This is an updated version of the final accepted thesis. The main changes are a correction in Section 4, an update to the Magma code in the Appendix, and the inclusion of hypertext links in the document to make navigation easier.

IMPLEMENTATION OF A THUE-MAHLER EQUATION SOLVER

by

KYLE DAVID HAMBROOK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

October 2011

Abstract

A practical algorithm for solving an arbitrary Thue-Mahler equation is presented, and its correctness is proved. Methods of algebraic number theory are used to reduce the problem of solving the Thue-Mahler equation to the problem of solving a finite collection of related Diophantine equations having parameters in an algebraic number field. Bounds on the solutions of these equations are computed by employing the theory of linear forms in logarithms of algebraic numbers. Computational Diophantine approximation techniques based on lattice basis reduction are used to reduce the upper bounds to the point where a direct enumerative search of the solution space becomes possible. Such an enumerative search is carried out with the aid of a sieving procedure to finally determine the complete set of solutions of the Thue-Mahler equation. The algorithm is implemented in full generality as a function in the Magma computer algebra system. This is the first time a completely general algorithm for solving Thue-Mahler equations has been implemented as a computer program.

Table of Contents

Acknowledgements

I am indebted to Dr. Michael A. Bennett for suggesting to me the line of research on which this thesis is based and for numerous comments and suggestions that helped me to improve this thesis. I also thank Dr. Dragos Ghioca for helping me to improve the style of this thesis.

This research was funded in part by a National Sciences and Engineering Research Council Postgraduate Scholarship and a Li Tze Fong Memorial Fellowship.

Dedication

To my mom.

1 Introduction

A Thue-Mahler equation is an equation of the form

$$F(x, y) = ap_1^{z_1} \cdots p_v^{z_v}, \tag{1}$$

where F is an irreducible binary form over $\mathbb{Z}$ of degree $n \geq 3$, a is a nonzero integer, $p_1, \ldots, p_v$ are rational primes, and the unknowns $x, y, z_1, \ldots, z_v$ are integers with $z_i \geq 0$ $(i = 1, \ldots, v)$. Thue-Mahler equations generalize Thue equations, which are simply equations of the form $F(x, y) = a$. When discusing the number of solutions of a Thue-Mahler equation, it is standard to consider only those solutions with $(x, y) = 1$; without this consideration, every Thue-Mahler equation with at least one solution would trivially have infinitely many solutions. Moreover, as we will see in Section 3, each solution of (1) with $(x, y) > 1$ is generated by a solution of one of finitely many other Thue-Mahler equations with $(x, y) = 1$.

In 1933, Mahler [Mah] proved that an arbitrary Thue-Mahler equation has only finitely many solutions with $(x, y) = 1$. His proof was ineffective in the sense that it provided no means to actually find the solutions. In the mid-1960s, Baker proved his ground-breaking results on effective lower bounds for linear forms in logarithms of algebraic numbers. By generalizing Baker's results to the p-adic case, it was proved by Sprindžuk and Vinogradov [SV] and by Coates [Coa1], [Coa2] that the solutions of any Thue-Mahler equation could, at least in principle, be determined effectively. The first general ideas on the practical computation of the solutions of an arbitrary Thue-Mahler equation are due to Tzanakis and de Weger (cf. [dW1], [dW2], [TW1], [TW2]). These ideas were inspired in part by the method that Agrawal, Coates, Hunt, and van der Poorten used in [ACHP] to solve a specific Thue-Mahler equation. In [TW2], Tzanakis and de Weger gave a practical procedure for finding all the solutions of an arbitrary Thue-Mahler equation.

Up until now, no general algorithm for solving an arbitrary Thue-Mahler equation had been implemented as a computer program. Anyone wishing to solve a Thue-Mahler equation would need to apply the method of [TW2] (or a more specialized method if the equation was of a special form) to the equation themselves. The numerous computational tasks involved in the method would, of course, be performed with the assistance of a computer, but the set-up for these tasks could represent a significant challenge to the would-be solver. The main novel contribution of this thesis is the implementation of a

modified version of Tzanakis and de Weger's procedure for solving an arbitrary Thue-Mahler equation as a computer program. Hence, to solve a given Thue-Mahler equation now, one only needs to enter the parameters of the equation into our program and wait for the list of solutions to be returned.

Following [TW2], we present in this thesis a practical algorithm for finding all the solutions of an arbitrary Thue-Mahler equation. The general outline of the algorithm consists of six steps.

Step 1. We reduce the problem of solving the given Thue-Mahler equation to the problem of solving a collection of finitely many Diophantine equations whose defining parameters belong to a certain algebraic number field K. The collection is such that if we know the solutions of each equation in the collection, then we can easily derive all of the solutions of the Thue-Mahler equation. So this is a reduction indeed. The principal computational work here consists of: computing some data for K (including an integral basis, a system of fundamental units, and a splitting field); computing the factorizations of the primes $p_1, ..., p_v$ into prime ideals in the ring of integers of K; finding the roots of the defining polynomial of K in $\mathbb{C}$ and in the algebraic closure of the p-adic field $\mathbb{Q}_p$ for $p \in \{p_1, ..., p_v\}$; applying a lemma of Tzanakis and de Weger to ensure that the collection of Diophantine equations that we will need to solve is small; and, finally, computing the defining parameters of these Diophantine equations.

We perform the next four steps for each of the Diophantine equations in our collection.

Step 2. We perform numerous p-adic computations to make optimal choices for certain parameters which arise and (whenever possible) to either determine exact values for or find small bounds on certain variables involved in the Diophantine equation under consideration. These computations, in general, ultimately reduce the amount of computation needed in subsequent steps. Specifically, we (potentially) reduce the number of linear forms that need to be considered considered in Steps 3 and 4, as well as reduce the number of terms in the linear forms that are considered. This results in smaller bounds and fewer computations in Step 3, in lower dimensional and fewer approximation lattices in Step 4, and in a smaller number of candidate solutions to test in Step 5. All of these features help to decrease the overall computation involved in Steps 3-5. The individual computations involved

in this step are not very difficult (the most involved being probably the evaluation of p-adic logarithms), but if there are many possibilities for the parameters, this step could take some time. However, it is very unlikely that the computational cost of this step is anywhere near that of the most costly steps in the algorithm.

Step 3. We use the theory of linear forms in real/complex and p-adic logarithms of algebraic numbers to derive very large bounds on those variables in the Diophantine equation that were not bounded in the previous step. For this purpose we use the best theorems available. In the real/complex case we appeal to the results of Matveev [Mat], and in the p-adic case we employ the results of Yu [Yu2]. The most difficult computations for this step involve factoring rational primes into ideals in the splitting field of K, constructing a specific subfield of the splitting field, and computing heights of specific elements of the splitting field.

Step 4. We drastically reduce the upper bounds derived in Step 3 by using computational Diophantine approximation techniques. These techniques involve applying LLL-lattice basis reduction methods to approximation lattices associated with the linear forms in logarithms considered in Step 3. The main computations here involve using the LLL algorithm to compute an LLL-reduced basis for a given lattice and using the Fincke-Pohst algorithm to enumerate all the short vectors in a given lattice.

Step 5. We use a sieving procedure to find all the solutions of the Diophantine equation that live in the box defined by the bounds derived in the previous three steps. The procedure essentially works by first running through all the possible solutions in the box and 'sieving' out the vast majority of non-solutions by checking congruence conditions that have relatively low computational cost to check, and then testing the small number of possible solutions that remain for the Diophantine equation directly. Though we expect the bounds defining the box to be small, there can still be a very large number of possible solutions to check (especially if the number of primes involved in the Thue-Mahler equation is large and/or the number of elements in a system of fundamental units for K is large). The computations performed on each individual candidate solution are relatively simple, but the sheer number of candidates often makes this step the computational bottleneck of the entire algorithm.

Step 6. Having performed Steps 2-5 for each Diophantine equation in our collection, we now know all the solutions of each such equation, and we use this knowledge to determine all the solutions of the Thue-Mahler equation. The individual computations involved in this step are practically trivial, and there should not be very many to do.

The above steps more or less accurately describe our algorithm for solving an arbitrary Thue-Mahler equation. However, we have omitted two important details in order to make the algorithm easier to understand upon first reading. The first omitted detail is that during Step 4 we will likely accumulate a number of explicitly known exceptional candidate solutions that don't fit in the final reduced box that we will run through in Step 5. Since we know them explicitly, these exceptional candidates don't hold us back from reducing the size of the box, but we must test them as solutions of the Thue-Mahler equation separately. The second omitted detail is that, instead of waiting to perform Step 6 until after Step 5 has been performed for every equation in the collection, we will essentially perform a portion of Step 6 at the end of each Step 5. More explicitly, as we are finding the solutions of each Diophantine equation, we will also find the corresponding solutions of the Thue-Mahler equation. We do this to improve efficiency.

We mentioned above that Step 5 could be the main bottleneck for the algorithm. This will be the case in general except possibly when the degree of K (which is the degree of $F(x, y)$) is large or when the degree of the splitting field of K is very large. When the degree of K is large, computing a system of fundamental units and/or performing certain other necessary computations in K can be extremely time consuming. In particular, the worst-case complexity of calculating a system of fundamental units is exponential in the degree of K. Performing the necessary computations in the splitting field of K can likewise be very difficult if the splitting field has large degree. It is difficult to say much about the precise computational complexity of the algorithm without delving into a very complicated analysis. However, since the sieving process will often be the computational bottleneck for the algorithm, we feel it is important to give a rough estimate of its running time. Let $E(k)$ denote the time required for modular exponentiation when the exponent is a k-bit integer and $M(k)$ denotes the time required to multiply two k-bit integers. It is reasonable to assume $E(k) = O(k)$ and $M(k) = O(k^2)$. The running time for our implementation of the sieve procedure is roughly $O\left((E(d^{-1}\log N) + M(n^{-1}N^{1/d}))dN\right)$ bit operations, where N is the total number of tuples that need to be searched through, n is the degree of K,

and d is the sum of the number v of rational primes in the Thue-Mahler equation and the number r of elements in a complete set of fundamental units for K. Note that N will depend exponentially on d.

The algorithm lends itself well to being implemented in parallel in a master/slave paradigm as follows. The master process performs Step 1 and then, for each Diophantine equation in the collection it builds, farms out Steps 2-5 to a slave. Once the slaves are done, each slave passes back the list of solutions for its Diophantine equation to the master process, and the master process performs Step 6. Alternatively, each slave could perform a part of Step 6 itself after finishing Step 5 as discussed above.

Our algorithm is very similar to that of Tzanakis and de Weger [TW2]. Indeed, the overall strategy is the same and every Lemma and Theorem that we present appears (at least implicitly) in [TW2]. However, our algorithm does differ from Tzanakis and de Weger's in a few minor ways that can potentially result in significant computational savings. One difference is that our algorithm puts more work into minimizing how often the most computationally expensive techniques are employed; this work is mostly done in Step 2 above. Another difference is that our algorithm does more work to optimize the bounds arising from the theory of linear forms in logarithms. We have endeavoured to present the algorithm essentially in the manner in which it is implemented. As a consequence, our exposition differs from [TW2] in both overall structure and in the increased level of practical detail provided in the various steps.

The implementation of our algorithm as a function in the Magma computer algebra system [BCP] can be found in the Appendix.

2 Decompositions of primes and polynomials; p-adic embeddings; p-adic valuations

In this section, we give a concise description of the relationships between how a rational prime p decomposes in the ring of integers of a number field K, how the defining polynomial of K decomposes over the field $\mathbb{Q}_p$ of p-adic numbers, how K embeds into the algebraic closure of $\mathbb{Q}_p$, and how the p-adic valuation on $\mathbb{Q}$ extends to a p-adic valuation on K. We also discuss how the p-adic valuation on $\mathbb{Q}_p$ extends to larger fields. As references for this material we give [BS] (especially Theorem 3 in Chapter 4, Section 2), [Ca] (especially Lemma 2.1 in Chapter 9), [Ha] (especially Chapter 18), [Ko] (especially Chapter 3, Section 2), and [Na] (especially Theorem 6.1).

Let p be a rational prime. Let $g(t)$ be an irreducible polynomial in $\mathbb{Q}[t]$ of degree n, and let $K = \mathbb{Q}(\theta)$, where $g(\theta) = 0$. There are one-to-one correspondences between each of the following four sets of objects:

- The prime ideals in (the ring of integers of) K that divide p.
- The irreducible polynomial factors of $g(t)$ in $\mathbb{Q}_p[t]$.
- The classes of conjugate embeddings of K into the algebraic closure $\overline{\mathbb{Q}_p}$ of $\mathbb{Q}_p$.
- The extensions of the p-adic valuation ord_p on $\mathbb{Q}$ to K.

In what follows, we will describe the important features of these correspondences. Note that two embeddings of K into $\overline{\mathbb{Q}_p}$ are called conjugate if they map θ to roots of the same irreducible polynomial in $\mathbb{Q}_p[t]$. Note also that what we call a p-adic valuation is sometimes called a p-adic order.

Let

$$(p) = \mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_m^{e_m}$$

be the decomposition of (p) into prime ideals in $\mathcal{O}_K$ (the ring of integers of K), and let f_i be the residue degree of $\mathfrak{p}_i$ over p. For $x \in K$, let $\mathrm{ord}_{\mathfrak{p}_i}(x)$ denote the $\mathfrak{p}_i$-adic valuation of x, i.e., the exponent of $\mathfrak{p}_i$ in the prime ideal factorization of the (fractional) ideal (x). Let $K_{\mathfrak{p}_i}$ denote the completion of K with respect to $\mathrm{ord}_{\mathfrak{p}_i}$. Let

$$g(t) = g_1(t) \cdots g_m(t)$$

be the decomposition of $g(t)$ into irreducible polynomials in $\mathbb{Q}_p[t]$. For each $i \in \{1, \ldots, m\}$, let $n_i = \deg g_i(t)$. The correspondence $\mathfrak{p}_i \longleftrightarrow g_i(t)$ is such that $n_i = e_i f_i$ and $K_{\mathfrak{p}_i} \simeq \mathbb{Q}_p(\theta_i)$,

where $g_i(\theta_i) = 0$.

There are n embeddings of K into $\overline{\mathbb{Q}_p}$, and each one fixes $\mathbb{Q}$ and maps θ to a root of g in $\overline{\mathbb{Q}_p}$. Let $\theta_i^{(1)}, \ldots, \theta_i^{(n_i)}$ denote the roots of $g_i(t)$ in $\overline{\mathbb{Q}_p}$. For $i = 1, \ldots, m$ and $j = 1, \ldots, n_i$, let σ_{ij} be the embedding of K into $\mathbb{Q}_p(\theta_i^{(j)})$ defined by $\theta \mapsto \theta_i^{(j)}$. The m classes of conjugate embeddings are $\{\sigma_{i1}, \ldots, \sigma_{in_i}\}$ $(i = 1, \ldots, m)$. Note that σ_{ij} coincides with the embedding $K \hookrightarrow K_{\mathfrak{p}_i} \simeq \mathbb{Q}_p(\theta_i) \simeq \mathbb{Q}_p(\theta_i^{(j)})$.

For any finite extension L of $\mathbb{Q}_p$, the p-adic valuation on $\mathbb{Q}_p$ extends uniquely to L as

$$\mathrm{ord}_p(x) = \frac{1}{[L : \mathbb{Q}_p]} \mathrm{ord}_p \left(N_{L/\mathbb{Q}_p}(x) \right). \tag{2}$$

This definition is independent of the field L containing x. So, since each element of $\overline{\mathbb{Q}_p}$ is by definition contained in some finite extension of $\mathbb{Q}_p$, (2) can be used to define the p-adic valuation of any $x \in \overline{\mathbb{Q}_p}$. Every finite extension of $\mathbb{Q}_p$ is complete with respect to ord_p, but $\overline{\mathbb{Q}_p}$ is not. The completion of $\overline{\mathbb{Q}_p}$ with respect to ord_p is denoted by $\mathbb{C}_p$. Note that the formulas

$$\mathrm{ord}_p(xy) = \mathrm{ord}_p(x) + \mathrm{ord}_p(y), \qquad \mathrm{ord}_p(x + y) \geq \min \{\mathrm{ord}_p(x), \mathrm{ord}_p(y)\}$$

still hold when $x, y \in \mathbb{C}_p$. It is convenient to record here that an element $x \in \mathbb{C}_p$ having $\mathrm{ord}_p(x) = 0$ is called a p-adic unit.

The m extensions of the p-adic valuation on $\mathbb{Q}$ to K are just divisors of the $\mathfrak{p}_i$-adic valuations on K:

$$\mathrm{ord}_p(x) = \frac{1}{e_i} \mathrm{ord}_{\mathfrak{p}_i}(x) \qquad (i = 1, \ldots m).$$

We can also view these extensions as arising from the various embeddings of K into $\overline{\mathbb{Q}_p}$. Indeed, the extension to $\mathbb{Q}_p(\theta_i^{(j)})$ of the p-adic valuation on $\mathbb{Q}_p$ induces a p-adic valuation on K via the embedding σ_{ij} as

$$\mathrm{ord}_p(x) = \mathrm{ord}_p(\sigma_{ij}(x)),$$

and we have

$$\mathrm{ord}_p(\sigma_{ij}(x)) = \frac{1}{e_i} \mathrm{ord}_{\mathfrak{p}_i}(x) \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n_i). \tag{3}$$

3 First steps

Fix a nonzero integer a and a set of distinct rational primes $p_1, \dots, p_v$. Let

$$F(x,y) = c_0 x^n + c_1 x^{n-1} y + \cdots + c_{n-1} x y^{n-1} + c_n y^n$$

be an irreducible binary form over $\mathbb{Z}$ of degree $n \geq 3$. We want to solve the Thue-Mahler equation

$$F(x,y) = a p_1^{z_1} \cdots p_v^{z_v} \tag{4}$$

for unknown integers $x, y, z_1, \dots, z_v$ with $z_i \geq 0$ for $i = 1, \dots, v$. We will first discuss how to reduce this general problem to the case where

$$c_0 = 1, \quad (a, p_1 \cdots p_v) = 1, \quad (x,y) = 1.$$

Observe that a tuple $(x, y, z_1, \dots, z_v)$ is a solution of (4) if and only if the tuple $(x, y, z_1', \dots, z_v')$ with $z_i' = z_i + \mathrm{ord}_{p_i}(a)$ for $i = 1, \dots, v$ is a solution of

$$F(x,y) = a' p_1^{z_1'} \cdots p_v^{z_v'},$$

where $a = a' \prod_{i=1}^v p_i^{\mathrm{ord}_{p_i}(a)}$. Since $(a', p_1 \cdots p_v) = 1$, this observations explains how to reduce to the case where $(a, p_1 \cdots p_v) = 1$.

Next we see how to reduce to the case of finding solutions with $(x,y) = 1$. We assume that we have already reduced to a situation where $(a', p_1 \cdots p_v) = 1$. Let $q_1, \dots, q_w$ denote the distinct prime divisors of a. First, observe that if $(x, y, z_1, \dots, z_v)$ is a solution of (4), then $(x,y)^n$ divides $a p_1^{z_1} \cdots p_v^{z_v}$, and so

$$(x,y) = \prod_{i=1}^v p_i^{a_i} \prod_{i=1}^w q_i^{b_i}$$

for some nonnegative integers a_i, b_i satisfying $n a_i \leq z_i$ and $n b_i \leq \mathrm{ord}_{q_i}(a)$. Let $a_1, \dots, a_v$, $b_1, \dots, b_w$ be arbitrary nonnegative integers such that $n b_i \leq \mathrm{ord}_{q_i}(a)$ for $i = 1, \dots, w$. Define $a' = a / \prod_{i=1}^w q_i^{n b_i}$, $d = \prod_{i=1}^v p_i^{a_i} \prod_{i=1}^w q_i^{b_i}$, and the change of variables $x' = x/d$, $y' = y/d$, $z_i' = z_i - n a_i$ for $i = 1, \dots, v$. Then the tuple $(x, y, z_1, \dots, z_v)$ is a solution of (4) satisfying $(x,y) = d$ if and only if the tuple $(x', y', z_1', \dots, z_v')$ is a solution of

$$F(x', y') = a' p_1^{z_1'} \cdots p_v^{z_v'}, \tag{5}$$

satisfying $(x', y') = 1$. So to solve (4) it suffices to follow the following procedure. First, for each of the finitely many possible values for $(b_1, \ldots, b_w)$, we find the finitely many solutions $(x', y', z_1', \ldots, z_v')$ of (5) satisfying $(x', y') = 1$. Then the solutions $(x, y, z_1, \ldots, z_v)$ of (4) are given by

$$\begin{aligned} x &= x' \prod_{i=1}^{v} p_i^{a_i} \prod_{i=1}^{w} q_i^{b_i}, \\ y &= y' \prod_{i=1}^{v} p_i^{a_i} \prod_{i=1}^{w}, q_i^{b_i} \\ z_i &= z_i' + na_i \quad (i = 1, \ldots, v), \end{aligned} \tag{6}$$

for each pair $(b_1, \ldots, b_w)$, $(x', y', z_1', \ldots, z_v')$ determined in the first step. In (6), each a_i runs through the complete set of nonnegative integers.

Now we will describe how to reduce to the case where $c_0 = 1$. Since $F(x, y)$ is irreducible, at least one of c_0 and c_n is nonzero. Hence, we can always transform the given Thue-Mahler equation to one with $c_0 \neq 0$ by interchanging x and y and renaming the coefficients c_i appropriately ($c_i \leftarrow c_{n-i}$). In what follows, we assume that we have reduced to a situation where $c_0 \neq 0$ and $(a, p_1 \cdots p_v) = 1$. Let $q_1, \ldots, q_w$ denote the distinct prime divisors of a. Note that if $(x, y, z_1, \ldots, z_v)$ is a solution of (4), then (c_0, y) divides $ap_1^{z_v} \ldots p_v^{z_v}$, and so

$$(c_0, y) = \prod_{i=1}^{v} p_i^{a_i} \prod_{i=1}^{w} q_i^{b_i}$$

for some nonnegative integers $a_1, \ldots, a_v, b_1, \ldots, b_w$ such that $a_i \leq \min\{z_i, \mathrm{ord}_{p_i}(c_0)\}$ and $b_i \leq \min\{\mathrm{ord}_{q_i}(a), \mathrm{ord}_{q_i}(c_0)\}$. Let $a_1, \ldots, a_v, b_1, \ldots, b_w$ be arbitrary nonnegative integers such that $a_i \leq \mathrm{ord}_{p_i}(c_0)$ and $b_i \leq \min\{\mathrm{ord}_{q_i}(a), \mathrm{ord}_{q_i}(c_0)\}$. Note that, since c_0 is fixed, there are only finitely many possibilities for the a_i and b_i. Define $d = \prod_{i=1}^{v} p_i^{a_i} \prod_{i=1}^{w} q_i^{b_i}$, $a' = ac_0^{n-1}/(a, d)d^{n-1}$, and $C_i = c_i c_0^{i-1}$ for $i = 1, \ldots, n$. If $(x, y, z_1, \ldots, z_v)$ is a solution of (4) with $(x, y) = 1$ and $(c_0, y) = d$, then it is easy to verify that $(X, Y, z_1', .., z_v')$, where $X = c_0 x/d$, $Y = y/d$, and $z_i' = z_i - \mathrm{ord}_{p_i}(d) \geq 0$, is a solution of

$$X^n + C_1 X^{n-1} Y + \cdots + C_{n-1} X Y^{n-1} + C_n Y^n = a' p_1^{z_1'} \cdots p_v^{z_v'} \tag{7}$$

with $(X, Y) = 1$. So to find all the solutions $(x, y, z_1, \ldots, z_v)$ of (4) with $(x, y) = 1$, it suffices to find, for each of the finitely many possible values of $(a_1, \ldots, a_v, b_1, \ldots, b_w)$, all the solutions $(X, Y, z_1', .., z_v')$ of (7) with $(X, Y) = 1$, and use the change of variables forumula $X = c_0 x/d$, $Y = y/d$, $z_i' = z_i - \mathrm{ord}_{p_i}(d) \geq 0$. Since the coefficient of X^n in (7) is 1, the above explains how to reduce to the case where $c_0 = 1$.

Now that we have shown that it is always possible to reduce our general problem to solving (4) in the case

$$c_0 = 1, \quad (a, p_1 \cdots p_v) = 1, \quad (x, y) = 1,$$

we will assume these conditions from now on.

