2
    - e*r*m*n - a*p*n**2
y0 = -c*m**2*q - d*m*n*r - e*m**2*r - 2*a*m*n*p -
    b*m**2*p + a*n**2*q
z0 = a*n**2*r + b*m*n*r + c*m**2*r

else:
    # Transform so that (p, q, r) = (1, 0, 0):
    A = integer_inverse_matrix(p, q, r)
    a1, b1, c1, d1, e1, f1 = transform_conic(coeffs,
        A.inverse())
    C1 = R(a1*x**2 + b1*x*y + c1*y**2 + d1*x + e1*y +
        f1)

    # Use the line-intersection property in this
    transformed place:
    x1 = c1*n**2 - e1*m*n + f1*m**2
    y1 = -b1*n**2 + d1*m*n
    z1 = b1*m*n - d1*m**2
    solution = vector(QQ[m,n], [x1, y1, z1])

    final_solution = A * solution # Transform back
    x0, y0, z0 = final_solution[0], final_solution
        [1], final_solution[2]

    #x0, y0, z0 = clean_parametrization(x0, y0, z0)

    return x0, y0, z0

```

This code checks if a given rational point is on a conic. It is used for testing and demonstrations.

```

def check(f, solution):
    if solution is None:
        print('f(x,y) = ' + str(f) + ' returned No Solutions\
            n')
        return

    elif len(solution) == 2:
        v = f.subs(x=solution[0], y=solution[1])
        if v == 0:
            print('f(x,y) = ' + str(f) + ' on ' + str(solution)
                + ' was 0\n')

```

```

else:
    print('f(x,y) = '+str(f)+' on '+str(solution)
          +' was '+str(v)+' as opposed to 0\n')

elif len(solution) == 3:
    g = f.homogenize('z')
    v = g.subs(x=solution[0], y=solution[1], z=
              solution[2])
    if v == 0:
        print('f(x,y) = '+str(f)+' on '+str(solution)
              +' was 0\n')
    else:
        print('f(x,y) = '+str(f)+' on '+str(solution)
              +' was '+str(v)+' as opposed to 0\n')

```

This function is used to demonstrate the code which finds rational points on conics.

```

def demo_solve(f):
    '''This is used to deonstrate the code to solve
    conics.'''
    print('f(x,y) = '+str(f))
    coeffs = coefficients(f)
    coeffs = remove_denominators(coeffs)

    if test_degenerate(coeffs):
        print('Conic is degenerate')
        return

    solution = get_initial_point(coeffs)
    if solution is None:
        print('No solutions')
        return

    [p,q,r] = solution
    print('Initial point:', (p, q, r), '\n')

    x0, y0, z0 = full_solve(f)
    print('Parametrization:', (x0/z0, y0/z0), '\n')
    for m0, n0 in [(1, 0), (1, 2)]:
        try:
            sol = ((x0/z0).subs(m=m0, n=n0), (y0/z0).subs
                  (m=m0, n=n0))
            print('Plugging in m='+str(m0)+' , n='+str(n0)
                  +':', sol)

```

```

        check(f, tuple(sol))
    except:
        sol = (x0.subs(m=m0, n=n0), y0.subs(m=m0, n=
            n0), z0.subs(m=m0, n=n0))
        print('Plugging in m='+str(m0)+' , n='+str(n0)
            , 'gives a point at infinity:', sol)
        check(f, tuple(sol))

```

This function was used to test the code on a variety of different conics.

```

def test_solve():
    print('Running tests:\n')
    functions = [R(x**2 + y**2 - 1), # Pythagorean
        triples
        R(2*x**2 + 4*y**2 - 1),
        R(x**2 + 1/2*y**2 - 1/2),
        R(29*x**2 + 11*y**2 - 1),
        R(x**2 + y**2 - 3), # Circle of Radius
        sqrt(3)
        R(17*x**2 + 8*y**2 - 1),
        R(x**2 + 2*x*y + y**2 - x + 1),
        R(x**2 + 2*x*y + y**2 - 32*x + 16),
        R(x**2 + x*y + y**2 - 32*x + 16),
        R(y**2 + 2*x),
        R(x**2 + 2*x*y + 4*y**2), # Degenerate
        Conic
        R(x**2 + y**2), # Degenerate Conic
        R(83*x**2 + 32*x*y - 2*y**2 + 1/2*x +
            25*y + 74)]
    for f in functions:
        demo_solve(f)
    print('\n', end='')

```

This code maps a to the vector space $\mathbb{Q}_p^\times/\mathbb{Q}_p^{\times 2}$. See 3.11.