Put $g(t) = F(t, 1) = t^n + c_1 t^{n-1} + \cdots + c_{n-1} t + c_n$ and note that $g(t)$ is irreducible in $\mathbb{Z}[t]$. Let $K = \mathbb{Q}(\theta)$ with $g(\theta) = 0$. Then (4) is equivalent to the norm equation

$$N_{K/\mathbb{Q}}(x - y\theta) = a p_1^{z_1} \cdots p_v^{z_v}. \tag{8}$$

Let

$$(p_i) = \prod_{j=1}^{m_i} \mathfrak{p}_{ij}^{e_{ij}}$$

be the factorization of p_i into prime ideals in the ring of integers $\mathcal{O}_K$ of K, and let f_{ij} be the residue degree of $\mathfrak{p}_{ij}$ over p_i. Then, since $N(\mathfrak{p}_{ij}) = p_i^{f_{ij}}$, (8) implies finitely many ideal equations of the form

$$(x - y\theta) = \mathfrak{a} \prod_{j=1}^{m_1} \mathfrak{p}_{1j}^{z_{1j}} \cdots \prod_{j=1}^{m_v} \mathfrak{p}_{vj}^{z_{vj}}, \tag{9}$$

where $\mathfrak{a}$ is an ideal of norm $|a|$ and the z_{ij} are unknown nonnegative integers related to the z_i by $\sum_{j=1}^{m_i} f_{ij} z_{ij} = z_i$.

Our first task will be to cut down the number of variables appearing in (9). We will do this by showing that only a few prime ideals can divide $(x - y\theta)$ to a large power.

4 The prime ideal removing lemma

In this section, we establish the key lemma that will allow us to cut down the number of prime ideals that can appear to a large power in the factorization of $(x - y\theta)$.

Let p be a rational prime and keep the same notation as in Section 2.

Lemma 4.1. (The Prime Ideal Removing Lemma).

(a) For every pair $i \neq j \in \{1, \dots, m\}$, if

$$\operatorname{ord}_{\mathfrak{p}_i}(x - y\theta) > e_i \min_{k,l} \operatorname{ord}_p\big(\theta_i^{(k)} - \theta_j^{(l)}\big), \tag{10}$$

then

$$\operatorname{ord}_{\mathfrak{p}_j}(x - y\theta) \leq e_j \min_{k,l} \operatorname{ord}_p\big(\theta_i^{(k)} - \theta_j^{(l)}\big). \tag{11}$$

(b) For every $i \in \{1, \dots, m\}$,

$$\operatorname{ord}_{\mathfrak{p}_i}(x - y\theta) \leq e_i \min_{k,l} \operatorname{ord}_p\big(\theta_i^{(k)} - \theta_i^{(l)}\big). \tag{12}$$

Note that the right-hand side of (12) is finite if and only if $e_i f_i > 1$.

Proof. Let $i, j \in \{1, \dots, m\}$. Put $v_i = \operatorname{ord}_{\mathfrak{p}_i}(x - y\theta)$, $v_j = \operatorname{ord}_{\mathfrak{p}_j}(x - y\theta)$. It suffices to show

$$\min\left\{\frac{v_i}{e_i}, \frac{v_j}{e_j}\right\} \leq \operatorname{ord}_p(\theta_i^{(k)} - \theta_j^{(l)})$$

for every $k \in \{1, \dots, n_i\}$, $l \in \{1, \dots, n_j\}$. In view of (3), we have $\dfrac{v_i}{e_i} = \operatorname{ord}_p\big(x - y\theta_i^{(k)}\big)$ and $\dfrac{v_j}{e_j} = \operatorname{ord}_p\big(x - y\theta_j^{(l)}\big)$. Hence,

$$\min\left\{\frac{v_i}{e_i}, \frac{v_j}{e_j}\right\} \leq \operatorname{ord}_p\big(\big(x - y\theta_i^{(k)}\big) - \big(x - y\theta_j^{(l)}\big)\big) = \operatorname{ord}_p\big(\theta_i^{(k)} - \theta_j^{(l)}\big) + \operatorname{ord}_p(y).$$

If $p \nmid y$, we are done. In case p divides y, each prime ideal above p also divides y. But a prime ideal dividing y cannot divide $(x - y\theta)$ because $(x, y) = 1$. So if p divides y, the left-hand sides of (10)-(12) are zero. □

Corollary 4.2. There is at most one $\mathfrak{p}_i$ dividing p with

$$\operatorname{ord}_{\mathfrak{p}_i}(x - y\theta) > \frac{1}{2}\, e_i \operatorname{ord}_p(\operatorname{Discr} g(t)), \tag{13}$$

and it satisfies $e_i = f_i = 1$ (if it exists).

Proof. Let $\theta^{(1)}, \dots, \theta^{(n)}$ denote the roots of g in $\overline{\mathbb{Q}_p}$. Since

$$\operatorname{Discr} g(t) = \prod_{1 \le h < h' \le n} (\theta^{(h)} - \theta^{(h')})^2,$$

and since θ is an algebraic integer, we have

$$\operatorname{ord}_p(\operatorname{Discr} g(t)) = \sum_{1 \le h < h' \le n} 2\operatorname{ord}_p\big(\theta^{(h)} - \theta^{(h')}\big) \ge 2\operatorname{ord}_p\big(\theta^{(h)} - \theta^{(h')}\big)$$

for any two distinct roots $\theta^{(h)}, \theta^{(h')}$. So if $i \ne j \in \{1, \dots, m\}$, we have $\operatorname{ord}_p(\operatorname{Discr} g(t)) \ge 2\operatorname{ord}_p\big(\theta_i^{(k)} - \theta_j^{(l)}\big)$ for all k, l. It follows from this fact and Lemma 4.1(a) that if (13) holds for some $\mathfrak{p}_i$, then it fails to hold for every $\mathfrak{p}_j$ with $j \ne i$. If a $\mathfrak{p}_i$ has either $e_i > 1$ or $f_i > 1$, then there are (at least) two distinct roots $\theta_i^{(k)}$, $\theta_i^{(l)}$, and we have $\operatorname{ord}_p(\operatorname{Discr} g(t)) \ge 2\operatorname{ord}_p\big(\theta_i^{(k)} - \theta_i^{(l)}\big)$. It follows from Lemma 4.1(b) that (13) cannot hold for such a $\mathfrak{p}_i$. □

In light of the results above, we see that for each solution $(x, y, z_1, \dots, z_v)$ of (4) and each $p \in \{p_1, \dots, p_v\}$, there can be at most one prime ideal $\mathfrak{p}_k$ above p whose exponent in the factorization of $(x - y\theta)$ need not be bounded by $\frac{1}{2} \max\{e_1, \dots, e_m\} \operatorname{ord}_p(\operatorname{Discr} g(t))$. We call such a $\mathfrak{p}_k$ the unconstrained prime above p. We know that the unconstrained prime must have $e_k = f_k = 1$. For each choice of unconstrained prime $\mathfrak{p}_k$ above p, there are only finitely many possibilities for the exponents of the other primes above p in the factorization of $(x - y\theta)$. If there is no prime ideal $\mathfrak{p}_k$ above p with $e_k = f_k = 1$, there can be no unconstrained prime above p, and there are only finitely many possibilities for the exponent of each prime above p in the factorization of $(x - y\theta)$.

Now we describe an algorithm that, for a given $p \in \{p_1, \dots, p_v\}$, computes for each choice of unconstrained prime above p the set of all possible combinations of exponents that the other primes above p can have in the factorization of $(x - y\theta)$. If there can be no unconstrained prime above p (i.e., if there is no prime ideal above p with residue degree and ramification index both equal to 1), the algorithm computes the set of all possible combinations of exponents that the primes above p can have in the factorization of $(x - y\theta)$.

The Prime Ideal Removing Algorithm.
Input: One of the primes p_l along with its prime ideal factorization $(p_l) = \mathfrak{p}_{l1}^{e_1} \cdots \mathfrak{p}_{lm_l}^{e_{m_l}}$ in $\mathcal{O}_K$ and the residue degrees f_i of the $\mathfrak{p}_{li}$.
Output: If p_l has at least one prime ideal $\mathfrak{p}_{lk}$ above it with $e_k = f_k = 1$, then for each such prime ideal $\mathfrak{p}_{lk}$ above p with $e_k = f_k = 1$, the algorithm outputs the set A_k of all tuples $(a_{l1}, \dots, a_{lm_l})$ such that $a_{lk} = 0$ and, for each $i \in \{1, \dots, m_l\} \setminus \{k\}$, a_{li} is a

possible exponent of the prime ideal $\mathfrak{p}_{li}$ in the factorization of $(x-y\theta)$. If there is no prime ideal $\mathfrak{p}_{lk}$ above p_l with $e_k = f_k = 1$, then the algorithm outputs the set A_0 of all tuples $(a_{l1}, \ldots, a_{lm_l})$, where, for each $i \in \{1, \ldots, m_l\}$, a_{li} is a possible exponent of the prime ideal $\mathfrak{p}_{li}$ in the factorization of $(x - y\theta)$.

Algorithm:

For $i = 1, \ldots, m_l$, compute the roots $\theta_i^{(1)}, \ldots, \theta_i^{(n_i)}$ of the irreducible factor of $g(t)$ in $\mathbb{Q}_{p_l}[t]$ that corresponds to $\mathfrak{p}_{li}$.

For each $i, j \in \{1, \ldots, m_l\}$, compute

$$c_{ij} = \max\{e_i, e_j\} \min_{k,l} \left\{ \operatorname{ord}_{p_l}\left(\theta_i^{(k)} - \theta_j^{(l)}\right) \right\}.$$

For each $i \in \{1, \ldots, m_l\}$, set

$$C_i = \min\left\{ \frac{1}{2}\, e_i \operatorname{ord}_p(\operatorname{Discr} g(t)), e_i \min_{k,l} \operatorname{ord}_p\left(\theta_i^{(k)} - \theta_i^{(l)}\right) \right\}.$$

IF there is some $k \in \{1, \ldots, m_l\}$ such that $e_k = f_k = 1$ THEN

FOR each $k \in \{1, \ldots, m_l\}$ such that $e_k = f_k = 1$ DO

//Consider $\mathfrak{p}_{lk}$ as the unconstrained prime above p_l.

Form the set A_k of all tuples $(a_{l1}, \ldots, a_{lm_l})$ of nonnegative integers subject to the constraints

(a) $a_{li} \le C_i, \quad 1 \le i \le m_l$

(b) $a_{li} > c_{ij} \Rightarrow a_{lj} \le c_{ij}, \quad 1 \le i, j \le m_l, \;\; i \ne j$

(c) $a_{lk} = 0$

END FOR

ELSE //there is no $k \in \{1, \ldots, m_l\}$ such that $e_k = f_k = 1$.

Form the set A_0 of all tuples $(a_{l1}, \ldots, a_{lm_l})$ of nonnegative integers subject to the constraints

(a) $a_{li} \le C_i, \quad 1 \le i \le m_l$

(b) $a_{li} > c_{ij} \Rightarrow a_{lj} \le c_{ij}, \quad 1 \le i, j \le m_l, \;\; i \ne j$

END IF

END FOR

Note that the algorithm above is a simplified version of the one we actually implement.

5 Factorization of the Thue-Mahler equation

Upon running the Prime Ideal Removing Algorithm (see Section 4) for each p_i ($i = 1, \ldots, v$), the task of solving the finite set of equations represented by (9) is reduced to the task of solving the set of equations of the form

$$(x - y\theta) = \mathfrak{a}\mathfrak{b}\mathfrak{p}_1^{u_1} \cdots \mathfrak{p}_v^{u_v} \tag{14}$$

in integer variables $x, y, u_1, \ldots, u_v$ with $u_i \geq 0$ for $i = 1, \ldots, v$. Here

- $\mathfrak{a}$ is an ideal of $\mathcal{O}_K$ of norm $|a|$.
- $\mathfrak{p}_i$ is a prime ideal of $\mathcal{O}_K$ above p_i with ramification index and residue degree both equal to 1, provided one exists. If there are no such prime ideals, $\mathfrak{p}_i = (1)$ and $u_i = 0$.
- $\mathfrak{b}$ is an ideal of $\mathcal{O}_K$ whose prime ideal factors are those that divide one of the p_i, but are not equal to one of the $\mathfrak{p}_i$.
- $u_i + \mathrm{ord}_{p_i}(N(\mathfrak{b})) = z_i$.

To be precise, $\mathfrak{b}$ is of the form

$$\mathfrak{b} = \prod_{j=1}^{m_1} \mathfrak{p}_{1j}^{a_{1j}} \cdots \prod_{j=1}^{m_v} \mathfrak{p}_{vj}^{a_{vj}},$$

where each $(a_{i1}, \ldots, a_{im_i})$ is a tuple from the Prime Ideal Removing Lemma Algorithm that corresponds to $\mathfrak{p}_i$ being the unconstrained prime above p_i. So, in particular, if $\mathfrak{p}_i = \mathfrak{p}_{ik}$, we have $a_{ik} = 0$. Of course, we must add the qualification that if there are no primes above p_i with ramification index and residue degree both equal to 1, $(a_{i1}, \ldots, a_{im_i})$ is a tuple from the set A_0 output by the Prime Ideal Removing Algorithm. Note $\mathrm{ord}_{p_i}(N(\mathfrak{b})) = \sum_{j=1}^{m_i} f_{ij} a_{ij}$.

Consider a particular instance of (14). That is, fix choices for $\mathfrak{a}, \mathfrak{b}, \mathfrak{p}_1, \ldots, \mathfrak{p}_v$. Fix a complete set of fundamental units for $\mathcal{O}_K$: $\varepsilon_1, \ldots, \varepsilon_r$. Here $r = s+t-1$, with s denoting the number of real embeddings of K into $\mathbb{C}$, and t denoting the number of complex conjugate pairs of non-real embeddings of K into $\mathbb{C}$. For $i = 1, \ldots, v$, let h_i be the smallest positive integer for which $\mathfrak{p}_i^{h_i}$ is principal and let $\pi_i \in \mathcal{O}_K$ be a generator for $\mathfrak{p}_i^{h_i}$. Then

$$(x - y\theta) = \mathfrak{a}\mathfrak{b}\mathfrak{p}_1^{s_1} \cdots \mathfrak{p}_v^{s_v} (\pi_1^{n_1}) \ldots (\pi_v^{n_v}) \tag{15}$$

where the unknown integers $s_1, \ldots, s_v, n_1, \ldots, n_v$ satisfy

$$u_i = h_i n_i + s_i, \quad n_i \geq 0, \quad 0 \leq s_i < h_i.$$

Since the s_i vary in a finite set, we treat them as parameters. Now fix values for $s_1, \ldots, s_v$. The ideal $\mathfrak{a}\mathfrak{b}\mathfrak{p}_1^{s_1} \cdots \mathfrak{p}_v^{s_v}$ is necessarily principal (because the set of nonzero fractional principal ideals forms a subgroup of the group of nonzero fractional ideals). Fix $\alpha \in \mathcal{O}_K$ such that

$$\mathfrak{a}\mathfrak{b}\mathfrak{p}_1^{s_1} \cdots \mathfrak{p}_v^{s_v} = (\alpha).$$

Since (15) is an equality of principal ideals and since generators of principal ideals are determined only up to multiplication by units, we are led to the equation

$$x - y\theta = \alpha\zeta\varepsilon_1^{a_1} \cdots \varepsilon_r^{a_r} \pi_1^{n_1} \cdots \pi_v^{n_v} \tag{16}$$

with unknowns $a_i \in \mathbb{Z}$, $n_i \in \mathbb{Z}_{\geq 0}$, and ζ in the set T of roots of unity in $\mathcal{O}_K$. Since T is finite, we will treat ζ as another parameter.

To summarize, our original problem of solving (4) has been reduced to the problem of solving finitely many equations of the form (16) for the variables $x, y, a_1, \ldots, a_r, n_1, \ldots, n_v$. Since the π_i are determined by the $\mathfrak{p}_i$, and since α is a fixed generator of $\mathfrak{a}\mathfrak{b}\mathfrak{p}_1^{s_1} \cdots \mathfrak{p}_v^{s_v}$, the parameters in (16) are $\zeta, \mathfrak{a}, \mathfrak{b}, \mathfrak{p}_1, \ldots, \mathfrak{p}_v, s_1, \ldots, s_v$. We must solve (16) for each combination of possible values of these parameters. Note that each solution $(x, y, z_1, \ldots, z_v)$ of (4) corresponds to a solution $(x, y, n_1, \ldots, n_v, a_1, \ldots, a_r)$ of some instance of (11), where

$$z_i = n_i h_i + s_i + t_i \tag{17}$$

with $t_i = \mathrm{ord}_{p_i}(N(\mathfrak{b}))$.

Note that we only need to consider half the elements in T as possible values for the parameter ζ. More specifically, we can restrict ζ to a subset T' of T formed by starting with T and removing one member of each pair $\zeta_0, -\zeta_0$. This is justified by the observation that if $\zeta_0 \in T$, and if $(x, y, n_1, \ldots, n_v, a_1, \ldots, a_r)$ satisfies an instance of (16) with $\zeta = \zeta_0$, then $(-x, -y, n_1, \ldots, n_v, a_1, \ldots, a_r)$ satisfies the corresponding instance of (16) with $\zeta = -\zeta_0$ and all other parameters unchanged.

We make one final observation. If $l \in \{1, \ldots, v\}$ is an index such that p_l has no unramified degree one prime ideals of $\mathcal{O}_K$ above it, we have $\pi_l = 1$ and $n_l = 0$. Thus, we can essentially remove such indices from consideration in the algorithm entirely. Specifically, we won't need to worry about bounding n_l at all, and we can remove the terms corresponding to l from the linear forms Λ_i ($i \neq l$) and Λ_0 (see Sections 8 and 15 for the definitions of these forms). This will significantly reduce the amount of computation we will need to do. We must still consider the index l in the numerators of c_8' and c_9', and when we test

candidate solutions of (4). In the implementation of the algorithm, if there are indices l such that p_l has no unramified degree one prime ideals dividing it, we proceed as described above. However, to save us some tedious bookkeeping in the exposition, we will assume from now on that, for each $i \in \{1, \ldots, v\}$, there is at least one prime ideal of $\mathcal{O}_K$ above p_i with ramification index and residue degree both equal to 1.

In what follows, we assume that we have fixed a particular choice of values for the parameters in (16).

6 The S-unit equation

From (16) we will deduce a so-called S-unit equation. In doing so, we will eliminate the variables x, y and set ourselves up to bound the exponents $a_1, \dots, a_r, n_1, \dots, n_v$.

Let $p \in \{p_1, \dots, p_v, \infty\}$. Denote the roots of $g(t)$ in $\overline{\mathbb{Q}_p}$ (where $\overline{\mathbb{Q}_\infty} = \overline{\mathbb{R}} = \mathbb{C}$) by $\theta^{(1)}, \dots, \theta^{(n)}$. Let $i_0, j, k \in \{1, \dots, n\}$ be distinct indices and consider the three embeddings of K into $\overline{\mathbb{Q}_p}$ defined by $\theta \mapsto \theta^{(i_0)}, \theta^{(j)}, \theta^{(k)}$. We will use $z^{(i)}$ to denote the image of z under the embedding $\theta \mapsto \theta^{(i)}$. From the identity

$$\big(\theta^{(i_0)} - \theta^{(j)}\big)\big(x - y\theta^{(k)}\big) + \big(\theta^{(j)} - \theta^{(k)}\big)\big(x - y\theta^{(i_0)}\big) + \big(\theta^{(k)} - \theta^{(i_0)}\big)\big(x - y\theta^{(j)}\big) = 0,$$

we deduce

$$\lambda = \frac{\theta^{(i_0)} - \theta^{(j)}}{\theta^{(i_0)} - \theta^{(k)}} \frac{x - y\theta^{(k)}}{x - y\theta^{(j)}} - 1 = \frac{\theta^{(j)} - \theta^{(k)}}{\theta^{(k)} - \theta^{(i_0)}} \frac{x - y\theta^{(i_0)}}{x - y\theta^{(j)}}. \tag{18}$$

Note λ is being defined by (18). Using (16) to substitute for $x - y\theta$ yields the S-unit equation

$$\lambda = \delta_1 \prod_{i=1}^{v} \left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right)^{n_i} \prod_{i=1}^{r} \left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right)^{a_i} - 1 = \delta_2 \prod_{i=1}^{v} \left(\frac{\pi_i^{(i_0)}}{\pi_i^{(j)}}\right)^{n_i} \prod_{i=1}^{r} \left(\frac{\varepsilon_i^{(i_0)}}{\varepsilon_i^{(j)}}\right)^{a_i}, \tag{19}$$

where

$$\delta_1 = \frac{\theta^{(i_0)} - \theta^{(j)}}{\theta^{(i_0)} - \theta^{(k)}} \frac{\alpha^{(k)}}{\alpha^{(j)}} \frac{\zeta^{(k)}}{\zeta^{(j)}}, \quad \delta_2 = \frac{\theta^{(j)} - \theta^{(k)}}{\theta^{(k)} - \theta^{(i_0)}} \frac{\alpha^{(i_0)}}{\alpha^{(j)}} \frac{\zeta^{(i_0)}}{\zeta^{(j)}}.$$

Note that δ_1 and δ_2 are constants (they don't depend on $(x, y, n_1, \dots, n_v, a_1, \dots, a_r)$).

We now restrict attention to those $p \in \{p_1, \dots, p_v\}$ and study the p-adic valuations of the numbers appearing in (19). Recall that an element $z \in \mathbb{C}_p$ having $\mathrm{ord}_p(z) = 0$ is called a p-adic unit.

Lemma 6.1. Let p be a rational prime. If $\gamma \in \mathcal{O}_K$ and $p \nmid N_{K/\mathbb{Q}}(\gamma)$, then $\gamma^{(h)} \in \mathbb{C}_p$ is a p-adic unit for every $h \in \{1, \dots, n\}$.

Proof. In the notation of Section 2, there is $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n_i\}$ such that $\gamma^{(h)} = \sigma_{ij}(\gamma)$. So, according to (3), $e_i \mathrm{ord}_p(\gamma^{(h)}) = \mathrm{ord}_{\mathfrak{p}_i}(\gamma)$. Therefore, if $\gamma^{(h)}$ is not a p-adic unit, we have that $\mathfrak{p}_i$ divides γ, which implies $N(\mathfrak{p}_i) = p^{f_i}$ divides $N_{K/\mathbb{Q}}(\gamma)$. □

Let $l \in \{1, \dots, v\}$.

Lemma 6.2.

(a) For every $i \in \{1, \dots, r\}$, $\dfrac{\varepsilon_i^{(i_0)}}{\varepsilon_i^{(j)}}$ and $\dfrac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}$ are p_l-adic units.

(b) For every $i \in \{1, \dots, v\}$ with $i \neq l$, $\dfrac{\pi_i^{(i_0)}}{\pi_i^{(j)}}$ and $\dfrac{\pi_i^{(k)}}{\pi_i^{(j)}}$ are p_l-adic units.

Proof. Consider any $i \in \{1, \dots, r\}$. Since ε_i is a unit in $\mathcal{O}_K$, we have $N_{K/\mathbb{Q}}(\varepsilon_i) = \pm 1$, and hence $\varepsilon_i^{(i_0)}, \varepsilon_i^{(j)}, \varepsilon_i^{(k)}$ are p_l-adic units by Lemma 6.1. Now consider an $i \in \{1, \dots, v\}$. Since $N_{K/\mathbb{Q}}(\pi_i) = \pm N(\mathfrak{p}_i^{h_i}) = \pm p_i^{f_i h_i}$, Lemma 6.1 implies $\pi_i^{(i_0)}, \pi_i^{(j)}, \pi_i^{(k)}$ are p_l-adic units if $i \neq l$. To conclude it suffices to observe that, since $\mathrm{ord}_{p_l}(x/y) = \mathrm{ord}_{p_l}(x) - \mathrm{ord}_{p_l}(y)$, any ratio of p_l-adic units is a p_l-adic unit. □

From now on we make the following choice for the index i_0. Let $g_l(t)$ be the irreducible factor of $g(t)$ in $\mathbb{Q}_{p_l}[t]$ corresponding to the prime ideal $\mathfrak{p}_l$. Since $\mathfrak{p}_l$ has ramification index and residual degree equal to 1, $\deg g_l(t) = 1$. We choose $i_0 \in \{1, \dots, n\}$ so that $\theta^{(i_0)}$ is the root of $g_l(t)$. The indices j, k are fixed, but arbitrary.

Lemma 6.3.

(a) $\dfrac{\pi_l^{(k)}}{\pi_l^{(j)}}$ is a p_l-adic unit.

(b) $\mathrm{ord}_{p_l}\left(\dfrac{\pi_l^{(i_0)}}{\pi_l^{(j)}}\right) = h_l$.

Proof. Consider the factorization of $g(t)$ in $\mathbb{Q}_{p_l}[t]$: $g(t) = g_1(t) \cdots g_m(t)$. Note $\theta^{(j)}$ is a root of some $g_h(t) \neq g_l(t)$. Let $\mathfrak{p}_h$ be the corresponding prime ideal above p_l and e_h its ramification index. Then $\mathfrak{p}_h \neq \mathfrak{p}_l$ and, since $(\pi_l) = \mathfrak{p}_l^{h_l}$, (3) implies

$$\mathrm{ord}_{p_l}(\pi_l^{(j)}) = \frac{1}{e_h}\mathrm{ord}_{\mathfrak{p}_h}(\pi_l) = 0.$$

An analogous argument gives $\mathrm{ord}_{p_l}(\pi_l^{(k)}) = 0$. On the other hand,

$$\mathrm{ord}_{p_l}(\pi_l^{(i_0)}) = \frac{1}{e_l}\mathrm{ord}_{\mathfrak{p}_l}(\pi_l) = h_l.$$

Exploiting the identity $\mathrm{ord}_{p_l}(x/y) = \mathrm{ord}_{p_l}(x) - \mathrm{ord}_{p_l}(y)$ yields the result. □

The next lemma deals with a special case in which n_l can be computed directly.

Lemma 6.4. If $\mathrm{ord}_{p_l}(\delta_1) \neq 0$, then

$$n_l = \frac{1}{h_l} \left(\min\left\{\mathrm{ord}_{p_l}(\delta_1), 0\right\} - \mathrm{ord}_{p_l}(\delta_2)\right).$$

Proof. Applying Lemmas 6.2 and 6.3 to both expressions for λ in (19) yields $\mathrm{ord}_{p_l}(\lambda) = \min\left\{\mathrm{ord}_{p_l}(\delta_1), \mathrm{ord}_{p_l}(1)\right\}$ and $\mathrm{ord}_{p_l}(\lambda) = \mathrm{ord}_{p_l}(\delta_2) + n_l h_l$. □

In Section 9, we will discuss how to choose the indices j and k in the p_l-adic cases. We will work with the indices i_0, j, k in the $p = \infty$ case in Section 15.

7 The p-adic logarithm

In this section, we introduce the p-adic logarithm function, extend its domain to include all p-adic units in $\overline{\mathbb{Q}_p}$, and state several of its important properties.

Fix a rational prime p. For $z \in \mathbb{C}_p$ with $\mathrm{ord}_p(z-1) > 0$, the p-adic logarithm of z is defined to be

$$\log_p z = -\sum_{i=1}^{\infty} \frac{(1-z)^i}{i}.$$

By the n^{th} term test, the series converges precisely in the region where $\mathrm{ord}_p(z-1) > 0$. Three important properties of the p-adic logarithm are:

1. $\log_p(xy) = \log_p(x) + \log_p(y)$ whenever $\mathrm{ord}_p(x-1) > 0$ and $\mathrm{ord}_p(y-1) > 0$;
2. $\log_p(z^k) = k \log_p(z)$ whenever $\mathrm{ord}_p(z-1) > 0$ and $k \in \mathbb{Z}$;
3. $\mathrm{ord}_p(\log_p z) = \mathrm{ord}_p(z-1)$ whenever $\mathrm{ord}_p(z-1) > 1/(p-1)$.

Proofs of the first and last property can be found in [Ha] (pp. 264-265). The second property follows from the first.

We will use the following lemma to extend the definition of the p-adic logarithm to include all p-adic units in $\overline{\mathbb{Q}_p}$.

Lemma 7.1. Let z be a p-adic unit belonging to a finite extension L of $\mathbb{Q}_p$. Let e and f be the ramification index and residue degree of L.

(a) There is a positive integer r such that $\mathrm{ord}_p(z^r - 1) > 0$.

(b) If r is the smallest positive integer having $\mathrm{ord}_p(z^r - 1) > 0$, then r divides $p^f - 1$, and an integer q satisfies $\mathrm{ord}_p(z^q - 1) > 0$ if and only if it is a multiple of r.

(c) If r is a nonzero integer with $\mathrm{ord}_p(z^r - 1) > 0$, and if k is an integer with $p^k(p-1) > e$, then

$$\mathrm{ord}_p(z^{rp^k} - 1) > \frac{1}{p-1}.$$

Proof. The residue field of L is the quotient $R/\mathfrak{m}$, where $R = \{z' \in L : \mathrm{ord}_p(z') \geq 0\}$ and $\mathfrak{m} = \{z' \in L : \mathrm{ord}_p(z') > 0\}$. It is a finite field containing p^f elements. Since z is a p-adic unit, the image of z in $R/\mathfrak{m}$ belongs to the multiplicative group $(R/\mathfrak{m})^\times$. Let r be the order of the image of z in $(R/\mathfrak{m})^\times$. So, by definition, r is the smallest positive integer with $\mathrm{ord}_p(z^r - 1) > 0$, and it follows that r divides $p^f - 1$ and that an integer q satisfies $\mathrm{ord}_p(z^q - 1) > 0$ if and only if it is a multiple of r. We have now proved (a) and (b). For

the proof of (c), let r be any nonzero integer with $\mathrm{ord}_p(z^r-1)>0$ and let k be an integer with $p^k(p-1)>e$. Write

$$z^{rp^k}=(1+(z^r-1))^{p^k}=1+\sum_{l=1}^{p^k-1}\binom{p^k}{l}(z^r-1)^l+(z^r-1)^{p^k}. \tag{20}$$

The least positive value that ord_p attains on L is $1/e$, so we know $\mathrm{ord}_p(z^r-1)\geq 1/e$. Applying this inequality and the fact that p divides $\binom{p^k}{l}$ for $l=1,\ldots,p^k-1$ in (20) yields

$$\mathrm{ord}_p(z^{rp^k}-1)\geq \min\left\{1+\frac{1}{e},\ldots,1+\frac{p^k-1}{e},\frac{p^k}{e}\right\}>\frac{1}{p-1},$$

where the last inequality follows from definition of k. □

For z a p-adic unit in $\overline{\mathbb{Q}_p}$, we define

$$\log_p z=\frac{1}{q}\log_p z^q,$$

where q is an arbitrary nonzero integer such that $\mathrm{ord}_p(z^q-1)>0$. To see that this definition is independent of q, let r be the smallest positive integer with $\mathrm{ord}_p(z^r-1)>0$, note that q/r is an integer, and use the second property of p-adic logarithms above to write

$$\frac{1}{q}\log_p z^q=\frac{1}{r(q/r)}\log_p z^{r(q/r)}=\frac{1}{r}\log_p z^r.$$

Choosing q such that $\mathrm{ord}_p(z^q-1)>1/(p-1)$ helps to speed up and control the convergence of the series defining $\log_p$ (cf. [Sm] (pp. 28-30) and [Coh2] (pp. 263-265)).

It is straightforward to see that Properties 1 and 2 above extend to the case where x,y,z are p-adic units. Combining this fact with Property 3, we obtain:

Lemma 7.2. Let $z_1,\ldots,z_m\in\overline{\mathbb{Q}_p}$ be p-adic units and let $b_1,\ldots,b_m\in\mathbb{Z}$. If

$$\mathrm{ord}_p(z_1^{b_1}\cdots z_m^{b_m}-1)>\frac{1}{p-1}$$

then

$$\mathrm{ord}_p(b_1\log_p z_1+\ldots+b_m\log_p z_m)=\mathrm{ord}_p(z_1^{b_1}\cdots z_m^{b_m}-1).$$

8 A small upper bound for n_l in a special case

Let $l \in \{1, \dots, v\}$. In this section, we will identify conditions under which n_l can be bounded by a small explicit constant. We will need to assume that $\mathrm{ord}_{p_l}(\delta_1) = 0$.

Here (and in Sections 19-23) we find it convenient to use the notation

$$b_1 = 1, \qquad b_{1+i} = n_i \quad (i = 1, \dots, v), \qquad b_{1+v+i} = a_i \quad (i = 1, \dots, r).$$

Put

$$\alpha_1 = \log_{p_l} \delta_1, \quad \alpha_{1+i} = \log_{p_l}\left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right) \quad (i = 1, \dots, v), \quad \alpha_{1+v+i} = \log_{p_l}\left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right) \quad (i = 1, \dots, r).$$

Define

$$\Lambda_l = \sum_{i=1}^{1+v+r} b_i \alpha_i$$

Let L be a finite extension of $\mathbb{Q}_{p_l}$ containing δ_1, $\pi_i^{(k)}/\pi_i^{(j)}$ $(i = 1, \dots, v)$, $\varepsilon_i^{(k)}/\varepsilon_i^{(j)}$ $(i = 1, \dots, r)$. Since finite extensions of p-adic fields are complete, $\alpha_i \in L$ for $i = 1, \dots, 1+v+r$ also. Choose $\phi \in \overline{\mathbb{Q}_{p_l}}$ such that $L = \mathbb{Q}_{p_l}(\phi)$ and $\mathrm{ord}_{p_l}(\phi) \geq 0$. It is always possible to satisfy the second condition because $\mathbb{Q}_{p_l}(\phi') = \mathbb{Q}_{p_l}(p_l^m \phi')$ for every integer m and every $\phi' \in \overline{\mathbb{Q}_{p_l}}$. Let $G(t)$ be the minimal polynomial of ϕ over $\mathbb{Q}_{p_l}$, and let S be its degree. For $i = 1, \dots, 1 + v + r$, write

$$\alpha_i = \sum_{h=1}^{S} \alpha_{ih} \phi^{h-1}, \quad \alpha_{ih} \in \mathbb{Q}_{p_l}.$$

Then

$$\Lambda_l = \sum_{h=1}^{S} \Lambda_{lh} \phi^{h-1} \tag{21}$$

with

$$\Lambda_{lh} = \sum_{i=1}^{v+r+1} b_i \alpha_{ih} \qquad (h = 1, \dots, S).$$

Lemma 8.1. For every $h \in \{1, \dots, S\}$, we have

$$\mathrm{ord}_{p_l}(\Lambda_{lh}) \geq \mathrm{ord}_p(\Lambda_l) - u_l, \tag{22}$$

where

$$u_l = \mathrm{ord}_{p_l}\left(\prod_{1 \leq j < k \leq S} (\phi^{(k)} - \phi^{(j)})\right) = \frac{1}{2}\mathrm{ord}_{p_l}(\mathrm{Discr}\, G(t)).$$

Proof. Taking the images of (21) under the conjugation maps $\phi \mapsto \phi^{(h)}$ $(h = 1, \ldots, S)$ gives

$$\begin{bmatrix} \Lambda_l^{(1)} \\ \vdots \\ \Lambda_l^{(S)} \end{bmatrix} = \begin{bmatrix} 1 & \phi^{(1)} & \cdots & \phi^{(1)S-1} \\ \vdots & \vdots & & \vdots \\ 1 & \phi^{(S)} & \cdots & \phi^{(S)S-1} \end{bmatrix} \begin{bmatrix} \Lambda_{l1} \\ \vdots \\ \Lambda_{lS} \end{bmatrix}.$$

The $S \times S$ matrix $\left(\phi^{(h)i-1}\right)$ above has inverse

$$\frac{1}{\prod\limits_{1 \le j < k \le S} (\phi^{(k)} - \phi^{(j)})} \begin{bmatrix} \gamma_{11} & \cdots & \gamma_{1S} \\ \vdots & & \vdots \\ \gamma_{S1} & \cdots & \gamma_{SS} \end{bmatrix},$$

where each γ_{jk} is a polynomial in the entries of $\left(\phi^{(h)i-1}\right)$ having integer coefficients. Since $\mathrm{ord}_{p_l}(\phi) \ge 0$ and since $\mathrm{ord}_{p_l}(\phi^{(h)}) = \mathrm{ord}_{p_l}(\phi)$ for every h, it follows that $\mathrm{ord}_{p_l}(\gamma_{jk}) \ge 0$ for every γ_{jk}. Therefore, since $\Lambda_{lh} = \prod_{1 \le j < k \le S} (\phi^{(k)} - \phi^{(j)})^{-1} \sum_{i=1}^{S} \gamma_{hi} \Lambda_l^{(i)}$, we have

$$\mathrm{ord}_{p_l}(\Lambda_{lh}) \ge \min_{1 \le i \le S} \mathrm{ord}_{p_l}(\Lambda_l^{(i)}) - u_l = \mathrm{ord}_{p_l}(\Lambda_l) - u_l$$

for every $h \in \{1, \ldots, S\}$. □

Lemma 8.2. If $n_l > \dfrac{1}{h_l}\left(\dfrac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)$, then

$$\mathrm{ord}_{p_l}(\Lambda_l) = n_l h_l + \mathrm{ord}_{p_l}(\delta_2).$$

Proof. Immediate from Lemma 7.2. □

Lemma 8.3.

(a) If $\mathrm{ord}_p(\alpha_1) < \min\limits_{2 \le i \le 1+v+r} \mathrm{ord}_p(\alpha_i)$, then

$$n_l \le \max\left\{ \left\lfloor \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rfloor, \left\lceil \frac{1}{h_l}\left(\min_{2 \le i \le 1+v+r} \mathrm{ord}_p(\alpha_i) - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rceil - 1 \right\}.$$

(b) For all $h \in \{1, \ldots, S\}$, if $\mathrm{ord}_p(\alpha_{1h}) < \min\limits_{2 \le i \le 1+v+r} \mathrm{ord}_p(\alpha_{ih})$, then

$$n_l \le \max\left\{ \left\lfloor \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rfloor, \left\lceil \frac{1}{h_l}\left(u_l + \min_{2 \le i \le 1+v+r} \mathrm{ord}_p(\alpha_{ih}) - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rceil - 1 \right\}.$$

Proof. The proof of (a) is analogous to (and simpler than) the proof of (b). Hence we prove only (b). Let $h \in \{1, \ldots, S\}$. First observe that

$$\mathrm{ord}_{p_l}(\alpha_{1h}) = \mathrm{ord}_{p_l}\left(\Lambda_{lh} - \sum_{i=2}^{1+v+r} b_i \alpha_{ih}\right) \ge \min\left(\mathrm{ord}_{p_l}(\Lambda_{lh}), \min_{2 \le i \le 1+v+r} \mathrm{ord}_p(\alpha_{ih})\right).$$

Therefore, it suffices to show that

$$\mathrm{ord}_{p_l}(\Lambda_{lh}) \geq \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_p(\alpha_{ih})$$

assuming the inequalities

$$n_l > \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right), \qquad n_l \geq \frac{1}{h_l}\left(u_l + \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_p(\alpha_{ih}) - \mathrm{ord}_{p_l}(\delta_2)\right).$$

By Lemma 8.2, the first inequality implies $\mathrm{ord}_{p_l}(\Lambda_l) = n_l h_l + \mathrm{ord}_{p_l}(\delta_2)$. Combining this with (22) yields

$$\mathrm{ord}_{p_l}(\Lambda_{lh}) \geq n_l h_l + \mathrm{ord}_{p_l}(\delta_2) - u_l.$$

The desired result now follows from the second assumed inequality. □

9 Choosing the indices j and k in the p_l-adic cases

We give here some guidelines for choosing the indices j and k discussed in Section 6 in the p_l-adic cases.

Let $l \in \{1, \ldots, v\}$ be given. Recall that we require the indices i_0, j, k to be distinct. Also recall that i_0 has already been chosen so that $\theta^{(i_0)}$ is the root of a particular linear factor of $g(t)$ in $\mathbb{Q}_{p_l}[t]$. We list the guidelines in the order in which they should be followed:

1. If possible, one should choose j, k such that $\mathrm{ord}_{p_l}(\delta_1) \neq 0$ so that Lemma 6.4 applies and yields n_l exactly. If this is not possible we know that $\mathrm{ord}_{p_l}(\delta_1) = 0$ for every choice of j, k.
2. Try to choose j, k in such a way that Lemma 8.3 can be used to obtain an upper bound for n_l. Such an upper bound will be essentially equal to the smallest upper bound that could possibly be obtained through our general procedure for bounding n_l (which we will describe in later sections). Moreover, the general procedure is typically much more computationally expensive than the process of searching through the possible values of j, k and checking the hypotheses of Lemma 8.3.
3. If $g(t)$ has three or more linear factors in $\mathbb{Q}_{p_l}[t]$, choose j, k so that $\theta^{(j)}, \theta^{(k)}$ are roots of such factors.
4. If $g(t)$ has an irreducible factor in $\mathbb{Q}_{p_l}[t]$ of degree two, choose j, k so that $\theta^{(j)}, \theta^{(k)}$ are the roots of such a factor.
5. If $g(t)$ has a nonlinear irreducible factor in $\mathbb{Q}_{p_l}[t]$ that splits completely in the extension of $\mathbb{Q}_{p_l}$ that it generates, choose j, k so that $\theta^{(j)}, \theta^{(k)}$ are roots of such a factor. To check whether $g(t)$ has such a factor, it may be helpful to use the fact that the extension of $\mathbb{Q}_{p_l}$ generated by an irreducible factor $g_i(t)$ of $g(t)$ is isomorphic to the completion of K at the prime ideal above p_l to which $g_i(t)$ corresponds.
6. Try to choose j, k, so that $\alpha_1, \ldots, \alpha_{1+v+r}$ (see Section 8) all belong to $\mathbb{Q}_{p_l}$.
7. Try to choose j, k, so that there exists an index $i' \in \{2, \ldots, 1+v+r\}$ such that $\mathrm{ord}_{p_l}(\alpha_{i'}) = \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_{p_l}(\alpha_i)$ and $\alpha_i/\alpha_{i'} \in \mathbb{Q}_{p_l}$ for $i = 1, \ldots, 1+v+r$.
8. Select a nonlinear irreducible factor of $g(t)$ in $\mathbb{Q}_{p_l}[t]$ of minimal degree and choose j, k so that $\theta^{(j)}, \theta^{(k)}$ are roots of this polynomial. The case where $g(t)$ has no nonlinear factors in $\mathbb{Q}_{p_l}[t]$ is handled in Guideline 3.

Guidelines 1, 2, 6, and 7 involve searching through all possible choices of j, k until a choice satisfying certain conditions is found. Note that we can restrict these searches to

those choices with $j < k$. This is because going from the choice $j = i_1$, $k = i_2$ to the choice $j = i_2$, $k = i_1$ just has the effect of multiplying $\mathrm{ord}_{p_l}(\delta_1)$ and the α_i by -1.

If we are successful in choosing j, k such that $\mathrm{ord}_{p_l}(\delta_1) \neq 0$, then we know the value of n_l (by Lemma 6.4). If n_l is not an integer or if $n_l < 0$, we know the particular instance of (16) we are considering has no solutions. If n_l is a nonnegative integer, we can absorb $\pi_l^{n_l}$ into α and essentially remove the index $l \in \{1, \ldots, v\}$ from consideration in the rest of the algorithm. Specifically, we can skip any work we would do to bound n_l, and we can remove the terms corresponding to l from the linear forms Λ_i ($i \neq l$) and Λ_0 (see Sections 8 and 15 for the definitions of these forms). This will save us a substantial amount of computation time. Note that we will still need to consider the index l in the numerators of c_8' and c_9', and when we test candidate solutions of (4). We do all of what we have just described in the implementation of the algorithm. However, in order to simplify the exposition, we will assume from now on that $\mathrm{ord}_{p_l}(\delta_1) = 0$.

10 The general strategy for bounding the n_i and the a_i

Here we outline our strategy for bounding the a_i and for bounding those n_i to which lemma 8.3 does not apply. Put

$$N = \max\{n_1, \ldots, n_v\}, \quad A = \max\{|a_1|, \ldots, |a_r|\}, \quad H = \max\{N, A\}.$$

We will find an explicit upper bound for H. Our strategy depends on the simple observation that if H satisfies an inequality of the form

$$H \leq a + b \log H,$$

for some constants a, b, then H must be bounded (see Lemma 10.1 below). To obtain such an inequality for H, we will bound N and A by linear functions of $\log H$ separately. To bound N, we will utilize the theory of linear forms in p-adic logarithms. For A, we will appeal to the theory of linear forms in real/complex logarithms.

Lemma 10.1. Let a and b be real numbers with $b \geq e^2$. If $x \leq a + b \log x$, then $x < 2(a + b \log b)$.

Proof. Let $y = x/b \log b - 1$. Then

$$x \leq a + b \log x = a + b \log(1 + y) + b \log b + b \log \log b \leq a + by + b \log b + b \log \log b$$

By substituting $y = x/b \log b - 1$ in the last expression and rearranging, we find

$$x \leq b \log b + \frac{\log b}{\log b - 1}(a + b \log \log b) < b \log b + 2\left(a + \frac{1}{2} b \log b\right),$$

where the last inequality assumes that $b \geq e^2$. □

11 A lower bound for linear forms in p-adic logarithms

For α an algebraic number, the absolute logarithmic height of α is

$$h(\alpha) = \frac{1}{N} \log \left(a \prod_{i=1}^{N} \max \left\{ 1, \left| \alpha^{(i)} \right| \right\} \right),$$

where a, N, and $\alpha^{(1)}, \ldots, \alpha^{(N)}$ denote, respectively, the leading coefficient, degree, and roots in $\mathbb{C}$ of the minimal polynomial of α over $\mathbb{Z}$.

Fix a rational prime p. Let $\alpha_1, \ldots, \alpha_m$ be nonzero algebraic numbers and let L be a number field of degree D containing the α_i. Let $\mathfrak{p}$ be a prime ideal of the ring of integers of L lying above p. Let $e_\mathfrak{p}$ and $f_\mathfrak{p}$ be, respectively, the ramification index and residue degree of $\mathfrak{p}$. If $p = 2$, define

$$d = \begin{cases} D & \text{if } e^{2\pi i/3} \in L, \\ 2D & \text{if } e^{2\pi i/3} \notin L, \end{cases} \qquad f = \begin{cases} f_\mathfrak{p} & \text{if } e^{2\pi i/3} \in L, \\ \max\{2, f_\mathfrak{p}\} & \text{if } e^{2\pi i/3} \notin L. \end{cases}$$

If $p \geq 3$ and $p^{f_\mathfrak{p}} \equiv 3 \pmod 4$, set

$$d = D, \qquad f = f_\mathfrak{p}.$$

Finally, if $p \geq 3$ and $p^{f_\mathfrak{p}} \equiv 1 \pmod 4$, set

$$d = \begin{cases} D & \text{if } e^{2\pi i/4} \in L, \\ 2D & \text{if } e^{2\pi i/4} \notin L, \end{cases} \qquad f = \begin{cases} f_\mathfrak{p} & \text{if } e^{2\pi i/4} \in L \text{ or } p \equiv 1 \pmod 4, \\ \max\{2, f_\mathfrak{p}\} & \text{if } e^{2\pi i/4} \notin L \text{ and } p \equiv 3 \pmod 4. \end{cases}$$

Put

$$\kappa = \left\lfloor \frac{2 e_\mathfrak{p} \log p}{p - 1} \right\rfloor$$

and

$$(\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6) = \begin{cases} (160, 32, 40, 276, 16, 8) & \text{if } p = 2, \\ (759, 16, 20, 1074, 8, 4) & \text{if } p = 3, d \geq 2, \\ (537, 16, 20, 532, 8, 4) & \text{if } p = 3, d = 1, \\ (1473, 8\tau, 10, 394\tau, 8, 4) & \text{if } p \geq 5, e_\mathfrak{p} = 1, p \equiv 1 \pmod 4, \\ (1282, 8\tau, 10, 366\tau, 8, 4) & \text{if } p \geq 5, e_\mathfrak{p} = 1, p \equiv 3 \pmod 4, d \geq 2, \\ (1288, 8\tau, 10, 396\tau, 8, 4) & \text{if } p \geq 5, e_\mathfrak{p} = 1, p \equiv 3 \pmod 4, d = 1, \\ (319, 16, 20, 402, 8, 4) & \text{if } p = 5, e_\mathfrak{p} \geq 2, \\ (1502, 16, 20, 1372, 8, 4) & \text{if } p \geq 7, e_\mathfrak{p} \geq 2, p \equiv 1 (\mathrm{mod} 4), \\ (2190, 16, 20, 1890, 8, 4) & \text{if } p \geq 7, e_\mathfrak{p} \geq 2, p \equiv 3 (\mathrm{mod} 4), \end{cases}$$

where $\tau = (p-1)/(p-2)$. Set

$$q = \begin{cases} 3 & \text{if } p = 2, \\ 2 & \text{if } p \geq 3. \end{cases} \qquad L' = \begin{cases} L(e^{2\pi i/3}) & \text{if } p = 2, \\ L(e^{2\pi i/4}) & \text{if } p \geq 3 \text{ and } p^{f_{\mathfrak{p}}} \equiv 1 \pmod 4, \\ L & \text{if } p \geq 3 \text{ and } p^{f_{\mathfrak{p}}} \equiv 3 \pmod 4. \end{cases}$$

Define $u = \max\left\{t \in \mathbb{N} : e^{2\pi i/q^t} \in L'\right\}$, so that q^u is the order of the maximal q-subgroup of the group of roots of unity in L'. Set

$$c_2 = \frac{(m+1)^{m+2}d^{m+2}}{(m-1)!}\frac{p^f}{(f\log p)^3}\max\{1, \log d\}\max\left\{\log(e^4(m+1)d), e_{\mathfrak{p}}, f\log p\right\}$$

$$c_3' = \kappa_1\kappa_2^m\left(\frac{m}{f\log p}\right)^{m-1}\prod_{i=1}^{m}\max\left\{h(\alpha_i), \frac{f\log p}{\kappa_3(m+4)d}\right\}$$

$$c_3'' = \kappa_4(e\kappa_5)^m p^{\kappa(m-1)}\prod_{i=1}^{m}\max\left\{h(\alpha_i), \frac{f\log p}{e^2\kappa_6 p^{\kappa}d}\right\}.$$

Let $b_1, \ldots, b_m \in \mathbb{Z}$, and put

$$\lambda = \alpha_1^{b_1}\cdots\alpha_m^{b_m} - 1, \qquad B = \max\{|b_1|, \ldots, |b_m|\}.$$

The following theorem is a consequence of Theorems 1 and 3 of [Yu2] and the lemma in the appendix of [Yu1].