```

def to_square_quotient(a, p):
    a = ZZ(a*a.denom()**2)
    if a == 0:
        raise ValueError('a cannot be 0')

    if p == infinity:
        return sign(a)

    elif not is_prime(p):

```

```

    raise ValueError('p must be prime or infinity')

elif p != 2:
    v = valuation(a, p)
    u = a / p**v
    return vector(ZZ, [(-1)**v, kronecker(u, p)])

# p = 2
v2 = valuation(a, 2)
u = IntegerModRing(8)(a / 2**v2)
v3, v5 = 0, 0
if u == 3 or u == 7:
    v3 = 1
if u == 5 or u == 7:
    v5 = 1
return vector(ZZ, [(-1)**v2, (-1)**v3, (-1)**v5])

```

This function changes from the vector space $\mathbb{Q}_p^\times/\mathbb{Q}_p^{\times 2}$ over $\{1, -1\}$ to the same but over $\mathbb{Z}/2\mathbb{Z}$.

```

def to_mod_2_vector_space(xs):
    return vector(ZZ, [0 if x == 1 else 1 for x in xs])

```

This function computes the Hilbert symbol $(a, b)_p$. See 8.3 and 8.4.

```

def hilbert(a, b, p):
    a = to_square_quotient(a, p)
    b = to_square_quotient(b, p)

    if p == infinity:
        [a, b] = to_mod_2_vector_space([a, b])
        return ZZ((-1)**(a*b))
    else:
        a = to_mod_2_vector_space(a)
        b = to_mod_2_vector_space(b)

    if p % 4 == 1:
        M = Matrix(ZZ, [[0, 1], [1, 0]])
        return ZZ((-1)**(a*M*b))
    elif p % 4 == 3:
        M = Matrix(ZZ, [[1, 1], [1, 0]])
        return ZZ((-1)**(a*M*b))

# p = 2

```

```
M = Matrix(ZZ, [[0, 1, 1], [1, 1, 0], [1, 0, 0]])
return ZZ((-1)**(a*M*b))
```

This function computes the discriminant d_p for p prime (see 9.4), and the inertia for $p = \infty$ (see 9.3).

```
def d(a, b, c, p):
    if a*b*c == 0:
        raise ValueError('The form must be nondegenerate')
    if p == infinity:
        return vector(ZZ, [
            [x > 0 for x in [a, b, c]].count(True),
            [x < 0 for x in [a, b, c]].count(True)])
    elif not is_prime(p):
        raise ValueError('p must be prime')
    return to_square_quotient(a*b*c, p)
```

This function computes the invariant ε_p 9.5.

```
def epsilon(a, b, c, p):
    return hilbert(a, b, p) * hilbert(a, c, p) * hilbert(
        b, c, p)
```

This function is used to check if two quadratic forms are \mathbb{Q} -isomorphic to one another. See 8.

```
def is_isomorphic_as_forms(coeffs1, coeffs2):
    a1, b1, c1 = coeffs1
    a2, b2, c2 = coeffs2
    d1 = a1*b1*c1
    d2 = a2*b2*c2

    if d1 == 0 or d2 == 0:
        raise ValueError('The forms must be nondegenerate')

    if not d(a1, b1, c1, infinity) == d(a2, b2, c2,
        infinity):
        return False

    # For the forms to be isomorphic, we need d1 = d2 in
    # Q*/(Q*^2)
```

```

if not is_square(QQ(d1/d2)):
    return False

for p in prime_divisors(2*d1*d2) + [infinity]:
    if not epsilon(a1, b1, c1, p) == epsilon(a2, b2,
        c2, p):
        return False

return True

```

This function is used to check if two conics are \mathbb{Q} -isomorphic to one another. See 9.10.