Theorem 11.1. Assume $\mathrm{ord}_{\mathfrak{p}}(\alpha_i) = 0$ for $i = 1, \ldots, m$. If $\lambda \neq 0$ and $B \geq 3$, then

$$\mathrm{ord}_p(\lambda) < \frac{c_2\min\{c_3', c_3''\}}{q^u}\log B.$$

A slightly weaker but computationally easier version of the theorem is obtained upon replacing q^u by

$$Q = \begin{cases} 3 & \text{if } p = 2, \\ 4 & \text{if } p \geq 3 \text{ and } p^f \equiv 1 \pmod 4, \\ 1 & \text{if } p \geq 3 \text{ and } p^f \equiv 3 \pmod 4. \end{cases}$$

12 A lower bound for linear forms in real/complex logarithms

Let $\alpha_1, \ldots, \alpha_m$ be nonzero algebraic numbers and let L be a number field of degree d containing the α_i. Let L be embedded into $\mathbb{C}$ and identify the α_i with their images. For $i = 1, \ldots, m$, fix a nonzero determination of $\log \alpha_i$. If L is embedded in $\mathbb{R}$, set $\kappa = 1$; otherwise, set $\kappa = 2$. Put

$$c_{17} = \min\left\{\frac{1}{\kappa}\left(\frac{1}{2}em\right)^{\kappa} 30^{m+3} m^{7/2}, 2^{6m+20}\right\} d^2 \log(ed) \prod_{i=1}^{m} \max\left\{dh(\alpha_i), |\log \alpha_i|, 0.16\right\}.$$

Let $b_1, \ldots, b_m \in \mathbb{Z}$, and put

$$\Lambda = b_1 \log \alpha_1 + \cdots + b_m \log \alpha_m, \qquad B = \max\left\{|b_1|, \ldots, |b_m|\right\}.$$

The following theorem is Corollary 2.3 of [Mat].

Theorem 12.1. If $\Lambda \neq 0$, then

$$|\Lambda| > \exp(-c_{17} \log B - c_{17}).$$

13 A bound for n_l in terms of $\log H$

In this section, we show how the theory of linear forms in p-adic logarithms can be used to bound each n_l by a linear function of $\log H$.

Lemma 13.1. For each $l \in \{1, \dots, v\}$, there is an effectively computable constant $c_1(l)$ such that

$$n_l < \frac{c_1(l)}{h_l} \log H - \frac{1}{h_l} \mathrm{ord}_{p_l}(\delta_2)$$

holds whenever $H > 2$.

Proof. Fix $l \in \{1, \dots, v\}$. By Lemmas 6.2 and 6.3 applied to (19), we have $n_l h_l = \mathrm{ord}_{p_l}(\lambda) - \mathrm{ord}_{p_l}(\delta_2)$. So it suffices to find an effectively computable constant $c_1(l)$ such that $\mathrm{ord}_{p_l}(\lambda) < c_1(l) \log H$.

Let F be the splitting field of g over $\mathbb{Q}$ and let σ be an embedding of F into $\overline{\mathbb{Q}_{p_l}}$. Put

$$\alpha_1 = \sigma^{-1}(\delta_1), \quad \alpha_{1+i} = \sigma^{-1}\left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right) \quad (i = 1, \dots, v), \quad \alpha_{1+v+i} = \sigma^{-1}\left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right) \quad (i = 1, \dots, r).$$

Since the preimages of $\theta^{(i_0)}, \theta^{(j)}, \theta^{(k)}$ are easy to determine and since $\pi_1, \dots, \pi_v, \varepsilon_1, \dots, \varepsilon_r \in \mathbb{Q}(\theta)$, it is straightforward to calculate the α_i. Set $L = \mathbb{Q}(\alpha_1, \dots, \alpha_{1+v+r})$. Since the restriction of σ to L is an embedding of L into $\overline{\mathbb{Q}_{p_l}}$, it corresponds to a prime ideal $\mathfrak{p}$ of L above p_l (precisely speaking, it is the equivalence class of embeddings of L into $\overline{\mathbb{Q}_{p_l}}$ to which the restriction of σ belongs that corresponds to $\mathfrak{p}$, not the embedding itself). So we are in the situation of Section 11 (with $p = p_l$ and $m = 1 + v + r$), and we can compute the constants q^u,c_2,c_3',c_3'' appearing there. Note that these constants depend on the embedding σ chosen only through the ramification index $e_{\mathfrak{p}}$ and residue degree $f_{\mathfrak{p}}$ of $\mathfrak{p}$. Put

$$c_1(l) = \frac{c_2 \max\{c_3', c_3''\}}{e_{\mathfrak{p}} q^u}.$$

By Lemmas 6.2 and 6.3 and since $\mathrm{ord}_{p_l}(\delta_1) = 0$, we know the $\sigma(\alpha_i)$ are p_l-adic units. Therefore, $\mathrm{ord}_{\mathfrak{p}}(\alpha_i) = e_{\mathfrak{p}} \mathrm{ord}_{p_l}(\sigma(\alpha_i)) = 0$ for all i. Identify λ with its preimage $\alpha_1^{b_1} \cdots \alpha_{1+v+r}^{b_{1+v+r}} - 1$ in L. By (19), $\lambda \neq 0$. So, by Theorem 11.1, we have

$$\mathrm{ord}_{p_l}(\lambda) = \frac{1}{e_{\mathfrak{p}}} \mathrm{ord}_{\mathfrak{p}}(\lambda) \leq \frac{c_2 \max\{c_3', c_3''\}}{e_{\mathfrak{p}} q^u} \log H = c_1(l) \log H$$

provided $H > 2$. To reduce the work involved in computing $c_1(l)$, we can replace q^u above by the constant Q from Section 11.

Here is a simple method to choose the embedding σ and determine the prime ideal $\mathfrak{p}$ of L corresponding to it. Just choose a prime ideal $\mathfrak{P}$ of F that divides p_l and take σ to be the embedding of F into the completion $F_{\mathfrak{P}}$. Then $\mathfrak{p}$ is the unique ideal of L below $\mathfrak{P}$: $\mathfrak{p} = \mathfrak{P} \cap L$. □

For each $l \in \{1, \dots, v\}$ for which Lemma 8.3 yields upper bounds for n_l, let N_l^* be the smallest such upper bound. Let I be the set of all indices $l \in \{1, \dots, v\}$ for which Lemma 8.3 does not furnish an upper bound for n_l.

Put

$$c_4' = \max_{l \in I} \frac{c_1(l)}{h_l}, \quad c_4'' = \max_{l \notin I} N_l^*, \quad c_4 = \max\left\{c_4', c_4''\right\}, \quad c_5 = \max_{l \in I} \frac{-1}{h_l} \mathrm{ord}_{p_l}(\delta_2).$$

Lemma 13.2. If $H > 2$, we have

$$N \leq c_4 \log H + c_5.$$

Proof. Immediate from Lemma 13.1. □

Put

$$c_6 = \left\lceil 2 \max_{l \in I} \left(\frac{-1}{h_l} \mathrm{ord}_{p_l}(\delta_2) + \max\left\{ \frac{c_1(l)}{h_l}, e^2 \right\} \log\left(\max\left\{ \frac{c_1(l)}{h_l}, e^2 \right\} \right) \right) \right\rceil - 1.$$

Lemma 13.3. If $H = N$, we have

$$N \leq c_7 = \max\left\{2, c_4'', c_6\right\}.$$

Proof. We have $N = n_l$ for some $l \in \{1, \dots, v\}$. If $l \notin I$, then $N \leq N_l^* \leq c_4''$, and we are done. If $l \in I$, then, since $H = N$, Lemma 13.1 says we have

$$N < \frac{c_1(l)}{h_l} \log N - \frac{1}{h_l} \mathrm{ord}_{p_l}(\delta_2)$$

whenever $N > 2$. It follows from Lemma 10.1 that

$$N < 2 \left(\frac{-1}{h_l} \mathrm{ord}_{p_l}(\delta_2) + \max\left\{ \frac{c_1(l)}{h_l}, e^2 \right\} \log\left(\max\left\{ \frac{c_1(l)}{h_l}, e^2 \right\} \right) \right)$$

if $N > 2$. □

14 A bound for A when $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right|$ is large

In this section, we establish an upper bound for A in terms of $\log H$ in the case where $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right|$ is large.

Here and in the next section, $\theta^{(1)},\dots,\theta^{(n)}$ will denote the roots of $g(t)$ in $\mathbb{C}$. We order the roots of $g(t)$ in $\mathbb{C}$ so that

$$\theta^{(1)},\dots,\theta^{(s)}\in\mathbb{R},\qquad \theta^{(s+1)}=\overline{\theta^{(s+t+1)}},\dots,\theta^{(s+t)}=\overline{\theta^{(s+2t)}}\in\mathbb{C}\setminus\mathbb{R},$$

where $n=s+2t$. Put

$$c_8'=\log\frac{|a|\,p_1^{s_1+t_1}\cdots p_v^{s_v+t_v}}{\min_{1\le i\le n}\left|\alpha^{(i)}\zeta^{(i)}\right|},\qquad c_9'=\log\frac{p_1^{h_1}\cdots p_v^{h_v}}{\min_{1\le i\le n}\left|\pi_1^{(i)}\cdots\pi_v^{(i)}\right|},$$

$$c_8''=\log\max_{1\le i\le n}\left|\alpha^{(i)}\zeta^{(i)}\right|,\qquad c_9''=\log\max_{1\le i\le n}\left|\pi_1^{(i)}\cdots\pi_v^{(i)}\right|.$$

For any $r\times r$ matrix $U=(u_{ij})$, the row-norm of U is

$$\|U\|=\max_{1\le i\le r}\sum_{j=1}^{r}|u_{ij}|\,.$$

For each set $\kappa=\{k_1,\dots,k_r\}\subseteq\{1,\dots,s+t\}$ of indices with $k_1<\cdots<k_r$, define

$$U_\kappa=(u_{ij})=\left(\log\left|\varepsilon_j^{(k_i)}\right|\right).$$

Note each U_κ is invertible because its determinant is a multiple of the regulator of K. Set

$$c_{10}=\frac{1}{\min_{\kappa}\left\|U_\kappa^{-1}\right\|},$$

and let c_{11} be any number satisfying

$$0<c_{11}<\frac{c_{10}}{n-1}.$$

So we have

$$c_{10}-c_{11}>c_{10}-(n-1)c_{11}>0.$$

Lemma 14.1. If $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right|>e^{-c_{11}A}$, we have

$$A<\frac{c_8'+c_9'N}{c_{10}-(n-1)c_{11}}\qquad\text{or}\qquad A<\frac{c_8''+c_9''N}{c_{10}-c_{11}}.$$

Proof. Let $\kappa_0 = \{k_1, \dots, k_r\}$ be an index set such that

$$\left\|U_{\kappa_0}^{-1}\right\| = \min_{\kappa} \left\|U_{\kappa}^{-1}\right\| = \frac{1}{c_{10}}.$$

Put $\varepsilon = \varepsilon_1^{a_1} \dots \varepsilon_r^{a_r}$. Since

$$\log \left|\varepsilon^{(k_i)}\right| = \sum_{j=1}^{r} a_j \log \left|\varepsilon_j^{(k_i)}\right| \qquad (i = 1, \dots, r),$$

we have

$$a_i = \sum_{j=1}^{r} u'_{ij} \log \left|\varepsilon^{(k_j)}\right| \qquad (i = 1, \dots, r),$$

where $(u'_{ij}) = U_{\kappa_0}^{-1}$. Choose $k \in \kappa_0$ such that

$$\left|\log \left|\varepsilon^{(k)}\right|\right| = \max_{k \in \kappa_0} \left|\log \left|\varepsilon^{(i)}\right|\right|.$$

Then

$$A = \max_{1 \le i \le r} |a_i| \le \max_{1 \le i \le r} \sum_{j=1}^{r} \left|u'_{ij}\right| \left|\log \left|\varepsilon^{(k_j)}\right|\right| \le \left|\log \left|\varepsilon^{(k)}\right|\right| \max_{1 \le i \le r} \sum_{j=1}^{r} \left|u'_{ij}\right| = \left|\log \left|\varepsilon^{(k)}\right|\right| \left\|U_{\kappa_0}^{-1}\right\|.$$

It follows that we have either $\left|\varepsilon^{(k)}\right| \le e^{-c_{10}A}$ or $\left|\varepsilon^{(k)}\right| \ge e^{c_{10}A}$. According to (16),

$$\left|\varepsilon^{(k)}\right| = \frac{\left|x - y\theta^{(k)}\right|}{\left|\alpha^{(k)}\right| \left|\zeta^{(k)}\right| \left|\pi_1^{(k)}\right|^{n_1} \cdots \left|\pi_v^{(k)}\right|^{n_v}}.$$

So $\left|\varepsilon^{(k)}\right| \le e^{-c_{10}A}$ implies

$$\begin{aligned} e^{-c_{10}A} &\ge \frac{\left|x - y\theta^{(k)}\right|}{\left|\alpha^{(k)}\right| \left|\zeta^{(k)}\right| \left|\pi_1^{(k)}\right|^{n_1} \cdots \left|\pi_v^{(k)}\right|^{n_v}} \\ &> \frac{e^{-c_{11}A}}{\max\limits_{1 \le i \le n} \left|\alpha^{(i)} \zeta^{(i)}\right| \max\limits_{1 \le i \le n} \left|\pi_1^{(i)} \cdots \pi_v^{(i)}\right|^N} \\ &= \exp\left(-c_{11}A - c_8'' - c_9'' N\right), \end{aligned}$$

whence

$$A < \frac{c_8'' + c_9'' N}{c_{10} - c_{11}}.$$

By (8) and (17),

$$\left|x - y\theta^{(k)}\right| = |a| \, p_1^{z_1} \cdots p_v^{z_v} \prod_{i \ne k} \left|x - y\theta^{(i)}\right|^{-1} < |a| \, p_1^{n_1 h_1 + s_1 + t_1} \cdots p_v^{n_v h_v + s_v + t_v} e^{(n-1)c_{11}}.$$

So $\left|\varepsilon^{(k)}\right| \geq e^{c_{10}A}$ implies

$$\begin{aligned} e^{c_{10}A} &\leq \frac{\left|x - y\theta^{(k)}\right|}{\left|\alpha^{(k)}\right|\left|\zeta^{(k)}\right|\left|\pi_1^{(k)}\right|^{n_1}\cdots\left|\pi_v^{(k)}\right|^{n_v}} \\ &< \frac{|a|\, p_1^{s_1+t_1}\cdots p_v^{s_v+t_v}}{\left|\alpha^{(k)}\right|\left|\zeta^{(k)}\right|} \cdot \frac{p_1^{n_1h_1}\cdots p_v^{n_vh_v}}{\left|\pi_1^{(k)}\right|^{n_1}\cdots\left|\pi_v^{(k)}\right|^{n_v}} \cdot e^{(n-1)c_{11}A} \\ &< \frac{|a|\, p_1^{s_1+t_1}\cdots p_v^{s_v+t_v}}{\min\limits_{1\leq i\leq n}\left|\alpha^{(i)}\zeta^{(i)}\right|} \cdot \left(\frac{p_1^{h_1}\cdots p_v^{h_v}}{\min\limits_{1\leq i\leq n}\left|\pi_1^{(i)}\cdots\pi_v^{(i)}\right|}\right)^N \cdot e^{(n-1)c_{11}A} \\ &= \exp\left(c_8' + c_9'N + (n-1)c_{11}A\right), \end{aligned}$$

from which we deduce

$$A < \frac{c_8' + c_9'N}{c_{10} - (n-1)c_{11}}.$$

□

Put

$$\begin{aligned} c_{12}' &= \frac{c_8' + c_5c_9'}{c_{10} - (n-1)c_{11}}, & c_{13}' &= \max\left\{\frac{c_4c_9'}{c_{10} - (n-1)c_{11}}, e^2\right\}, \\ c_{12}'' &= \frac{c_8'' + c_5c_9''}{c_{10} - c_{11}}, & c_{13}'' &= \max\left\{\frac{c_4c_9''}{c_{10} - c_{11}}, e^2\right\} \end{aligned}$$

Lemma 14.2. If $\min\limits_{1\leq i\leq n}\left|x - y\theta^{(i)}\right| > e^{-c_{11}A}$ and $H = A$, then

$$H \leq c_{14},$$

where

$$c_{14} = \max\left\{\left\lceil 2(c_{12}' + c_{13}'\log c_{13}')\right\rceil - 1, \left\lceil 2(c_{12}'' + c_{13}''\log c_{13}'')\right\rceil - 1, 2\right\}.$$

Proof. Assume $H > 2$. Then Lemmas 13.2 and 14.1 imply that we have either

$$A < \frac{c_8' + c_5c_9'}{c_{10} - (n-1)c_{11}} + \frac{c_4c_9'}{c_{10} - (n-1)c_{11}}\log H$$

or

$$A < \frac{c_8'' + c_5c_9''}{c_{10} - c_{11}} + \frac{c_4c_9''}{c_{10} - c_{11}}\log H.$$

Since $H = A$, Lemma 10.1 and the first inequality imply $H < 2(c_{12}' + c_{13}'\log c_{13}')$, while Lemma 10.1 and the second inequality yield $H < 2(c_{12}'' + c_{13}''\log c_{13}'')$. □

Note that (except for the requirement $0 < c_{11} < c_{10}/(n-1)$) c_{11} is a free parameter that we will use later to optimize bounds that depend on it.

15 A bound for A when $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right|$ is small

In Section 14, we established an upper bound for A in terms of $\log H$ in the case $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right| > e^{-c_{11}A}$. In this section, we will prove an analogous an estimate when $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right| \le e^{-c_{11}A}$. Unless $s=0$, we will need to use the theory of linear forms in real/complex logarithms.

Put

$$c_{15} = \begin{cases} \max\left\{\left\lfloor -\dfrac{1}{c_{11}}\log \min\limits_{s+1\le j\le s+t}\left|\operatorname{Im}\theta^{(j)}\right|\right\rfloor, 0\right\} & \text{if } t>0, \\ 0 & \text{if } t=0. \end{cases}$$

Lemma 15.1. If $t>0$, $i\in\{s+1,\dots,s+2t\}$, and $\left|x-y\theta^{(i)}\right| \le e^{-c_{11}A}$, then

$$A \le c_{15}.$$

Proof. If $y=0$, we have $x=\pm 1$ (since $(x,y)=1$), and it follows from (16) that $A=0$. If $|y|\ge 1$, then

$$e^{-c_{11}A} \ge \left|x-y\theta^{(i)}\right| \ge \left|\operatorname{Im}\theta^{(i)}\right| \ge \min_{s+1\le j\le s+t}\left|\operatorname{Im}\theta^{(j)}\right|,$$

whence $A\le c_{15}$. □

The following lemma takes care of the $s=0$ case.

Lemma 15.2. If $\min_{1\le i\le n}\left|x-y\theta^{(i)}\right| \le e^{-c_{11}A}$ and $s=0$, then

$$A \le c_{15}.$$

Proof. Immediate from Lemma 15.1 □

For the $s>0$ case, we will work with the indices $i_0, j, k \in \{1,\dots,n\}$ from the $p=\infty$ case of Section 6. Our method for computing an upper bound for A requires that we choose i_0 to be an index satisfying $\left|x-y\theta^{(i_0)}\right| = \min_{1\le i\le n}\left|x-y\theta^{(i)}\right|$. As this condition depends on (x,y), we cannot make a definite choice for i_0 without actually solving (16). We can overcome this difficulty by computing a bound for A for each of the cases $i_0=1,\dots,n$ and then taking the largest of the bounds obtained. In fact, since Lemma 15.1 yields an upper bound for A when $i_0\in\{s+1,\dots,s+2t\}$, $t>0$, we only need to concern ourselves with the cases $i_0=1,\dots,s$.

For each $i_0 \in \{1, \ldots, s\}$, we define numbers $j = j(i_0)$, $k = k(i_0)$, $\Lambda_0 = \Lambda_0(i_0)$, $c_{16} = c_{16}(i_0)$, $c_{17} = c_{17}(i_0)$, $c_{18} = c_{18}(i_0)$, and (in the $s = 1, 2$ case) $a_0 = a_0(i_0)$ as follows. We separate the two cases $s \geq 3$ and $s = 1, 2$. These are called, respectively, *the real case* and *the complex case*. In both cases, j, k will be indices in $\{1, \ldots, n\}$ such that $i_0 \neq j \neq k \neq i_0$. If $s \geq 3$, we stipulate that $j, k \in \{1, \ldots, s\}$ in order to have $\theta^{(i_0)}, \theta^{(j)}, \theta^{(k)} \in \mathbb{R}$. If $s = 1, 2$, we require that $j \in \{s+1, \ldots, s+t\}$ and $k = j + t$ so that $\theta^{(j)} = \overline{\theta^{(k)}}$. Subject to these restrictions, we fix a choice for j, k that minimizes the number

$$c_{16} = \frac{2}{\left|\theta^{(i_0)} - \theta^{(j)}\right|} \left| \frac{\theta^{(j)} - \theta^{(k)}}{\theta^{(k)} - \theta^{(i_0)}} \right|.$$

In the real case, put

$$\Lambda_0 = \log(|1 + \lambda|) = \log\left|\delta_1\right| + \sum_{i=1}^{v} n_i \log \left| \frac{\pi_i^{(k)}}{\pi_i^{(j)}} \right| + \sum_{i=1}^{r} a_i \log \left| \frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}} \right|.$$

By the choice of j, k, the numbers $1 + \lambda$, δ_1, $\pi_i^{(k)}/\pi_i^{(j)}$ $(i = 1, \ldots, v)$, $\varepsilon_i^{(k)}/\varepsilon_i^{(j)}$ $(i = 1, \ldots, r)$ are all real, so taking their absolute values just multiplies them by ± 1. Let F be the splitting field of g over $\mathbb{Q}$ and let σ be an embedding of F into $\mathbb{C}$. Put

$$\alpha_1 = \sigma^{-1}(|\delta_1|), \quad \alpha_{1+i} = \sigma^{-1}\left(\left| \frac{\pi_i^{(k)}}{\pi_i^{(j)}} \right|\right) \quad (i = 1, \ldots, v),$$
$$\alpha_{1+v+i} = \sigma^{-1}\left(\left| \frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}} \right|\right) \quad (i = 1, \ldots, r).$$

As with the α_i from the proof of Lemma 13.1, the α_i here are straightforward to calculate. Set $L = \mathbb{Q}(\alpha_1, \ldots, \alpha_{1+v+r})$. We are now in the situation of Section 12 (with L embedded into $\mathbb{C}$ via σ and $m = 1 + v + r$), and we can compute the constant c_{17} appearing there. Note that the value of c_{17} depends on the embedding σ chosen only through the number κ. Note also that Λ_0 is equal to Λ from Section 12 (with the obvious identifications for the b_i). So, according to Theorem 12.1,

$$|\Lambda_0| > \exp(-c_{17} \log H - c_{17}) \tag{23}$$

if $\Lambda_0 \neq 0$.

In the complex case, define

$$\Lambda_0 = \frac{1}{i}\mathrm{Log}(1 + \lambda).$$

Here $\mathrm{Log}(z)$ denotes the principal branch of the logarithm of z, so $\mathrm{Re}(\mathrm{Log}(z)) = \log|z|$ and $\mathrm{Im}(\mathrm{Log}(z)) \in (-\pi, \pi]$. We have

$$\begin{aligned} i\Lambda_0 = \mathrm{Log}(1+\lambda) &= \mathrm{Log}\left(\frac{\theta^{(i_0)}-\theta^{(j)}}{\theta^{(i_0)}-\theta^{(k)}}\frac{x-y\theta^{(k)}}{x-y\theta^{(j)}}\right) \\ &= \mathrm{Log}\,(\delta_1) + \sum_{i=1}^{v} n_i \mathrm{Log}\left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right) + \sum_{i=1}^{r} a_i \mathrm{Log}\left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right) + 2\pi i a_0 \\ &= \mathrm{Log}\left(\frac{\theta^{(i_0)}-\theta^{(j)}}{\theta^{(i_0)}-\theta^{(k)}}\frac{\alpha^{(k)}}{\alpha^{(j)}}\frac{\zeta^{(k)}}{\zeta^{(j)}}\right) + \sum_{i=1}^{v} n_i \mathrm{Log}\left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right) + \sum_{i=1}^{r} a_i \mathrm{Log}\left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right) + 2\pi i a_0 \end{aligned}$$

for some integer a_0. As $\theta^{(i_0)} \in \mathbb{R}$ and $\theta^{(j)} = \overline{\theta^{(k)}}$, each logarithm above has real part equal to zero and, therefore, modulus less than or equal to π. So by the triangle inequality

$$2\pi\,|a_0| \le 2\pi + \sum_{i=1}^{v} n_i\pi + \sum_{i=1}^{r} a_i\pi \le (2+(v+r)H)\pi \le (1+v+r)H\pi \tag{24}$$

for $H \ge 2$. Since $\pi i = \mathrm{Log}(-1)$, $i\Lambda_0$ is a linear form in logarithms of $2+v+r$ complex numbers. So, as in the real case, we can compute the constant c_{17} from Section 12, and we have

$$|\Lambda_0| > \exp(-c_{17}\log(\max\{H, 2\,|a_0|\}) - c_{17}) \ge \exp(-c_{17}\log((1+v+r)H) - c_{17}) \tag{25}$$

if $\Lambda_0 \neq 0$ by Theorem 12.1 and (24).

Finally, put

$$c_{18} = \begin{cases} \log(2(\log 2)c_{16}) + c_{17} & \text{if } s \ge 3, \\ \log(4\sin^{-1}(1/4)c_{16}) + c_{17}(1+\log(1+v+r)) & \text{if } s = 1, 2. \end{cases}$$

Lemma 15.3. Assume $\min\limits_{1\le i\le n}\left|x-y\theta^{(i)}\right| \le e^{-c_{11}A}$, $H \ge 2$, and $s > 0$. For each $i_0 \in \{1,\dots,s\}$, if $\left|x-y\theta^{(i_0)}\right| = \min\limits_{1\le i\le n}\left|x-y\theta^{(i)}\right|$ and

$$A \ge \max\left\{\left\lceil\frac{\log(2c_{16}(i_0))}{c_{11}}\right\rceil, 1\right\},$$

then

$$A < \frac{1}{c_{11}}(c_{18}(i_0) + c_{17}(i_0)\log H).$$

For the proof of Lemma 15.3, we will need the following two simple lemmas.

Lemma 15.4. Let $z \in \mathbb{C}$ and $0 < c < 1$. If $|e^z - 1| \le c$, then

$$|z| \le \frac{-\log(1-c)}{c}|e^z - 1|.$$

Proof.

$$|z| = |\mathrm{Log}(e^z)| = \left|\sum_{i=1}^{\infty} \frac{(-1)^{i-1}(e^z-1)^i}{i}\right| \leq |e^z-1| \sum_{i=1}^{\infty} \frac{c^{i-1}}{i} = \frac{-\log(1-c)}{c}|e^z-1|.$$

□

Lemma 15.5. Let $-\frac{\pi}{2} < z \leq \frac{\pi}{2}$ and $0 < c < 1$. If $|\sin z| \leq c$, then

$$|z| \leq \frac{\sin^{-1}(c)}{c}|\sin z|.$$

Proof. The inequality $-c \leq \sin z \leq c$ implies $-\sin^{-1}(c) \leq z \leq \sin^{-1}(c)$. So, since the function $u \mapsto u/\sin u$ has even symmetry and is increasing on $(0, \sin^{-1}(c)]$, it follows that

$$\frac{|z|}{|\sin z|} \leq \frac{\sin^{-1}(c)}{\sin(\sin^{-1}(c))} = \frac{\sin^{-1}(c)}{c}$$

when $z \neq 0$. □

Proof of Lemma 15.3. Let $i_0 \in \{1, \ldots, s\}$, and assume $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} |x - y\theta^{(i)}|$ and $A \geq \max\{\lceil \log(2c_{16})/c_{11} \rceil, 1\}$. Since $A > 0$, we have $|y| \geq 1$ (by the first sentence in the proof of Lemma 15.1). Therefore,

$$\left|\theta^{(i_0)} - \theta^{(j)}\right| \leq \left|y(\theta^{(i_0)} - \theta^{(j)})\right| = \left|(x - y\theta^{(i_0)}) - (x - y\theta^{(j)})\right| \leq 2\left|x - y\theta^{(j)}\right|.$$

So by (18)

$$|\lambda| = \left|\frac{x - y\theta^{(i_0)}}{x - y\theta^{(j)}}\right| \cdot \left|\frac{\theta^{(j)} - \theta^{(k)}}{\theta^{(k)} - \theta^{(i_0)}}\right| \leq c_{16}\left|x - y\theta^{(i_0)}\right| \leq c_{16}e^{-c_{11}A}.$$

Since $A \geq \log(2c_{16})/c_{11}$, it follows that $|\lambda| \leq \frac{1}{2}$. By (18), we also have $\lambda \neq 0$.

In the real case, $\lambda \in \mathbb{R}$. So, since $|\lambda| \leq \frac{1}{2}$, we have $0 < 1 + \lambda$. Then $\left|e^{\Lambda_0} - 1\right| = |\lambda| \leq \frac{1}{2}$ and, therefore, by Lemma 15.4 (with $z = \Lambda_0$ and $c = \frac{1}{2}$),

$$|\Lambda_0| \leq \frac{-\log\left(1 - \frac{1}{2}\right)}{\frac{1}{2}}\left|e^{\Lambda_0} - 1\right| = 2(\log 2)\,|\lambda| \leq 2(\log 2)c_{16}e^{-c_{11}A}.$$

Additionally, since $0 < 1 + \lambda$ and $\lambda \neq 0$, we have $\Lambda_0 \neq 0$, and so

$$|\Lambda_0| > \exp(-c_{17}\log H - c_{17}).$$

by (23). Combining these bounds for $|\Lambda_0|$ yields

$$c_{11}A \leq \log(2(\log 2)c_{16}) + c_{17} + c_{17}\log H = c_{18} + c_{17}\log H.$$

In the complex case, we have $\Lambda_0 \in (-\pi, \pi]$ and $2\left|\sin\left(\frac{1}{2}\Lambda_0\right)\right| = \left|e^{i\Lambda_0} - 1\right| = |\lambda| \leq \frac{1}{2}$. Applying Lemma 15.5 with $z = \frac{1}{2}\Lambda_0$ and $c = \frac{1}{4}$ then yields

$$|\Lambda_0| \leq 2 \cdot \frac{\sin^{-1}(1/4)}{1/4}\left|\sin\left(\frac{1}{2}\Lambda_0\right)\right| = 4\sin^{-1}(1/4)\,|\lambda| \leq 4\sin^{-1}(1/4)c_{16}e^{-c_{11}A}.$$

On the other hand, since $\lambda \neq 0$, we have $\Lambda_0 \neq 0$. Thus, by (25),

$$|\Lambda_0| > \exp(-c_{17}\log((1+v+r)H) - c_{17}).$$

Putting the upper and lower bounds for $|\Lambda_0|$ together leads to

$$c_{11}A \leq c_{18} + c_{17}\log H.$$

□

Put

$$c_{19} = \max_{1\leq i_0\leq s}\left\lceil\frac{\log(2c_{16}(i_0))}{c_{11}}\right\rceil - 1,$$
$$c_{20} = \left\lceil 2\max_{1\leq i_0\leq s}\left(\frac{c_{18}(i_0)}{c_{11}} + \max\left\{\frac{c_{17}(i_0)}{c_{11}}, e^2\right\}\log\left(\max\left\{\frac{c_{17}(i_0)}{c_{11}}, e^2\right\}\right)\right)\right\rceil - 1.$$

Lemma 15.6. If $\min_{1\leq i\leq n}\left|x - y\theta^{(i)}\right| \leq e^{-c_{11}A}$, $s > 0$, and $H = A$, then

$$A \leq c_{21} = \max\{c_{15}, c_{19}, c_{20}, 1\}$$

Proof. Assume $A > \max\{c_{15}, c_{19}, 1\}$, and let $i_0 \in \{1, \ldots, n\}$ be an index such that $|x - y\theta^{(i_0)}| = \min_{1\leq i\leq n}|x - y\theta^{(i)}|$. By Lemma 15.1, we know that $i_0 \in \{1, \ldots, s\}$. So, since $H = A \geq \max\{\lceil\log(2c_{16}(i_0))/c_{11}\rceil, 2\}$, Lemma 15.3 implies

$$A < \frac{c_{18}(i_0)}{c_{11}} + \frac{c_{17}(i_0)}{c_{11}}\log A.$$

It then follows from Lemma 10.1 that

$$A < 2\left(\frac{c_{18}(i_0)}{c_{11}} + \max\left\{\frac{c_{17}(i_0)}{c_{11}}, e^2\right\}\log\left(\max\left\{\frac{c_{17}(i_0)}{c_{11}}, e^2\right\}\right)\right),$$

whence $A \leq c_{20}$. □

16 A bound for H

By combining Lemma 13.3, Lemma 14.2, and (depending on the value of s) either Lemma 15.2 or Lemma 15.6, we obtain the following explicit and unconditional upper bound on H.

Theorem 16.1.

$$H \leq c_{22} = \begin{cases} \max\{c_7, c_{14}, c_{15}\} & \text{if } s = 0 \\ \max\{c_7, c_{14}, c_{21}\} & \text{if } s > 0 \end{cases}$$

The constants c_{14}, c_{15}, c_{21} depend on the parameter c_{11}, which we are free to choose so that the upper bound c_{22} is optimized (subject to the condition $0 < c_{11} < c_{10}/(n-1)$). In practice, the constants $c_1(l)$, c_4, and c_{17}, which arise directly from the estimates for linear forms in logarithms, will be very large. The same goes for all constants depending on $c_1(l)$, c_4, and c_{17}. All the other constants will be comparatively small. If $s = 0$, c_{22} will therefore be essentially of the size

$$\max\left\{c_4, \frac{c_4}{c_{10} - (n-1)c_{11}}\right\},$$

and an optimal choice for c_{11} will be near $(c_{10} - 1)/(n-1)$. In case $s > 0$, c_{22} will be of the size

$$\max\left\{c_4, \frac{c_4}{c_{10} - (n-1)c_{11}}, c_{17}/c_{11}\right\}.$$

So if $c_4 \gg c_{17}$, an optimal choice for c_{11} will be roughly c_{17}/c_4. If $c_4 \ll c_{17}$, an optimal choice for c_{11} will be of the size $(c_{10} - 1)/(n-1)$. And if $c_4 \approx c_{17}$, the size of the optimal c_{11} will be somewhere between the size of c_{17}/c_4 and the size of $(c_{10} - 1)/(n-1)$.

Corollary 16.2. For each $l \in \{1, \ldots, v\}$,

$$n_l \leq \min\left\{c_{22}, \max\left\{\left\lceil \frac{c_1(l)}{h_l} \log c_{22} - \frac{1}{h_l}\mathrm{ord}_{p_l}(\delta_2)\right\rceil - 1, 2\right\}\right\}$$

Proof. If $n_l > 2$, then $H > 2$, so Lemma 13.1 and Theorem 16.1 imply

$$n_l \leq \left\lceil \frac{c_1(l)}{h_l} \log H - \frac{1}{h_l}\mathrm{ord}_{p_l}(\delta_2)\right\rceil - 1 \leq \left\lceil \frac{c_1(l)}{h_l} \log c_{22} - \frac{1}{h_l}\mathrm{ord}_{p_l}(\delta_2)\right\rceil - 1.$$

□

Evidently, one should use Corollary 16.2 rather than Theorem 16.1 to bound n_l when $l \in I$.

17 The reduction strategy

The upper bounds on $A = \max\{|a_1|, \ldots, |a_r|\}$ and the n_l furnished by Theorem 16.1 and its corollary are expected to be very large. Indeed, bounds of the size 10^{40} are typical even when the degree n and the number v of primes are both small. Enumeration of the solutions of (16) by a naive search at this stage would be virtually impossible. In order to make a search feasible, we need to reduce the upper bounds on A and the n_l considerably. In this section, we outline our strategy for doing just that.

Our strategy has several steps, and the best known upper bounds on A and the n_l will change from step to step. In each step, N_l will denote the current best upper bound for n_l $(l = 1, \ldots, v)$, and A_0 will denote the current best upper bound for A. We will also use H_0 to denote the best known upper bound for $H = \max\{n_1, \ldots, n_v, |a_1|, \ldots, |a_r|\}$ in a given step. So $H_0 = \max\{N_1, \ldots, N_v, A_0\}$. We will use the b_i notation (see Section 8) frequently in Sections 19-23. It will therefore be convenient to let B_i stand for the current best upper bound for $|b_i|$ $(i = 1, \ldots, 1 + v + r)$. We will never bound the a_i individually, so we will always have

$$B_1 = 1, \quad B_{1+i} = N_i \quad (i = 1, \ldots, v), \quad B_{1+v+i} = A_0 \quad (i = 1, \ldots, r).$$

As we remarked in Section 9, for each index $l \in \{1, \ldots, v\}$ for which Lemma 8.3 gives an upper bound for n_l, there is no need to reduce that upper bound further. Recall that I denotes the set of all indices l in $\{1, \ldots, v\}$ to which Lemma 8.3 does not apply. There are four types of reduction procedures involved in the strategy. We refer to them as 'basic p-adic reduction', 'basic real reduction', 'refined p-adic reduction', and 'refined real reduction'.

Now for the strategy itself. First, for each $l \in I$ in turn, we perform the basic p_l-adic reduction procedure to (ideally) reduce N_l to the size of $\log H_0$. Next we perform the basic real reduction procedure to reduce A_0. If everything goes as planned, the new A_0 will be the size of $\log H_0$ also. We repeat the steps above (p_i-adics, then real) until the bounds stop improving. Then we move onto the refined reduction procedures. We start by performing the refined p_l-adic reduction procedure for each $l \in I$ to reduce the bounds N_l. Then we perform the refined real reduction procedure to improve A_0. As with the basic procedures, we expect a reduction to something like $\log H_0$ in each step, and we repeat until we stop making progress with the bounds.

18 Lattice basis reduction

The four types of reduction procedures involved in our reduction strategy all rely on Diophantine approximation techniques involving lattice basis reduction. In this section, we present those aspects of the theory of lattice basis reduction that we will need.

An n-dimensional lattice is a subset of $\mathbb{R}^n$ of the form

$$\Gamma = \left\{ \sum_{i=1}^{n} x_i \mathbf{c}_i : x_i \in \mathbb{Z} \right\},$$

where $\mathbf{c}_1, \ldots, \mathbf{c}_n$ are vectors forming a basis for $\mathbb{R}^n$. We say that the vectors $\mathbf{c}_1, \ldots, \mathbf{c}_n$ form a basis for Γ, or that they generate Γ. The following basic fact about lattice bases will be useful: If A and B are invertible $n \times n$ real matrices, then the lattice generated by the columns of A is equal to the lattice generated by the columns of B if and only if there is a unimodular integer matrix U such that $AU = B$.

Let Γ be a lattice with basis $\mathbf{c}_1, \ldots, \mathbf{c}_n$. Define vectors $\mathbf{c}_i^*$ $(i = 1, \ldots, n)$ and real numbers μ_{ij} $(1 \leq j < i \leq n)$ inductively by

$$\mathbf{c}_i^* = \mathbf{c}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{c}_j^*, \qquad \mu_{ij} = \frac{\langle \mathbf{c}_i, \mathbf{c}_j^* \rangle}{\langle \mathbf{c}_j^*, \mathbf{c}_j^* \rangle}$$

(this is just the Gram-Schmidt process). The basis $\mathbf{c}_1, \ldots, \mathbf{c}_n$ is called LLL-reduced if

$$|\mu_{ij}| \leq \frac{1}{2} \quad \text{for } 1 \leq j < i \leq n,$$
$$\frac{3}{4} \left| \mathbf{c}_{i-1}^* \right|^2 \leq \left| \mathbf{c}_i^* + \mu_{ii-1} \mathbf{c}_{i-1}^* \right|^2 \quad \text{for } 1 < i \leq n.$$

These properties imply that an LLL-reduced basis is approximately orthogonal and that, generically, its constituent vectors are of roughly the same length. Every n-dimensional lattice has an LLL-reduced basis and such a basis can be computed very quickly using the so-called LLL algorithm (see [LLL]). The LLL algorithm takes as input an arbitrary basis for a lattice and outputs an LLL-reduced basis for the lattice. The algorithm is typically modified to additionally output a unimodular integer matrix U such that $B = AU$, where A is the matrix whose column-vectors are the input basis and B is the matrix whose column-vectors are the LLL-reduced output basis. Several versions of this algorithm are implemented in Magma, including de Weger's exact integer version (see [dW1] (Section 3.5), [Sm] (Section V.4), or [Coh1] (Section 2.6.3)) and Nguyen and Stehlé's exact floating

point version (see [NS], [St]). Note that Nguyen and Stehlé's algorithm requires a slight modification of the notion of LLL-reduced given above.

For Γ an n-dimension lattice and $\mathbf{y}$ a vector in $\mathbb{R}^n$, we define

$$l(\Gamma, \mathbf{y}) = \min_{\mathbf{x} \in \Gamma \setminus \{\mathbf{y}\}} |\mathbf{x} - \mathbf{y}| .$$

The most important property of an LLL-reduced basis for us is the following lemma.

Lemma 18.1. Let Γ be a lattice with LLL-reduced basis $\mathbf{c}_1, \ldots, \mathbf{c}_n$ and let $\mathbf{y}$ be a vector in $\mathbb{R}^n$.

(a) If $\mathbf{y} = \mathbf{0}$, then $l(\Gamma, \mathbf{y}) \geq 2^{-(n-1)/2} |\mathbf{c}_1|$.

(b) Assume $\mathbf{y} \notin \Gamma$. Then $\mathbf{y} = s_1 \mathbf{c}_1 + \cdots s_n \mathbf{c}_n$ for some $s_1, \ldots, s_n \in \mathbb{R}$ with not all $s_i \in \mathbb{Z}$. Put $J = \{j \in \{1, \ldots, n\} : s_j \notin \mathbb{Z}\}$. For each $j \in J$, set $\delta(j) = \max_{i>j} \|s_i\| |\mathbf{c}_i|$ if $j < n$, and set $\delta(j) = 0$ if $j = n$. Here $\|\cdot\|$ denotes the distance to the nearest integer. We have

$$l(\Gamma, \mathbf{y}) \geq \max_{j \in J} \left(2^{-(n-1)/2} \|s_j\| |\mathbf{c}_1| - (n-j)\delta(j) \right) .$$

Lemma 18.1(a) is Proposition 1.11 in [LLL]; proofs can be found in [LLL], [dW1] (Section 3.4), or [Sm] (Section V.3). Lemma 18.1(b) is a combination of Lemmas 3.5 and 3.6 in [dW1].

In the refined reduction procedures, we will need to compute all the vectors in a given lattice which have norm less than or equal to some constant. This can be done efficiently using an algorithm of Fincke-Pohst (cf. [FP], [Coh1]). A version of this algorithm with some improvements due to Stehlé is implemented in Magma. A function to estimate the computational cost of this algorithm is also available.

19 Preliminaries for the p_l-adic reduction procedure

In this section, we set some notation and give some preliminaries for the p_l-adic reduction procedures. We will refer to the notation of Section 8. Also, recall from Section 13 that I is the set of all indices $l \in \{1, \dots, v\}$ for which Lemma 8.3 does not furnish an upper bound for n_l.

Consider a fixed $l \in I$. We have

$$\mathrm{ord}_{p_l}(\alpha_1) \geq \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_{p_l}(\alpha_i), \quad \mathrm{ord}_p(\alpha_{1h}) \geq \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_p(\alpha_{ih}) \quad (h = 1, \dots, s). \tag{26}$$

Let I' be the set of all indices $i' \in \{2, \dots, 1+v+r\}$ for which

$$\mathrm{ord}_{p_l}(\alpha_{i'}) = \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_{p_l}(\alpha_i).$$

There are two cases, *the special case* and *the general case*. The special case is when there is some index $i' \in I'$ such that $\alpha_i/\alpha_{i'} \in \mathbb{Q}_{p_l}$ for $i = 1, \dots, 1+v+r$. The general case is when there is no such index.

In the special case, we let $\hat{i}$ be an arbitrary index in I' for which $\alpha_i/\alpha_{\hat{i}} \in \mathbb{Q}_{p_l}$ for every $i \in \{1, \dots, 1+v+r\}$. We further define

$$\beta_i = -\frac{\alpha_i}{\alpha_{\hat{i}}} \qquad (i = 1, \dots, 1+v+r),$$

and

$$\Lambda_l' = \frac{1}{\alpha_{\hat{i}}} \Lambda_l = \sum_{i=1}^{1+v+r} b_i(-\beta_i).$$

In the general case, we first fix an $h \in \{1, \dots, S\}$ arbitrarily. Then we let $\hat{i}$ be an index in $\{2, \dots, 1+v+r\}$ such that

$$\mathrm{ord}_p(\alpha_{\hat{i}h}) = \min_{2 \leq i \leq 1+v+r} \mathrm{ord}_p(\alpha_{ih}),$$

and define

$$\beta_i = -\frac{\alpha_{ih}}{\alpha_{\hat{i}h}} \qquad (i = 1, \dots, 1+v+r),$$

and

$$\Lambda_l' = \frac{1}{\alpha_{\hat{i}h}} \Lambda_{lh} = \sum_{i=1}^{1+v+r} b_i(-\beta_i).$$

Note that in both cases we have $\beta_i \in \mathbb{Z}_{p_l}$ for $i = 1, \dots, 1+v+r$. This is clear for $i = 2, \dots, 1+v+r$. For $i = 1$, it follows from (26).

Lemma 19.1. Suppose

$$n_l > \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right).$$

In the special case, we have

$$\mathrm{ord}_{p_l}(\Lambda_l') = n_l h_l + d_l$$

with

$$d_l = \mathrm{ord}_{p_l}(\delta_2) - \mathrm{ord}_p(\alpha_{\hat{i}}).$$

In the general case, we have

$$\mathrm{ord}_{p_l}(\Lambda_l') \geq n_l h_l + d_l$$

with

$$d_l = \mathrm{ord}_{p_l}(\delta_2) - u_l - \mathrm{ord}_p(\alpha_{\hat{i}h}).$$

Proof. Immediate from Lemmas 8.1 and 8.2, and the definition of Λ_l'. □

Now we state several relatively-easy-to-check conditions that each imply that we are in the special case. In fact, each condition implies that we have $\alpha_{i_1}/\alpha_{i_2} \in \mathbb{Q}_{p_l}$ for every $i_1, i_2 \in \{1, \ldots, 1 + v + r\}$. Note that these conditions correspond to the guidelines 3-6 for choosing j and k in Section 9.

(a) $\alpha_1, \ldots, \alpha_{1+v+r}$ belong to $\mathbb{Q}_{p_l}$.

(b) $g(t)$ has three or more linear factors in $\mathbb{Q}_{p_l}[t]$, and $\theta^{(i_0)}, \theta^{(j)}, \theta^{(k)}$ are roots of such polynomials.

(c) $g(t)$ has an irreducible factor in $\mathbb{Q}_{p_l}[t]$ of degree two, and $\theta^{(j)}, \theta^{(k)}$ are the roots of this factor.