```

def is_isomorphic_as_conics(coeffs1, coeffs2):
    a1, b1, c1 = coeffs1
    a2, b2, c2 = coeffs2

    d1 = a1*b1*c1
    d2 = a2*b2*c2

    if d1 == 0 or d2 == 0:
        raise ValueError('The conics must be
            nondegenerate')

    return is_isomorphic_as_forms(d2*vector(QQ, coeffs1),
        d1*vector(QQ, coeffs2))

```

This function finds all the places for which the conic is not locally solvable. See 8 for an explanation.

```

def places_not_locally_solvable(a, b, c):
    '''Returns the places p at which the conic a*x**2 + b
        *y**2 + c*z**2 = 0 has no solutions'''
    failure_points = []

    if hilbert(-a*c, -b*c, infinity) == -1:
        failure_points.append(infinity)

    for p in prime_divisors(2*a*b*c):
        if hilbert(-a*c, -b*c, p) == -1:
            failure_points.append(p)

    return failure_points

```

This is a wrapper for the previous function which takes polynomials instead of coefficients of a diagonal matrix.

```
def places_conic_not_locally_solvable(f):
    coeffs = coefficients(f)
    coeffs, _ = diagonalize(coeffs)
    a, b, c = remove_denominators(coeffs)
    return places_not_locally_solvable(a, b, c)
```

This function is used to demonstrate the code that finds the places p for which a conic is not locally solvable over \mathbb{Q}_p .

```
def demo_solvable(f):
    '''This function can be called to demonstrate the
    code for local solvability over Q_p.'''
    places = places_conic_not_locally_solvable(f)
    if places == []:
        print('f(x, y) = '+str(f)+' is solvable over Q_p
              for all p (including Q_infinity = R), and
              therefore it is solvable over Q.')
    else:
        print('f(x, y) = '+str(f)+' is not locally
              solvable over Q_p for p = '+str(tuple(places))
              )
```

This function is used to demonstrate the code that tests whether two conics/forms are isomorphic.

```
def demo_isomorphic(f1, f2):
    '''This is used to demonstrate the code that tests if
    two conics/forms are isomorphic.'''
    coeffs1 = coefficients(f1)
    coeffs1, _ = diagonalize(coeffs1)
    coeffs1 = remove_denominators(coeffs1)
    coeffs2 = coefficients(f2)
    coeffs2, _ = diagonalize(coeffs2)
    coeffs2 = remove_denominators(coeffs2)

    if is_isomorphic_as_conics(coeffs1, coeffs2):
        if is_isomorphic_as_forms(coeffs1, coeffs2):
            print('f_1(x, y) = '+str(f1)+' and '+str(f2)+' are isomorphic as
                  quadratic forms and therefore also as
                  conics.')
```



```

else:
    print('f_1(x, y) = '+str(f1)+' and '+'f_2(x,
        y) = '+str(f2)+' are isomorphic as conics
        but not as quadratic forms.')
else:
    print('f_1(x, y) = '+str(f1)+' and '+'f_2(x, y) =
        '+str(f2)+' are not isomorphic as quadratic
        forms or even as conics.')

```

REFERENCES

1. **SageMath**, the Sage Mathematics Software System (Version 9.5), The Sage Developers, 2022, <https://www.sagemath.org/>
2. Neal Koblitz: ***p-adic Numbers, p-adic Analysis, and Zeta Functions*** Grad. Texts in Math., 58 Springer-Verlag, New York, 1984, ISBN: 0-387-96017-1
3. Henri Cohen: ***A Course in Computational Algebraic Number Theory*** Grad. Texts in Math., 138 Springer-Verlag, Berlin, 1993 ISBN:3-540-55640-0
4. Kenneth Ireland, Michael Rosen: ***A Classical Introduction to Modern Number Theory*** Grad. Texts in Math., 84 Springer-Verlag, New York, 1990, ISBN: 0-387-97329-X
5. Jean-Pierre Serre: ***A Course In Arithmetic*** Grad. Texts in Math., No. 7 Springer-Verlag, New York-Heidelberg, 1973
6. Shou-Wu Zhang: ***Rational Points on Curves*** (Online Lecture Notes)
<https://web.math.princeton.edu/~shouwu/teaching/RatP.pdf>
7. Nicolas Mascot: ***Notes on Number Theory 2021*** (Online Lecture Notes)
<https://www.maths.tcd.ie/~mascotn/teaching/2020/MAU23101/6%20Continued%20Fractions.pdf>