(d) $g(t)$ has a nonlinear irreducible factor in $\mathbb{Q}_{p_l}[t]$ that splits completely in the extension of $\mathbb{Q}_{p_l}$ that it generates, and $\theta^{(j)}, \theta^{(k)}$ are roots of this factor.

It is obvious that (a) implies $\alpha_{i_1}/\alpha_{i_2} \in \mathbb{Q}_{p_l}$ for every $i_1, i_2 \in \{1, \ldots, 1 + v + r\}$. If (b) holds, then δ_1, $\pi_i^{(k)}/\pi_i^{(j)}$ $(i = 1, \ldots, v)$, $\varepsilon_i^{(k)}/\varepsilon_i^{(j)}$ $(i = 1, \ldots, r)$ all belong to $\mathbb{Q}_{p_l}$, which, since $\mathbb{Q}_{p_l}$ is complete, implies (a). It is clear that (c) implies (d). We claim that (d) implies $\alpha_{i_1}/\alpha_{i_2} \in \mathbb{Q}_{p_l}$ for every $i_1, i_2 \in \{1, \ldots, 1 + v + r\}$. To see that the claim holds, assume (d), let L be the extension of $\mathbb{Q}_{p_l}$ generated by the factor of $g(t)$ in question, and consider

any $\alpha, \beta \in L$. The automorphism of L that maps $\theta^{(j)}$ to $\theta^{(k)}$ multiplies the logarithms $\log_{p_l}\left(\alpha^{(k)}/\alpha^{(j)}\right)$ and $\log_{p_l}\left(\beta^{(k)}/\beta^{(j)}\right)$ by -1 and hence fixes the quotient

$$\frac{\log_{p_l}\left(\alpha^{(k)}/\alpha^{(j)}\right)}{\log_{p_l}\left(\beta^{(k)}/\beta^{(j)}\right)}. \tag{27}$$

Therefore, since L is Galois, this quotient belongs to $\mathbb{Q}_{p_l}$. Since $\alpha_{i_1}/\alpha_{i_2}$ is of the form (27) for every $i_1, i_2 \in \{1, \ldots, 1+v+r\}$, the claim is proved.

When performing the computations associated with the refined reduction procedures (Section 22 and Section 23), we will encounter the situation where we have computed $\alpha_1, \ldots, \alpha_{1+v+r}$ and $\beta_1, \ldots, \beta_{1+v+r}$, we have explicit integers $b_1, \ldots, b_{1+v+r}$, and we wish to verify the conclusions of Lemma 8.2 and Lemma 19.1. Moreover, we will encounter this situation a large number of times, so it is desirable to perform the verifications as efficiently as possible. In the special case, the conclusions of Lemmas 8.2 and 19.1 are equivalent. Thus we can save on computation time by checking the conclusion for just one of the two lemmas. This explains why we bother distinguishing the special case from the general case. In both the special and general cases, verifying the conclusion of Lemma 8.2 requires us to compute Λ_l, which typically involves working in a nontrivial extension of $\mathbb{Q}_{p_l}$. Verifying the conclusion of Lemma 19.1 necessitates computing Λ_l', which only involves calculations in $\mathbb{Z}_{p_l}$. Hence, in the special case, it is preferable to skip checking Lemma 8.2 in favour of Lemma 19.1. In the general case, while we cannot simply skip checking one of the two lemmas, we can discuss which lemma we should check first. There are two competing factors to consider. The first factor is that the situation where we will need to verify the conclusions of the two lemmas will be such that if we find that the conclusion of one of the two lemmas does not hold, then we don't need to check the conclusion of the other lemma. So, since the conclusion of Lemma 8.2 implies the conclusion of Lemma 19.1, and not conversely, we have reason to favour checking Lemma 8.2 before Lemma 19.1. The second factor is that, as we noted above, it is computationally cheaper to check the conclusion of Lemma 19.1 than to check the conclusion of 8.2. This second factor suggests that we should check Lemma 19.1 before Lemma 8.2. In our experience, we have found it slightly preferable (from a speed perspective) to test Lemma 19.1 before Lemma 8.2, indicating that the second factor may more important than the first. However, it is not clear which order of testing the lemmas (if any) is optimal in general.

20 Basic p_l-adic reduction

Here we describe the basic p_l-adic reduction procedure.

Fix $l \in I$. We start with some preliminaries. Let $m, W_2, \ldots, W_{1+v+r}$ denote positive integers. These are parameters that we will need to balance in order to obtain a good reduction of the upper bound for n_l. We will discuss how to choose them later in this section. For each $x \in \mathbb{Z}_{p_l}$, $x^{\{m\}}$ will stand for the unique rational integer in $[0, p_l^m - 1]$ such that $\mathrm{ord}_{p_l}(x - x^{\{m\}}) \geq m$ (i.e., $x \equiv x^{\{m\}} \pmod{p_l^m}$). Let Γ_m be the $(v+r)$-dimensional lattice generated by the column-vectors of the matrix

$$A_m = \begin{bmatrix} W_2 & & & & & & \\ & \ddots & & & 0 & & \\ & & W_{\hat{i}-1} & & & & \\ & 0 & & W_{\hat{i}+1} & & & \\ & & & & \ddots & & \\ & & & & & W_{1+v+r} & \\ W_{\hat{i}}\beta_2^{\{m\}} & \cdots & W_{\hat{i}}\beta_{\hat{i}-1}^{\{m\}} & W_{\hat{i}}\beta_{\hat{i}+1}^{\{m\}} & \cdots & W_{\hat{i}}\beta_{1+v+r}^{\{m\}} & W_{\hat{i}}p_l^m \end{bmatrix}.$$

Put

$$\lambda = \frac{1}{p_l^m} \sum_{i=1}^{1+v+r} b_i \left(-\beta_i^{\{m\}}\right)$$

and

$$\mathbf{y} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -W_{\hat{i}}\beta_1^{\{m\}} \end{bmatrix} \in \mathbb{Z}^{v+r}.$$

Evidently, we must compute the β_i to p_l-adic precision at least m in order to avoid errors here. We observe that $\mathbf{y} \in \Gamma_m$ if and only if $\mathbf{y} = \mathbf{0}$. To see that this is true, note that $y \in \Gamma_m$ means there are integers $z_1, \ldots, z_{v+r}$ such that $\mathbf{y} = A_m[z_1, \ldots, z_{v+r}]^T$. The last equation forces $z_1 = \cdots = z_{v+r-1} = 0$ and $-\beta_1^{\{m\}} = z_{v+r}p_l^m$. Since $\beta_1^{\{m\}} \in [0, p_l^m - 1]$, we must then have $z_{v+r} = 0$ also. Hence $\mathbf{y} = \mathbf{0}$. Our observation means that Lemma 18.1 is available to estimate $l(\Gamma_m, \mathbf{y})$.

Put

$$Q = \sum_{i=2}^{1+v+r} W_i^2 B_i^2.$$

Lemma 20.1. If $l(\Gamma_m, \mathbf{y}) > Q^{1/2}$, then

$$n_l \leq \max\left\{\left\lfloor \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rfloor, \left\lceil \frac{1}{h_l}(m - d_l)\right\rceil - 1, 0\right\}.$$

Proof. We prove the contrapositive. Assume $n_l > (1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2))$, $n_l \geq (1/h_l)(m - d_l)$, and $n_l > 0$. Consider the vector

$$\mathbf{x} = A_m \begin{bmatrix} b_2 \\ \vdots \\ b_{\hat{i}-1} \\ b_{\hat{i}+1} \\ \vdots \\ b_{1+v+r} \\ \lambda \end{bmatrix} = \begin{bmatrix} W_2 b_2 \\ \vdots \\ W_{\hat{i}-1} b_{\hat{i}-1} \\ W_{\hat{i}+1} b_{\hat{i}+1} \\ \vdots \\ W_{1+v+r} b_{1+v+r} \\ W_{\hat{i}} b_{\hat{i}} \end{bmatrix} + \mathbf{y}.$$

By Lemma 19.1,

$$\mathrm{ord}_{p_l}\left(\sum_{i=1}^{1+v+r} b_i(-\beta_i)\right) = \mathrm{ord}_{p_l}(\Lambda_l') \geq n_l h_l + d_l \geq m.$$

Since $\mathrm{ord}_{p_l}(\beta_i^{\{m\}} - \beta_i) \geq m$ for $i = 1, \ldots, 1 + v + r$, it follows that

$$\mathrm{ord}_{p_l}\left(\sum_{i=1}^{1+v+r} b_i(-\beta_i^{\{m\}})\right) \geq m,$$

which means $\lambda \in \mathbb{Z}$. Hence $\mathbf{x} \in \Gamma_m$. Since $b_{l+1} = n_l > 0$, we cannot have $\mathbf{x} = \mathbf{y}$. Therefore,

$$l(\Gamma_m, \mathbf{y})^2 \leq |\mathbf{x} - \mathbf{y}|^2 = \sum_{i=2}^{1+v+r} W_i^2 b_i^2 \leq \sum_{i=2}^{1+v+r} W_i^2 B_i^2 = Q.$$

□

The reduction procedure works as follows. Taking A_m as input, we first compute an LLL-reduced basis for Γ_m. Then we find a lower bound for $l(\Gamma_m, \mathbf{y})$ using Lemma 18.1. If the lower bound is greater than $Q^{1/2}$, Lemma 20.1 gives a new upper bound for n_l. If not, we should increase m and try the procedure again. In such a situation, we can take advantage of the computations already done by starting with a partially reduced basis for the new lattice Γ_m' ($m' > m$) rather than the basis given by $A_{m'}$. We will have computed, for a certain m, a matrix B_m whose columns form a LLL-reduced basis for Γ_m, and a

unimodular integer matrix U_m such that $A_m U_m = B_m$. We want to compute a reduced basis for $\Gamma_{m'}$. Instead of using $A_{m'}$ as the input basis for the LLL algorithm, we can use the partially reduced basis for $\Gamma_{m'}$ given by the columns of $A_{m'}U_m$. If we find that several increases of m have failed to yield a new upper bound for n_l and that the value of m has become significantly larger than it was initially, then we should move on to the next $l \in I$ (or onto the real reduction procedure if there is no next $l \in I$).

Now we discuss how to choose m and the so-called weights W_i. To optimize the bound for n_l produced by the reduction procedure, one should take m as small as possible while ensuring that the lower bound for $l(\Gamma_m, \mathbf{y})$ furnished by Lemma 18.1 is large enough that the hypothesis of Lemma 20.1 is satisfied. We will use a heuristic argument to determine an approximately optimal choice for m. The main step in the argument is finding an inequality involving m and the W_i that is roughly equivalent to the hypothesis of Lemma 20.1. Let $\mathbf{c}_1, \ldots, \mathbf{c}_{v+r}$ denote any LLL-reduced basis for Γ_m and let B_m be the matrix having $\mathbf{c}_1, \ldots, \mathbf{c}_{v+r}$ as its column-vectors. Since the columns of both A_m and B_m are bases for Γ_m, there is a unimodular matrix U_m such that $A_m U_m = B_m$. Therefore, $|\mathrm{det} A_m| = |\mathrm{det} B_m|$. On the one hand, we see that $|\mathrm{det} A_m| = p_l^m \prod_{i=2}^{1+v+r} W_i$. On the other hand, since $\mathbf{c}_1, \ldots, \mathbf{c}_{v+r}$ are nearly orthogonal, we have $|\mathrm{det} B_m| \approx |\mathbf{c}_1| \cdots |\mathbf{c}_{v+r}|$. Since we can expect that the c_i are all of approximately the same length, we thus have $|\mathbf{c}_1|^{v+r} \approx p_l^m \prod_{i=2}^{1+v+r} W_i$. The lower bound for $l(\Gamma_m, \mathbf{y})$ from Lemma 18.1 is therefore of the size $2^{-(v+r)/2}(p_l^m \prod_{i=2}^{1+v+r} W_i)^{1/(v+r)}$. Thus the hypothesis of Lemma 20.1 is essentially the inequality

$$2^{-(v+r)/2} \left(p_l^m \prod_{i=2}^{1+v+r} W_i \right)^{1/(v+r)} > \left(\sum_{i=2}^{1+v+r} W_i^2 B_i^2 \right)^{1/2},$$

which is equivalent to

$$m > \frac{v+r}{2 \log p} \log \left(\frac{2^{v+r} \sum_{i=2}^{1+v+r} W_i^2 B_i^2}{\left(\prod_{i=2}^{1+v+r} W_i^2 \right)^{1/(v+r)}} \right).$$

Hence the optimal value of m will be approximately

$$m_0 = \frac{v+r}{2 \log p} \log \left(\frac{2^{v+r} \sum_{i=2}^{1+v+r} W_i^2 B_i^2}{\left(\prod_{i=2}^{1+v+r} W_i^2 \right)^{1/(v+r)}} \right).$$

We are free to choose the weights W_i to minimize this expression, but this is not easy to do in general. Instead, we consider two simple candidate choices for the weights. With the choice $W_i = 1$ for all i, we have

$$\frac{\sum_{i=2}^{1+v+r} W_i^2 B_i^2}{\left(\prod_{i=2}^{1+v+r} W_i^2\right)^{1/(v+r)}} = \sum_{i=2}^{1+v+r} B_i^2 \leq (v+r) H_0^2. \tag{28}$$

If we choose W_i as the positive integer nearest H_0/B_i for all i, then

$$\frac{\sum_{i=2}^{1+v+r} W_i^2 B_i^2}{\left(\prod_{i=2}^{1+v+r} W_i^2\right)^{1/(v+r)}} \approx \frac{(v+r) H_0^2}{\left(\prod_{i=2}^{1+v+r} H_0^2/B_i^2\right)^{1/(v+r)}} = (v+r) \left(\prod_{i=2}^{1+v+r} B_i^2\right)^{1/(v+r)} \leq \sum_{i=2}^{1+v+r} B_i^2. \tag{29}$$

We are thus inclined to prefer the second choice of weights over the first. Here, and elsewhere in this thesis, when we refer to the integer nearest to a given real number, we decide arbitrarily whether to round up or down in the case the real number is a half-integer. The correctness of the algorithm is completely insensitive to these decisions.

Note that we are free to make a choice for m that is smaller than m_0. A smaller value of m will produce a better upper bound for n_l if the hypothesis of Lemma 20.1 is satisfied. However, if the hypothesis of Lemma 20.1 cannot be verified, one will incur the cost of computing another LLL-reduced basis after increasing m.

We expect $(1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2))$ to be small, so the upper bound for n_l produced by the reduction procedure will generally look like $(1/h_l)(m - d_l)$. Therefore, the inequalities (28) and (29) imply that, with m the size of m_0 and with either of the two simple choices for the weights considered above, we expect to get a bound of size $\log H_0$, as desired.

21 Basic real reduction

Now we describe how the basic real reduction procedure works.

Under certain conditions, we can feed the improved upper bounds on the n_l obtained through the p_l-adic reduction procedures into Lemma 14.1 and combine the result with Lemma 15.1 to immediately obtain a new upper bound on A. In particular, we have the following lemma.

Lemma 21.1. If $i_0 \in \{s+1, \dots, s+2t\}$ and $\left|x - y\theta^{(i_0)}\right| = \min_{1 \le i \le n} \left|x - y\theta^{(i)}\right|$, then for any $0 < c_{11} < c_{10}/(n-1)$ we have

$$A \le \max\left\{c_{15}, \left\lceil \frac{c_8' + c_9' N_0}{c_{10} - (n-1)c_{11}} \right\rceil - 1, \left\lceil \frac{c_8'' + c_9'' N_0}{c_{10} - c_{11}} \right\rceil - 1 \right\},$$

where $N_0 = \max\{N_1, \dots, N_v\}$.

Proof. This follows immediately from Lemmas 14.1 and 15.1. □

Note that c_8, c_9, and c_{15} are expected to be small and that we are free to choose the parameter c_{11} in such a way that the upper bound for A in Lemma 21.1 becomes optimal.

If $s = 0$, then $\{s+1, \dots, s+2t\} = \{1, \dots, n\}$, and so we must have $\left|x - y\theta^{(i_0)}\right| = \min_{1 \le i \le n} \left|x - y\theta^{(i)}\right|$ for some $i_0 \in \{s+1, \dots, s+2t\}$. Hence Lemma 21.1 gives a new upper bound on A when $s = 0$.

In the $s > 0$ case, we will need to use a lattice reduction based Diophantine approximation method (similar to the one used in the basic p_l-adic reduction procedures) to improve the bound on A. We first treat the real case ($s \ge 3$). Let $C, W_2, \dots, W_{v+r}$ be positive integers. Later in this section, we will discuss how to choose C and the W_i (the weights) to obtain a good reduction of A_0. Note that $\log C$ is the analog of m from the p_l-adic reduction procedures.

Let $i_0 \in \{1, \dots, s\}$, and let $j = j(i_0)$, $k = k(i_0)$, $c_{16} = c_{16}(i_0)$, and $\Lambda_0 = \Lambda_0(i_0)$ be as in Section 15. Put

$$\alpha_1 = \log|\delta_1|, \quad \alpha_{1+i} = \log\left|\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right| \quad (i = 1, \dots, v), \quad \alpha_{1+v+i} = \log\left|\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right| \quad (i = 1, \dots, r).$$

Then

$$\Lambda_0 = \sum_{i=1}^{1+v+r} b_i \alpha_i.$$

For $i = 2, \dots, v+r$, let ϕ_i be the nearest integer to $C\alpha_i$. Let ϕ_{1+v+r} be the nearest integer to $C\alpha_{1+v+r}$ such that $|\phi_{1+v+r}| \geq 2$, and let ϕ_1 be the nearest integer to $C\alpha_1$ such that $\phi_1/\phi_{1+v+r} \notin \mathbb{Z}$. We must, of course, compute the α_i to precision at least $\log C/\log 10$ to avoid errors here.

Let Γ_C be the $(v+r)$-dimensional lattice generated by the column-vectors of the matrix

$$A_C = \begin{bmatrix} W_2 & & & \\ & \ddots & & 0 \\ 0 & & W_{v+r} & \\ \phi_2 & \cdots & \phi_{v+r} & \phi_{1+v+r} \end{bmatrix}.$$

Put

$$\mathbf{y} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\phi_1 \end{bmatrix} \in \mathbb{Z}^{v+r}.$$

If we had $\mathbf{y} \in \Gamma_C$, there would be integers $z_1, \dots, z_{v+r}$ such that $\mathbf{y} = A_m[z_1, \dots, z_{v+r}]^T$. This would then force $z_1 = \cdots = z_{v+r-1} = 0$ and $-\phi_1 = z_{v+r}\phi_{1+v+r}$. But this last equality contradicts the fact that $\phi_1/\phi_{1+v+r} \notin \mathbb{Z}$. Therefore we have $\mathbf{y} \notin \Gamma_C$, and, consequently, we will be able to apply Lemma 18.1 to bound $l(\Gamma_C, \mathbf{y})$ from below after computing a LLL-reduced basis for Γ_C. Put

$$R = \sum_{i=1}^{1+v+r} B_i \left|\phi_i - C\alpha_i\right|, \qquad S = \sum_{i=2}^{v+r} W_i^2 B_i^2.$$

Lemma 21.2. Suppose $i_0 \in \{1, \dots, s\}$ and $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$. Let D be any number such that $l(\Gamma_C, \mathbf{y}) \geq D$. If $D > (R^2 + S)^{1/2}$, then for any $0 < c_{11} < c_{10}/(n-1)$ we have

$$A \leq \max \left\{ \begin{array}{l} \left\lfloor \dfrac{1}{c_{11}} \left(\log(2(\log 2)c_{16}(i_0)) + \log C - \log\left((D^2 - S)^{1/2} - R \right) \right) \right\rfloor, \\ \left\lceil \dfrac{c_8' + c_9' N_0}{c_{10} - (n-1)c_{11}} \right\rceil - 1, \left\lceil \dfrac{c_8'' + c_9'' N_0}{c_{10} - c_{11}} \right\rceil - 1, \left\lceil \dfrac{\log(2c_{16}(i_0))}{c_{11}} \right\rceil - 1, 0 \end{array} \right\}$$

As with Lemma 20.1, we are free to choose c_{11} in Lemma 21.2 so that the upper bound on A becomes optimal.

Proof. Assume $l(\Gamma_C, \mathbf{y}) > (R^2 + S)^{1/2}$. Consider the lattice point

$$\mathbf{x} = A_C \begin{bmatrix} b_2 \\ \vdots \\ b_{1+v+r} \end{bmatrix} = \begin{bmatrix} W_2 b_2 \\ \vdots \\ W_{v+r} b_{v+r} \\ \Phi \end{bmatrix} + \mathbf{y},$$

where

$$\Phi = \sum_{i=1}^{1+v+r} b_i \phi_i.$$

We have

$$|\Phi| \le |\Phi - C\Lambda_0| + |C\Lambda_0| = \left| \sum_{i=1}^{1+v+r} b_i \left(\phi_i - C\alpha_i\right) \right| + |C\Lambda_0| \le R + |C\Lambda_0|,$$

and so, since $\mathbf{y} \notin \Gamma_C$,

$$l(\Gamma_C, \mathbf{y})^2 \le |\mathbf{x} - \mathbf{y}|^2 = \sum_{i=2}^{v+r} W_i^2 b_i^2 + |\Phi|^2 \le S + \left(R + |C\Lambda_0|\right)^2.$$

Since $l(\Gamma_C, \mathbf{y}) \ge D > (R^2 + S)^{1/2}$, the inequality above implies

$$\left(D^2 - S\right)^{1/2} - R \le |C\Lambda_0|.$$

Now assume

$$A \ge \frac{c_8' + c_9' N}{c_{10} - (n-1)c_{11}}, \qquad A \ge \frac{c_8'' + c_9'' N}{c_{10} - c_{11}}, \qquad A \ge \left\lceil \frac{\log(2c_{16}(i_0))}{c_{11}} \right\rceil, \qquad A \ge 1$$

By Lemma 14.1, the first two inequalities imply $\left|x - y\theta^{(i_0)}\right| \le e^{-c_{11}A}$. Then the third and fourth inequalities give, as in the proof of Lemma 15.3,

$$|\Lambda_0| \le 2(\log 2) c_{16}(i_0) e^{-c_{11}A}.$$

Therefore we have

$$\left(D^2 - S\right)^{1/2} - R \le 2(\log 2) c_{16}(i_0) C e^{-c_{11}A},$$

which is equivalent to

$$A \le \frac{1}{c_{11}} \left(\log(2(\log 2) c_{16}(i_0)) + \log C - \log\left((D^2 - S)^{1/2} - R \right) \right).$$

□

Now we describe the reduction procedure. For a given $i_0 \in \{1, \ldots, n\}$, we will call a number U an i_0-conditional upper bound for A if $A \leq U$ holds in the case $\left|x - y\theta^{(i_0)}\right| = \min_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$. For each $i_0 \in \{1, \ldots, n\}$, we will use $A_0(i_0)$ to denote the best i_0-conditional upper bound for A that we have at the current stage in the procedure. At the start of the procedure, $A_0(i_0) = A_0$ for every $i_0 \in \{1, \ldots, n\}$. For each $i_0 \in \{1, \ldots, s\}$ in turn, we compute an LLL-reduced basis for Γ_C, and then use Lemma 18.1 to find a lower bound D for $l(\Gamma_C, \mathbf{y})$. If D is greater than $(R^2+S)^{1/2}$, Lemma 21.2 gives an i_0-conditional upper bound for A that we hope reduces $A_0(i_0)$. If $D \leq (R^2 + S)^{1/2}$, we should increase C somewhat and try again. Furthermore, when D is greater than $(R^2 + S)^{1/2}$, but the i_0-conditional bound produced by Lemma 21.2 does not improve $A_0(i_0)$, we should also increase C and try again. As in Section 20, if C needs to be increased to C' we should take as input to the LLL algorithm the partially reduced basis for Γ'_C given by the columns of $A'_C U_C$ (where U_C is the unimodular integer matrix for which the columns of $A_C U_C$ form the LLL-reduced basis we computed for Γ_C) rather than the basis given by $A_{C'}$. Once we run through all $i_0 \in \{1, \ldots, s\}$, we use Lemma 21.1 to compute a number that is an i_0-conditional upper bound for A for each $i_0 \in \{s+1, \ldots, s+2t\}$. Since we must have $\left|x - y\theta^{(i_0)}\right| = \min_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$ for some $i_0 \in \{1, \ldots, n\}$, we know $\max_{i_0 \in \{1,\ldots,n\}} A_0(i_0)$ is an unconditional upper bound for A. Note that this will be an improvement on the upper bound for A we had at the start of the procedure if and only if every i_0-conditional upper bound for A calculated in the procedure is smaller than the starting upper bound for A. Therefore, if, for a given $i_0 \in \{1, \ldots, s\}$, we have found no new i_0-conditional upper bound for A after several increases of C, and the value of C has become significantly larger than it was initially, then we should terminate the procedure (there is no point in moving on to the next value of $i_0 \in \{1, \ldots, s\}$).

Now we discuss how to choose C and the weights W_i. To optimize the i_0-conditional upper bounds for A produced by Lemma 21.2, we want the expression

$$\log(2(\log 2)c_{16}(i_0)) + \log C - \log((D^2 - S)^{1/2} - R) \tag{30}$$

to be as small as possible while still ensuring $D > (R^2 + S)^{1/2}$. Here D is the lower bound for $l(\Gamma_C, \mathbf{y})$ from Lemma 18.1. Arguing heuristically as in Section 20, we see that D is of the size of $2^{-(v+r)/2} \left(C \left|\alpha_{1+v+r}\right| \prod_{i=2}^{v+r} W_i\right)^{1/(v+r)}$. Using that $R \approx \sum_{i=1}^{1+v+r} B_i/2$, we find

that $D > (R^2 + S)^{1/2}$ is essentially equivalent to the inequality $C > C_0$, where

$$C_0 = \left(\frac{\left(\sum_{i=1}^{1+v+r} B_i/2 \right)^2 + \sum_{i=2}^{v+r} W_i^2 B_i^2}{2^{-(v+r)} \left(|\alpha_{1+v+r}| \prod_{i=2}^{v+r} W_i \right)^{2/(v+r)}} \right)^{(v+r)/2}.$$

It follows that as C decreases towards C_0, D decreases towards $(R^2+S)^{1/2}$, and, therefore, $-\log\left((D^2 - S)^{1/2} - R\right)$ increases towards infinity. However, $-\log\left((D^2 - S)^{1/2} - R\right) \leq 0$ if we have $D > ((R+1)^2 + S)^{1/2}$, which is essentially equivalent to

$$C > \left(\frac{\left(1 + \sum_{i=1}^{1+v+r} B_i/2 \right)^2 + \sum_{i=2}^{v+r} W_i^2 B_i^2}{2^{-(v+r)} \left(|\alpha_{1+v+r}| \prod_{i=2}^{v+r} W_i \right)^{2/(v+r)}} \right)^{(v+r)/2}.$$

The difference between the expression on the right-hand side of this inequality and C_0 is negligible in our approximation. Therefore, we can safely choose a value the size of C_0 for C, and this choice is essentially the best possible with respect to minimizing the contribution of $\log C$ to the upper bound for A. We can do better with the bound for A if we try to minimize the contribution of the expression $\log C - \log\left((D^2 - S)^{1/2} - R\right)$ as a whole. This expression will be essentially minimized if we select C so that the two terms are equal, i.e, if $C^{-1} = (D^2 - S)^{1/2} - R$. Using the same approximations for D and R as before, this leads to a polynomial equation for C like

$$2^{-(v+r)^2} \left(|\alpha_{1+v+r}| \prod_{i=2}^{v+r} W_i \right)^2 C^{2(v+r+1)} = \left(\left(1 + C \sum_{i=1}^{1+v+r} B_i/2 \right)^2 + C^2 \sum_{i=2}^{v+r} W_i^2 B_i^2 \right)^{v+r}.$$

We can try to find an approximate solution to this equation that satisfies $C > C_0$ and use that as our choice of C. As long as we choose weights that are independent of C, we can solve this equation numerically. An easier method is to start with a value for C of the size of C_0, and then increment or decrement C (with some convenient step size) until $C^{-1} \approx (D^2 - S)^{1/2} - R$, where $D \approx 2^{-(v+r)/2} (C\,|\alpha_{1+v+r}| \prod_{i=2}^{v+r} W_i)^{1/(v+r)}$.

For the weights W_i, either choosing $W_i = 1$ for all i or choosing W_i to be the nearest positive integer to $H_0'/B_i = \max\limits_{2 \leq j \leq v+r} B_j/B_i$ for all i works well. It is not immediately obvious that one choice of weights is strictly better than the other. However, the second

choice tends to give a more balanced lattice, which improves the quality of the lower bound on $l(\Gamma, \mathbf{y})$ and the quality of the heuristics we used above.

We expect that the maximum of the upper bounds for A coming from Lemmas 21.2 and 21.1 will be of the size $\log C$. Since the discussed choices for C and the weights W_i will tend to give $\log C \approx \log H_0$, we thus expect a new bound for A like $\log H_0$.

The complex case ($s = 1, 2$) is nearly identical to the real case. Let $i_0 \in \{1, \dots, s\}$, and let $j = j(i_0)$, $k = k(i_0)$, $c_{16} = c_{16}(i_0)$, $\Lambda_0 = \Lambda_0(i_0)$, and $a_0 = a_0(i_0)$ be as in Section 15. Let $C, W_2, \dots, W_{1+v+r}$ be positive integers. We will discuss how to choose C and the weights W_i at the end. Put $b_{2+v+r} = 2a_0$. It follows from the proof of Lemma 15.3 that $|b_{2+v+r}|$ is bounded by $B_{2+v+r} = 2\sin^{-1}(1/4)/\pi + \sum_{i=1}^{1+v+r} B_i$. Define

$$\alpha_1 = \operatorname{Im}\left(\operatorname{Log}(\delta_1)\right), \quad \alpha_{1+i} = \operatorname{Im}\left(\operatorname{Log}\left(\frac{\pi_i^{(k)}}{\pi_i^{(j)}}\right)\right) \quad (i = 1, \dots, v),$$

$$\alpha_{1+v+i} = \operatorname{Im}\left(\operatorname{Log}\left(\frac{\varepsilon_i^{(k)}}{\varepsilon_i^{(j)}}\right)\right) \quad (i = 1, \dots, r), \quad \alpha_{2+v+r} = \operatorname{Im}\left(\operatorname{Log}(-1)\right) = \pi.$$

Then

$$\Lambda_0 = \sum_{i=1}^{2+v+r} b_i \alpha_i.$$

For $i = 2, \dots, 1 + v + r$, let ϕ_i be the nearest integer to $C\alpha_i$. Let ϕ_{1+v+r} be the nearest integer to $C\alpha_{2+v+r}$ such that $|\phi_{2+v+r}| \geq 2$ and let ϕ_1 be the nearest integer to $C\alpha_1$ such that $\phi_1/\phi_{2+v+r} \notin \mathbb{Z}$. Let Γ_C be the $(1 + v + r)$-dimensional lattice generated by the column-vectors of the matrix

$$A_C = \begin{bmatrix} W_2 & & & \\ & \ddots & & 0 \\ 0 & & W_{1+v+r} & \\ \phi_2 & \cdots & \phi_{1+v+r} & \phi_{2+v+r} \end{bmatrix}.$$

Put

$$\mathbf{y} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\phi_1 \end{bmatrix} \in \mathbb{Z}^{1+v+r}.$$

As in the real case, $\mathbf{y} \notin \Gamma_C$, so Lemma 18.1 can be applied to bound $l(\Gamma_C, \mathbf{y})$ from below.

It is easily proved that Lemma 21.2 holds with

$$R = \sum_{i=1}^{2+v+r} B_i \left|\phi - C\alpha_i\right|, \qquad S = \sum_{i=2}^{1+v+r} W_i^2 B_i^2,$$

and with the expression $2(\log 2)$ replaced by $4\sin^{-1}(1/4)$. The reduction procedure runs exactly as in the real case. An analysis like the one from the real case indicates choosing

$$C \approx \left(\frac{\left(\sum_{i=1}^{2+v+r} B_i/2 \right)^2 + \sum_{i=2}^{1+v+r} W_i^2 B_i^2}{2^{-(v+r+1)} \left(|\alpha_{2+v+r}| \prod_{i=2}^{1+v+r} W_i \right)^{2/(v+r+1)}} \right)^{(v+r+1)/2}$$

is optimal when it comes to minimizing the contribution of the $\log C$ term in the bound produced by (the complex case version of) Lemma 21.2. Adjusting the value of C a bit so that C^{-1} is closer to $(D^2 - S)^{1/2} - R$ (where we use the approximation $D \approx 2^{-(v+r+1)/2}\big(C\left|\alpha_{2+v+r}\right| \prod_{i=2}^{1+v+r} W_i\big)^{1/(1+v+r)}$) will help improve the bound further. The choice of weights $W_i = 1$ for all i is, of course, valid. For the same reason as in the real case, taking W_i equal to the nearest positive integer to $H_0/B_i = \max\limits_{2 \le j \le 1+v+r} B_j/B_i$ for all i tends to be preferable.

22 Refined p_l-adic reduction

In this section, we explain the refined p_l-adic reduction procedure.

Fix $l \in I$. Let m be a positive integer. The solutions of (16) fall into one of two cases:

Case 1.
$$n_l > \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right) \quad \text{and} \quad n_l \geq \frac{1}{h_l}(m - d_l)$$

Case 2.
$$n_l \leq \max\left\{\left\lfloor \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rfloor, \left\lceil \frac{1}{h_l}(m - d_l)\right\rceil - 1\right\}$$

The refined p_l-adic reduction procedure is essentially a technique to efficiently find all the solutions of (16) in Case 1. So, at the end of the procedure, we will know that all the solutions that have yet to be found satisfy

$$n_l \leq \max\left\{\left\lfloor \frac{1}{h_l}\left(\frac{1}{p_l - 1} - \mathrm{ord}_{p_l}(\delta_2)\right)\right\rfloor, \left\lceil \frac{1}{h_l}(m - d_l)\right\rceil - 1\right\}.$$

Thus, if m is small enough, we will have found an improvement of the upper bound for n_l. We will further discuss the choice of m later in this section.

Choose positive integers $W_2, \ldots, W_{1+v+r}$ with $W_2, \ldots, W_{v+1}$ even. As with m, we postpone the details of how to choose the W_i until later in this Section. Define Γ_m, A_m, and λ as in Section 20. If $2 \leq \hat{i} \leq v+1$, set

$$\mathbf{y} = \begin{bmatrix} \frac{1}{2}W_2 B_2 \\ \vdots \\ \frac{1}{2}W_{\hat{i}-1}B_{\hat{i}-1} \\ \frac{1}{2}W_{\hat{i}+1}B_{\hat{i}+1} \\ \vdots \\ \frac{1}{2}W_{v+1}B_{v+1} \\ 0 \\ \vdots \\ 0 \\ -W_{\hat{i}}\beta_1^{\{m\}} + \frac{1}{2}W_{\hat{i}}B_{\hat{i}} \end{bmatrix} \in \mathbb{Z}^{v+r};$$

otherwise set

$$\mathbf{y} = \begin{bmatrix} \frac{1}{2}W_2B_2 \\ \vdots \\ \frac{1}{2}W_{v+1}B_{v+1} \\ 0 \\ \vdots \\ 0 \\ -W_{\hat{i}}\beta_1^{\{m\}} \end{bmatrix} \in \mathbb{Z}^{v+r}.$$

Choose a vector $\mathbf{z} \in \Gamma_m$ that is close to $\mathbf{y}$. An efficient way to do this is as follows. Compute a matrix B_m whose column-vectors $\mathbf{c}_1, \ldots, \mathbf{c}_{v+r}$ form an LLL-reduced basis for Γ_m, and write

$$\mathbf{y} = s_1\mathbf{c}_1 + \cdots + s_{v+r}\mathbf{c}_{v+r}, \quad s_i \in \mathbb{R}$$

(in other words, compute $[s_1, \ldots, s_{v+r}]^T = B_m^{-1}\mathbf{y}$). Choose $\mathbf{t} \in \mathbb{Z}^{v+r}$ such that $|t_i - s_i| \leq 1$ for all i and $|\mathbf{y} - B_m\mathbf{t}|$ is minimal. Then $B_m\mathbf{t}$ is likely the closest lattice vector to $\mathbf{y}$, so we take $\mathbf{z} = B_m\mathbf{t}$.

Set

$$D = \left(\sum_{i=2}^{v+1} \left(\frac{1}{2}W_iB_i \right)^2 + \sum_{i=2+v}^{1+v+r} (W_iB_i)^2 \right)^{1/2}, \qquad D_0 = D + |\mathbf{y} - \mathbf{z}|\,.$$

Lemma 22.1. In Case 1, the tuple $(b_2, \ldots, b_{1+v+r})$ satisfies $\mathbf{x} - \mathbf{z} \in \Gamma_m$ and $|\mathbf{x} - \mathbf{z}| \leq D_0$, where

$$\mathbf{x} = A_m \begin{bmatrix} b_2 \\ \vdots \\ b_{\hat{i}-1} \\ b_{\hat{i}+1} \\ \vdots \\ b_{1+v+r} \\ \lambda \end{bmatrix}.$$

Proof. Suppose $n_l > (1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2))$ and $n_l \geq (1/h_l)(m - d_l)$. Then, as in the proof of Lemma 20.1, $\lambda \in \mathbb{Z}$. So $\mathbf{x}$ belongs to Γ_m, and, therefore, $\mathbf{x} - \mathbf{z} \in \Gamma_m$ also. If

$2 \leq \hat{i} \leq v+1$, then

$$\mathbf{x}-\mathbf{y} = \begin{bmatrix} W_2 b_2 - \frac{1}{2} W_2 B_2 \\ \vdots \\ W_{\hat{i}-1} b_{\hat{i}-1} - \frac{1}{2} W_{\hat{i}-1} B_{\hat{i}-1} \\ W_{\hat{i}+1} b_{\hat{i}+1} - \frac{1}{2} W_{\hat{i}+1} B_{\hat{i}+1} \\ \vdots \\ W_{v+1} b_{v+1} - \frac{1}{2} W_{v+1} B_{v+1} \\ W_{v+2} b_{v+2} \\ \vdots \\ W_{1+v+r} b_{1+v+r} \\ W_{\hat{i}} b_{\hat{i}} - \frac{1}{2} W_{\hat{i}} B_{\hat{i}} \end{bmatrix}, \tag{31}$$

while if $v+2 \leq \hat{i} \leq 1+v+r$, we have

$$\mathbf{x}-\mathbf{y} = \begin{bmatrix} W_2 b_2 - \frac{1}{2} W_2 B_2 \\ \vdots \\ W_{v+1} b_{v+1} - \frac{1}{2} W_{v+1} B_{v+1} \\ W_{v+2} b_{v+2} \\ \vdots \\ W_{\hat{i}-1} b_{\hat{i}-1} \\ W_{\hat{i}+1} b_{\hat{i}+1} \\ \vdots \\ W_{1+v+r} b_{1+v+r} \\ W_{\hat{i}} b_{\hat{i}} \end{bmatrix}. \tag{32}$$

Either way,

$$|\mathbf{x}-\mathbf{y}| \leq \left(\sum_{i=2}^{v+1} \left(\frac{1}{2} W_i B_i \right)^2 + \sum_{i=2+v}^{1+v+r} (W_i B_i)^2 \right)^{1/2} = D.$$

Hence the lattice vector $\mathbf{x}-\mathbf{z}$ must satisfy

$$|\mathbf{x}-\mathbf{z}| = |\mathbf{x}-\mathbf{y}| + |\mathbf{y}-\mathbf{z}| \leq D_0.$$

□

We now use the Fincke-Pohst algorithm to efficiently compute all vectors $\mathbf{u} \in \Gamma_m$ with $|\mathbf{u}| \leq D_0$. Each such $\mathbf{u}$ is a candidate to have $\mathbf{u} = \mathbf{x}-\mathbf{z}$. For each $\mathbf{u}$, we compute

the corresponding tuple $(b_2, \ldots, b_{1+v+r})$ assuming $\mathbf{u} = \mathbf{x} - \mathbf{z}$. This assumption lets us write $\mathbf{x} - \mathbf{y} = \mathbf{u} + \mathbf{z} - \mathbf{y}$, and so, since $\mathbf{u}, \mathbf{y}, \mathbf{z}$ are known explicitly, we can easily compute $(b_2, \ldots, b_{1+v+r})$ using (31) or (32), as appropriate. Each tuple $(b_2, \ldots, b_{1+v+r})$ that satisfies (16) (for some x, y) and satisfies the Case 1 inequalities must be one of the tuples computed in this way. However, the computed tuples are not necessarily all solutions of (16), so they will need to actually be tested for (16). This can be done, for example, by computing the coefficients of the right-hand side of (16) and verifying that the coefficients of $\theta^2, \ldots, \theta^{n-1}$ of (16) all vanish. However, due to the large number of tuples likely to be computed, it is not practical at this stage to test every tuple for (16) immediately. It is preferable to first perform on each tuple a series of tests (see below) that rule-out non-solutions with minimal computational effort. Then, if a tuple manages to survive these tests, we can test it for (16). In practice, very few non-solutions will survive the series of computationally cheap tests. One could also choose to store those tuples that pass the series of cheap tests for further testing later, rather than test the tuple for (16) at this stage.

We now make a remark on implementation. There are likely to be a large number of lattice vectors with norm less than or equal to D_0. To save on memory usage, immediately after each vector u is found, one should extract the tuple, test it, and, if and only if it corresponds to a solution, store it. One should not store all the computed vectors or all the extracted tuples.

Now we discuss a series of efficient tests that we can apply to the tuple $(b_2, \ldots, b_{1+v+r})$ to eliminate a non-solution. Note that the remarks in Section 19 about verifying the conclusions of Lemmas 8.2 and 19.1 concern the last few tests we will describe below. As a first test, we check that the b_i are all integers. After that, we check that the tuple satisfies the current bounds for the b_i: $|b_i| \leq B_i$ for all i, and $b_2, \ldots, b_{1+v} \geq 0$. Next, we eliminate the tuple if it satisfies $n_l \leq \max\{(1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2)), \lceil (1/h_l)(m - d_l) \rceil - 1\}$ because we are only looking for those solutions that fall into Case 1. By Lemma 19.1, a tuple that survives the last test must satisfy $\mathrm{ord}_{p_l}(\Lambda'_l) \geq n_l h_l + d_l$ if it corresponds to a solution. If this inequality does not hold, we eliminate the tuple. Note that if we are in the special case described in Section 19, Lemma 19.1 implies the tuple must actually satisfy $\mathrm{ord}_{p_l}(\Lambda'_l) = n_l h_l + d_l$. At this point, we have verified that the tuple does not contradict the instance of Lemma 19.1 corresponding to our fixed $l \in I$. We next check that the tuple does not contradict Lemma 19.1 for those indices $l_1 \in I$ with $l_1 \neq l$. More precisely, for each $l_1 \in I \setminus \{l\}$ with $n_{l_1} > (1/h_{l_1})(1/(p_{l_1} - 1) - \mathrm{ord}_{p_{l_1}}(\delta_2))$, we check whether $\mathrm{ord}_{p_{l_1}}(\Lambda'_{l_1}) \geq$

$n_{l_1}h_{l_1} + d_{l_1}$ holds (whether $\mathrm{ord}_{p_{l_1}}(\Lambda'_{l_1}) = n_{l_1}h_{l_1} + d_{l_1}$ holds if we are in the special case) and discard the tuple if not. Now, a tuple that has survived up to this stage must satisfy $n_l > (1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2))$ (a tuple with $n_l \leq (1/h_l)\,(1/(p_l - 1) - \mathrm{ord}_{p_l}(\delta_2))$ would have been eliminated at a previous stage). Hence, if the tuple corresponds to a solution, Lemma 8.2 implies that $\mathrm{ord}_{p_l}(\Lambda_l) = n_l h_l + \mathrm{ord}_{p_l}(\delta_2)$. If this equation does not hold, we eliminate the tuple. Note that there is no point in checking this equation if we are in the special case (see the comments made in Section 19). Finally, if the tuple has still not been ruled-out, we test it for the versions of Lemma 8.2 corresponding to those indices $l_1 \in I \setminus \{l\}$ for which the general case of Section 19 applies. That is, for each such index l_1, if $n_{l_1} > (1/h_{l_1})(1/(p_{l_1} - 1) - \mathrm{ord}_{p_{l_1}}(\delta_2))$, we check whether $\mathrm{ord}_{p_{l_1}}(\Lambda_{l_1}) = n_{l_1}h_{l_1} + \mathrm{ord}_{p_{l_1}}(\delta_2)$, and, if this equality does not hold, we discard the tuple. Note that it is easy to compute Λ_l and Λ'_l (and Λ_{l_1} and Λ'_{l_1}) because the α_i and β_i will have already been computed and the b_i are given to us.

Note that it is possible to make a different choice for $\mathbf{y}$. However, the one we have chosen produces a bound D_0 that is reasonably small in terms of the weights. For a given lattice and numbers $0 < D_1 < D_2$, it is of course cheaper to enumerate the smaller set consisting of those lattice vectors with norm less than D_1 than to enumerate the larger set of lattice vectors with norm less than D_2. Since the parameters m and W_i determine the lattice, it is advantageous to have D_0 be small in terms of them.

Now we discuss how to choose the weights. If the W_i are taken to be small, D_0 will also be small and we might hope that this will make the process of enumerating all the vectors with norm less than or equal to D_0 fast. However, small weights produce short basis vectors for the lattice Γ_C, which in turn produce more short vectors in the lattice overall. This offsets the decrease in the norm bound. Hence having smaller weights does not necessarily reduce the enumeration cost. There is a balance to be struck between the size of D_0 and the number of short vectors in the lattice. The following choice works well. For $i = 2, \ldots, v + 1$, define W_i so that $\frac{1}{2}W_i$ is the nearest positive integer to H_0/B_i. For $i = v + 2, \ldots, 1 + v + r$, let W_i be the nearest positive integer to H_0/B_i. This choice also prevents the lattice from being badly skewed (having a skewed lattice makes the enumeration more difficult).

Finally, we discuss the value of m. To make the new upper bound for n_l as small as possible, we should take m as small as possible. Unlike with the simple p_l-adic reduction, there is now no condition (like $l(\Gamma_m, \mathbf{y}) > Q^{1/2}$) to limit how small m can theoretically

be chosen. However, if we take m too small, the lattice Γ_m will have so many vectors of norm less than or equal to D_0 that the computational cost to enumerate (and test) them all will be prohibitive. Choosing m as in Section 20 seems to work well in practice. In fact, we have found that a significantly smaller choice of m (like 3/4 the size suggested in Section 20) works fine. Note that it is possible to estimate the cost of the Fincke-Pohst enumeration for a given lattice and norm bound before performing the enumeration. If the estimated cost is too large, one should increase m and try the procedure again. If several increases of m yield no new upper bound for n_l, we should move on to the next $l \in I$ (or onto the real reduction if there is no next $l \in I$).

23 Refined real reduction

Here we describe the refined real reduction procedure.

The outline of the refined real reduction procedure is the same as for the basic procedure. If $s = 0$, Lemma 21.1 immediately gives a new upper bound for A exactly as in the basic procedure. For the $s > 0$ case, we first perform lattice computations to find i_0-conditional upper bounds for A for each $i_0 \in \{1, \dots, s\}$ and use Lemma 21.1 to find an upper bound on A that is i_0-conditional for every $i_0 \in \{s+1, \dots, s+2t\}$. Then the maximum of all these upper bounds is taken as an unconditional upper bound for A. We describe below how the i_0-conditional upper bounds for A are computed for $i_0 \in \{1, \dots, s\}$. This is the only point on which the refined and basic reduction procedures differ.

Suppose first that we are in the real case ($s \geq 3$). Let $C, W_2, \dots, W_{v+r}$ be positive integers with $W_2, \dots, W_{v+1}$ even. We will discuss how to choose C and the weights W_i later in this section. Let $i_0 \in \{1, \dots, s\}$ and let $j = j(i_0)$, $k = k(i_0)$, $c_{16} = c_{16}(i_0)$, and $\Lambda_0 = \Lambda_0(i_0)$ be as in Section 15. Define α_i, ϕ_i ($i = 1, \dots, 1+v+r$), A_C, Γ_C, R, and S as in Section 21. Put

$$\mathbf{y} = \begin{bmatrix} \frac{1}{2}W_2B_2 \\ \vdots \\ \frac{1}{2}W_{v+1}B_{v+1} \\ 0 \\ \vdots \\ 0 \\ -\phi_1 \end{bmatrix} \in \mathbb{Z}^{v+r}$$

Choose a vector $\mathbf{z} \in \Gamma_C$ that is close to $\mathbf{y}$ (see Section 22 for a method to do this). Choose a number $D > (R^2 + S)^{1/2}$ and set

$$D_0 = D + |\mathbf{y} - \mathbf{z}|.$$

Let c_{11} be any number satisfying $0 < c_{11} < c_{10}/(n-1)$. The solutions of (16) are divided into two cases depending on A:

Case 1.

$$A \geq \max \left\{ \begin{array}{c} \frac{1}{c_{11}} \left(\log(2(\log 2)c_{16}(i_0)) + \log C - \log((D^2 - S)^{1/2} - R) \right), \\ \frac{c'_8 + c'_9 N_0}{c_{10} - (n-1)c_{11}}, \frac{c''_8 + c''_9 N_0}{c_{10} - c_{11}}, \left\lceil \frac{\log(2c_{16}(i_0))}{c_{11}} \right\rceil, 1 \end{array} \right\}$$

Case 2.

$$A \leq \max\left\{ \begin{array}{c} \left\lceil \frac{1}{c_{11}} \left(\log(2(\log 2)c_{16}(i_0)) + \log C - \log\left((D^2 - S)^{1/2} - R \right) \right) \right\rceil - 1, \\ \left\lceil \frac{c'_8 + c'_9 N_0}{c_{10} - (n-1)c_{11}} \right\rceil - 1, \left\lceil \frac{c''_8 + c''_9 N_0}{c_{10} - c_{11}} \right\rceil - 1, \left\lceil \frac{\log(2c_{16}(i_0))}{c_{11}} \right\rceil - 1, 0 \end{array} \right\}$$

Note that one should select c_{11} to minimize the upper bound for A in Case 2.

The main computational step in the procedure is finding all the solutions of (16) that fit the first case and that have $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$. Once that is done, we know that all the solutions of (16) with $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$ that still need to be found belong to the second case. So, if C is small enough, we will have found an improvement of the upper bound for A subject to the condition $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$. This is the i_0-conditional upper bound for A we seek.

Lemma 23.1. Suppose $i_0 \in \{1, \ldots, s\}$ and $\left|x - y\theta^{(i_0)}\right| = \min\limits_{1 \leq i \leq n} \left|x - y\theta^{(i)}\right|$. In Case 1, the tuple $(b_2, \ldots, b_{1+v+r})$ satisfies $\mathbf{x} - \mathbf{z} \in \Gamma_C$ and $|\mathbf{x} - \mathbf{z}| \leq D_0$, where

$$\mathbf{x} = A_C \begin{bmatrix} b_2 \\ \vdots \\ b_{1+v+r} \end{bmatrix}.$$

Proof. Since $\mathbf{x}$ and $\mathbf{z}$ both belong to Γ_C, $\mathbf{x} - \mathbf{z} \in \Gamma_C$ as well. By Lemma 14.1 and the inequalities

$$A \geq \frac{c'_8 + c'_9 N}{c_{10} - (n-1)c_{11}}, \quad A \geq \frac{c''_8 + c''_9 N}{c_{10} - c_{11}},$$

we have $\left|x - y\theta^{(i_0)}\right| \leq e^{-c_{11}A}$. Then, as in the proof of Lemma 15.3, it follows from the inequality $A \geq \max\left\{\lceil \log(2c_{16}(i_0))/c_{11} \rceil, 1\right\}$ that

$$|\Lambda_0| \leq 2(\log 2)c_{16}(i_0)e^{-c_{11}A}.$$

So since

$$\mathbf{x} - \mathbf{y} = \begin{bmatrix} W_2 b_2 - \frac{1}{2} W_2 B_2 \\ \vdots \\ W_{v+1} b_{v+1} - \frac{1}{2} W_{v+1} B_{v+1} \\ W_{v+2} b_{v+2} \\ \vdots \\ W_{v+r} b_{v+r} \\ \Phi \end{bmatrix} \tag{33}$$

with

$$\Phi = \sum_{i=1}^{1+v+r} b_i \phi_i,$$

we have

$$|\mathbf{x}-\mathbf{y}|^2 \le S + \Phi^2 \le S + (|C\Lambda_0| + |C\Lambda_0 - \Phi|)^2 \le S + \left(2(\log 2)c_{16}(i_0)Ce^{-c_{11}A} + R\right)^2.$$

As

$$A \ge \frac{1}{c_{11}}\left(\log(2(\log 2)c_{16}(i_0)) + \log C - \log((D^2-S)^{1/2} - R)\right),$$

it follows that $|\mathbf{x}-\mathbf{y}|^2 \le D^2$, whence $|\mathbf{x}-\mathbf{z}| \le D_0$. □

Lemma 23.1 justifies the following method for computing the solutions of (16) in Case 1 having $\left|x - y\theta^{(i_0)}\right| = \min_{1\le i\le n}\left|x - y\theta^{(i)}\right|$. We compute (using the Fincke-Pohst algorithm) all vectors $\mathbf{u} \in \Gamma_C$ with $|\mathbf{u}| \le D_0$. For each such u, we compute the corresponding tuple $(b_2, \ldots, b_{1+v+r})$ assuming $\mathbf{u} = \mathbf{x} - \mathbf{z}$. This assumption lets us write $\mathbf{x} - \mathbf{y} = \mathbf{u} + \mathbf{z} - \mathbf{y}$, so, since $\mathbf{u}, \mathbf{y}, \mathbf{z}$ are known explicitly, we can easily compute $(b_2, \ldots, b_{1+v+r})$ using (33). We then put the tuples through a series of tests (see below) that will eliminate most non-solutions through a small amount of computation. Finally, the few tuples that remain are tested for (16). As with the refined p_l-adic reduction procedure, it would be impractical to simply test in (16) every tuple we compute.

The tests we apply to the tuple $(b_2, \ldots, b_{1+v+r})$ are the essentially the same as in the refined p_l-adic reduction. First, we check that the b_i are all integers and that the current bounds $b_2, \ldots, b_{1+v} \ge 0$ and $|b_i| \le B_i$ hold. Next, if the tuple satisfies the Case 2 inequality for A (i.e., the soon to be new i_0-conditional upper bound for A), then it can be discarded because we are only looking for tuples that fit into Case 1. After that, we check, for each $l \in I$, that the tuple does not contradict Lemma 19.1. If a tuple passes those tests, we perform the same check for Lemma 8.2 for all $l \in I$. Note that this last test is redundant for those $l \in I$ for which we are in the special case described in Section 19.

Let us now discuss the choice of C. As it was in Section 22 with m and Γ_m, taking C too small will produce a lattice Γ_C that has more vectors with norm less D_0 than can be enumerated in a reasonable amount of time. It seems to work well in practice to choose C somewhat (but not too much) smaller than suggested in Section 21. For instance, a reasonable choice is to select a value of C that makes $\log C$ about 3/4 the size it would be if one used the value for C suggested in Section 21.

It is recommended that one estimates the cost of the enumeration before performing it. If the estimate is too large, one should increase C, recalculate the necessary quantities (like a basis for Γ_C) and estimate the enumeration cost again. Once a small enough estimate is found, the enumeration can proceed. If several increases of C have yielded no new i_0-conditional upper bound for A and C has become significantly larger than it was initially, the procedure should be terminated.

It is advantageous to choose a small value of D to keep the cost of the enumeration down. However, taking D too close to its lower limit $(R^2+S)^{1/2}$ will cause the term $-\log((D^2-S)-R)$ in the i_0-conditional upper bound for A (i.e., in the Case 2 inequality) to blow up. Note that the choice $D=((R+1)^2+S)^{1/2}$ is reasonably close $(R^2+S)^{1/2}$ and makes $-\log((D^2-S)-R)=0$.

Now comes the choice of the weights. Since $S=\sum_{i=2}^{v+1}\left(\frac{1}{2}W_iB_i\right)^2+\sum_{i=v+2}^{v+r}(W_iB_i)^2$, the smaller the weights are, the smaller D can be. For a given lattice, a smaller value of D leads to a lower cost to enumerate the vectors of norm less than D. However, the lattice Γ_C depends on the weights, and smaller weights tend to produce shorter vectors in the lattice overall. In terms of the enumeration cost, this last phenomenon tends to counteract the decrease in D that smaller weights produce. There is a balance to be struck between the size of D and the size of the weights. The following choice of weights seems to work well. For $i=2,\dots,1+v$, choose W_i so that $\frac{1}{2}W_i$ is the nearest positive integer to $\max_{2\le j\le v+r} B_j/B_i$. For $i=2+v,\dots,v+r$, W_i is the nearest positive integer to $\max_{2\le j\le v+r} B_j/B_i$.

The complex case ($s=1,2$) is identical to the real case ($s\ge 3$) case except for a few modifications. We let $C,W_2,\dots,W_{1+v+r}$ be positive integers with $W_2,\dots,W_{v+1}$ even. Let $i_0\in\{1,\dots,s\}$ and let $j=j(i_0)$, $k=k(i_0)$, $c_{16}=c_{16}(i_0)$, $\Lambda_0=\Lambda_0(i_0)$, and $a_0=a_0(i_0)$ be as in Section 15. Define α_i, ϕ_i ($i=1,\dots,2+v+r$), b_{2+v+r}, A_C, Γ_C, R, and S as in Section 21. Define $\mathbf{y}$ as in the real case but with an extra zero entry so that $\mathbf{y}\in\mathbb{Z}^{v+r+1}$. Choose a vector $\mathbf{z}$ in Γ_C close to $\mathbf{y}$ and a number $D>(R^2+S)^{1/2}$.

The two cases into which we divide the solutions of (16) are:

Case 1.

$$A\ge\max\left\{\begin{array}{c}\frac{1}{c_{11}}\left(\log(4\sin^{-1}(1/4)c_{16}(i_0))+\log C-\log((D^2-S)^{1/2}-R)\right),\\ \frac{c_8'+c_9'N_0}{c_{10}-(n-1)c_{11}},\frac{c_8''+c_9''N_0}{c_{10}-c_{11}},\left\lceil\frac{\log(2c_{16}(i_0))}{c_{11}}\right\rceil,1\end{array}\right\}$$

Case 2.

$$A \le \max \left\{ \begin{array}{c} \left\lceil \frac{1}{c_{11}} \left(\log(4 \sin^{-1}(1/4) c_{16}(i_0)) + \log C - \log((D^2 - S)^{1/2} - R) \right) \right\rceil - 1, \\ \left\lceil \frac{c_8' + c_9' N_0}{c_{10} - (n-1)c_{11}} \right\rceil - 1, \left\lceil \frac{c_8'' + c_9'' N_0}{c_{10} - c_{11}} \right\rceil - 1, \left\lceil \frac{\log(2c_{16}(i_0))}{c_{11}} \right\rceil - 1, 0 \end{array} \right\}$$

Here c_{11} is a parameter to be chosen strictly between $0 < c_{11} < c_{10}/(n-1)$. It is advantageous to choose c_{11} in such a way that the upper bound for A in Case 2 is minimized.

Now Lemma 23.1 holds with $\mathbf{x} = A_C[b_1, \ldots, b_{2+v+r}]^T$ and with $(b_1, \ldots, b_{1+v+r})$ replaced by $(b_1, \ldots, b_{2+v+r})$. The method for computing the solutions of (16) in Case 1 having $\left|x - y\theta^{(i_0)}\right| = \min_{1 \le i \le n} \left|x - y\theta^{(i)}\right|$ runs as before, but with the tuple $(b_1, \ldots, b_{2+v+r})$. There are a couple of additional tests that we can perform on the tuple $(b_1, \ldots, b_{2+v+r})$. Namely, we can check that b_{2+v+r} is an even integer (since it should be equal to $2a_0$) and that $|b_{2+v+r}| \le B_{2+v+r}$. The considerations for the choices of D, C, and the W_i are analogous to the $s \ge 3$ case. We note that the choice of weights should be modified as follows. For $i = 2, \ldots, 1+v$, choose W_i so that $\frac{1}{2}W_i$ is the nearest positive integer to $\max_{2 \le j \le 1+v+r} B_j/B_i$. For $i = 2+v, \ldots, 1+v+r$, choose W_i as the nearest positive integer to $\max_{2 \le j \le 1+v+r} B_j/B_i$.

24 The sieving procedure

In this final stage, we have upper bounds $N_1, \ldots, N_v, A_0$ for $n_1, \ldots, n_v$, $A = \max |a_i|$, respectively, that (we hope) are very small. We therefore have $(N_1+1)\cdots(N_v+1)(2A_0+1)^r$ tuples $(n_1, \ldots, n_v, a_1, \ldots, a_r)$ to test as solutions to (16). We can do this by checking whether the coefficients of $\theta^2, \ldots, \theta^{n-1}$ for the right-hand side of (16) are all zero. Since the number of tuples to test is likely very large, the cost of the field arithmetic involved makes this method impractical. We thus prefer the following strategy that uses chiefly modular arithmetic to sieve the set of tuples.

First some set-up. Let q be a rational prime having at least three distinct degree one prime ideals $\mathfrak{q}_1, \mathfrak{q}_2, \mathfrak{q}_3$ in its factorization in $\mathcal{O}_K$. Since $\mathcal{O}_K/\mathfrak{q}_i \cong \mathbb{Z}/q\mathbb{Z}$ for $i = 1, 2, 3$, we can find, for each i, explicit rational integers $m_i, A_i, P_{i1}, \ldots, P_{iv}, E_{i1}, \ldots, E_{ir}$ such that

$$\theta \equiv m_i, \quad \alpha\zeta \equiv A_i, \quad \pi_j \equiv P_{ij} \ \ (j = 1, \ldots, v), \quad \varepsilon_j \equiv E_{ij} \ \ (j = 1, \ldots, r) \qquad (\mathrm{mod}\ \mathfrak{q}_i).$$

It follows from (16) that

$$x - ym_i \equiv A_i P_{i1}^{n_1} \cdots P_{iv}^{n_v} E_{i1}^{a_1} \cdots E_{ir}^{a_r} \pmod{\mathfrak{q}_i} \qquad (i = 1, 2, 3).$$

Note that each of these congruences holds modulo q as well. So combining these congruences with the identity

$$(m_2 - m_3)(x - ym_1) + (m_3 - m_1)(x - ym_2) = (m_2 - m_1)(x - ym_3)$$

yields

$$\begin{aligned}(m_2 - m_3)A_1 P_{11}^{n_1} \cdots P_{1v}^{n_v} E_{11}^{a_1} \cdots E_{1r}^{a_r} + (m_3 - m_1)A_2 P_{21}^{n_1} \cdots P_{2v}^{n_v} E_{21}^{a_1} \cdots E_{2r}^{a_r} \\ \equiv (m_2 - m_1)A_3 P_{31}^{n_1} \cdots P_{3v}^{n_v} E_{31}^{a_1} \cdots E_{3r}^{a_r} \pmod{q}.\end{aligned} \tag{34}$$

Since (34) is an integer congruence, testing the tuple $(n_1, \ldots, n_v, a_1, \ldots, a_r)$ for it is computationally easy.

The sieving procedure runs as follows. We select several primes $q_1, \ldots, q_k$ that each have at least three distinct degree one prime ideals in their factorizations in $\mathcal{O}_K$. For each $q \in \{q_1, \ldots, q_k\}$, we compute the integers m_i, A_i, $P_{i1}, \ldots, P_{iv}$, $E_{i1}, \ldots, E_{ir}$ $(i = 1, 2, 3)$. Each tuple $(n_1, \ldots, n_v, a_1, \ldots, a_r)$ is then tested for the congruences (34) modulo $q = q_1, \ldots, q_k$. Heuristically, we expect that only one out of q random tuples will satisfy (34) modulo q. Thus, it is efficient to check the congruences in decreasing order of the sizes

of the q_j. Moreover, we expect all the non-solutions to be eliminated by the end of the procedure provided we select primes $q_1, \ldots, q_k$ such that $q_1 \cdots q_k$ is at least as large as the number of tuples to be tested. If the tuple survives every congruence test, we check it for (16) and then finally for (4) itself (utilizing (17)). Note the method of checking (16) described above gives the values of x and y. We can also skip the check of (16) and check (4) immediately. To do this, we need to compute the x and y corresponding to the tuple $(n_1, \ldots, n_v, a_1, \ldots, a_r)$. This can be done as follows. Put $\beta = \alpha\zeta\varepsilon_1^{a_1} \cdots \varepsilon_r^{a_r}\pi_1^{n_1} \cdots \pi_v^{n_v}$ and choose two embeddings of K into $\mathbb{C}$: $\theta \mapsto \theta^{(i)}$ $(i = 1, 2)$. By (16),

$$x - y\theta^{(1)} = \beta^{(1)}, \qquad x - y\theta^{(2)} = \beta^{(2)},$$

and so

$$x = \beta^{(1)} - \theta^{(1)}\frac{\beta^{(1)} - \beta^{(2)}}{\theta^{(1)} - \theta^{(2)}}, \qquad y = -\frac{\beta^{(1)} - \beta^{(2)}}{\theta^{(1)} - \theta^{(2)}}.$$

If K is not totally real, we need only look at one embedding. For if $\theta \mapsto \theta^{(1)} \in \mathbb{C} \setminus \mathbb{R}$, then with $\theta^{(2)} = \overline{\theta^{(1)}}$ the above gives

$$x = \beta^{(1)} - \theta^{(1)}\frac{\operatorname{Im}\beta^{(1)}}{\operatorname{Im}\theta^{(1)}}, \qquad y = -\frac{\operatorname{Im}\beta^{(1)}}{\operatorname{Im}\theta^{(1)}}.$$

25 Conclusion

In this thesis, we presented and proved the correctness of an algorithm to completely solve an arbitrary Thue-Mahler equation. We also presented an implementation the algorithm as a computer program using the Magma computer algebra system. The algorithm itself is due essentially to Tzanakis and de Weger [TW2], though our presentation introduced some minor modifications. Our implementation of the algorithm is completely original, and, to the best of our knowledge, it is the first fully general implementation.

There are several potential applications for our Thue-Mahler equation solver (i.e., the program we have written). In [BD], Bennett and Dahmen show that the problem of solving the generalized superelliptic equation

$$F(x, y) = z^l \tag{35}$$

in integers x, y, z, l with $(x, y) = 1$, where $F(x, y)$ is an irreducible binary form over $\mathbb{Z}$, is closely tied to the problem of solving families of Thue-Mahler equations. In particular, they show that, for a large, explicit class of irreducible binary forms $F(x, y) \in \mathbb{Z}[x, y]$ of degree 3, 4, 6, or 12, the equation (35) has at most finitely solutions in integers x, y, z, l with $(x, y) = 1$ and l sufficiently large whenever an associated pair of Thue-Mahler equations have no solutions in integers. Hence, one can use our Thue-Mahler equation solver to prove that many specific generalized superelliptic equations (35) have only finitely many integer solutions when l is large. Moreover, since our Thue-Mahler equation solver makes it possible to solve a large number of Thue-Mahler equations in a reasonable time frame with minimal personal effort on the part of the researcher, it may be useful in further developing the connection between solving generalized superelliptic equations and solving families of Thue-Mahler equations by allowing one to test conjectures and heuristics. Indeed, Bennett and Dahmen use a Thue-Mahler equation solver for degree three equations for this purpose in [BD].

Our Thue-Mahler equation solver can also be applied to the task of determining all elliptic curves over $\mathbb{Q}$ having prescribed conductor. Indeed, the problem of determining all elliptic curves of conductor $N = p_1^{a_1} \dots p_v^{a_v}$ (where the p_i are given rational primes, and the a_i are given nonnegative integers) can generally be reduced to the task of solving finitely many Thue-Mahler equations involving the primes $p_1, \dots, p_v$. Note that finding all the elliptic curves of conductor 11 was the impetus for Agrawal, Coates, Hunt, and van der Poorten to solve the Thue-Mahler equation $x^3 - x^2y + xy^2 + y^3 = \pm 11^z$ using

linear forms in logarithms (see [ACHP]). As we mentioned in Section 1, the method used for solving this equation inspired Tzanakis and de Weger to develop their algorithm. The principal method that has been used to compute elliptic curves of a given conductor is the so-called modular symbol method. In fact, this method has been used by J. Cremona to determine all elliptic curves over $\mathbb{Q}$ of conductor less than 19000 (see [Cr] for a description of Cremona's work for curves with conductor up to 13000). However, it may be that the Thue-Mahler equation method for computing elliptic curves over $\mathbb{Q}$ is superior to the modular symbol method for conductors having certain forms.

Another potential future research direction that stems even more directly from this thesis would be to perform a detailed complexity analysis for the algorithm we presented. Having access to a fully-general implementation of the algorithm will certainly be useful for one wishing to undertake such an analysis.

There are a few improvements that can be made to the algorithm presented in this thesis. For one thing, there are better results available for linear forms in real/complex and p-adic logarithms in the special case of two or three logarithms than the general results of Matveev [Mat] and Yu [Yu2] that we have used (see [BL], [La], [Mi1], and [Mi2]). Using the superior results would produce a smaller upper bound for H in Section 16. This would (potentially) reduce the computation time for the reduction procedures by reducing the total number of iterations of the procedures and by reducing the sizes of the entries in the basis vectors generating the approximation lattices (which would speed up the LLL algorithm and the Fincke-Pohst algorithm). Note, however, that a substantial reduction in relative computation time would be obtained only when the general theorems of Matveev and Yu produce a bound on H that is was very much larger than the bound produced by the theorems for two or three logarithms. Such a large discrepancy in the size of the bounds is unlikely to occur as the general results and the two/three-logarithm results depend on a common set of parameters, and do so in roughly the same way.

Another modification that can be made to the algorithm concerns the choice of approximation lattices in the p_l-reduction procedures, particularly, in the general case described in Section 19. When we choose the index $h \in \{1, \ldots, S\}$ in the general case, we are effectively choosing which coefficient Λ_{lh} of $\Lambda_l = \sum_{h=1}^{S} \Lambda_{lh}\phi^{h-1}$ we will be trying to prove has small p_l-adic valuation. However, what we really want is to prove that Λ_l has small p_l-adic value. (Considering Lemmas 8.1 and 8.2 should make all this clear.) So, since $\mathrm{ord}_{p_l}(\Lambda_l) \geq \min_{1 \leq h \leq S} \mathrm{ord}_{p_l}(\Lambda_{lh})$, and since working with the element Λ_l of $\mathbb{Q}_{p_l}(\phi)$ directly

presents problems when it comes to defining an approximation lattice, we should aim to prove that Λ_{lh} has small p_l-adic valuation for every $h \in \{1, \ldots, S\}$. One way to do this would be to run the whole reduction procedure for each $h \in \{1, \ldots, S\}$ separately. However, a more efficient way is to consider each $h \in \{1, \ldots, S\}$ simultaneously by working, not with the $(v+r)$-dimensional lattice Γ_m generated by the columns of the matrix A_m (see Section 20), but with the $(v+r+s-1)$-dimensional lattice generated by the columns of the matrix

$$A'_m = \begin{bmatrix} W_2 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & W_{\hat{i}-1} & & & & & 0 & & \\ & 0 & & W_{\hat{i}+1} & & & & & & \\ & & & & \ddots & & & & & \\ & & & & & W_{1+v+r} & & & & \\ W_{\hat{i}}\beta_{2,1}^{\{m\}} & \cdots & W_{\hat{i}}\beta_{\hat{i}-1,1}^{\{m\}} & W_{\hat{i}}\beta_{\hat{i}+1,1}^{\{m\}} & \cdots & W_{\hat{i}}\beta_{1+v+r,1}^{\{m\}} & W_{\hat{i}}p_l^m & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots & & \ddots & \\ W_{\hat{i}}\beta_{2,s}^{\{m\}} & \cdots & W_{\hat{i}}\beta_{\hat{i}-1,s}^{\{m\}} & W_{\hat{i}}\beta_{\hat{i}+1,s}^{\{m\}} & \cdots & W_{\hat{i}}\beta_{1+v+r,s}^{\{m\}} & 0 & \cdots & W_{\hat{i}}p_l^m \end{bmatrix},$$

where $\beta_{i,h} = \alpha_{ih}/\alpha_{\hat{i}\hat{h}}$ for $i = 1, \ldots, 1+v+r$ and $h = 1, \ldots, s$, and where $\hat{i}$ and $\hat{h}$ are indices in $\{2, \ldots, 1+v+r\}$ and $\{1, \ldots, s\}$, respectively, for which

$$\mathrm{ord}_{p_l}(\alpha_{\hat{i}\hat{h}}) = \min_{2 \le i \le 1+v+r} \left(\min_{1 \le h \le s} \mathrm{ord}_{p_l}(\alpha_{ih}) \right).$$

We are assuming here, of course, that $l \in I$. With this modification to the approximation lattices, analogs of Lemmas 20.1 and 22.1 are readily established, and both the basic and refined p_l-adic reduction procedures work virtually the same as before (cf. Chapter IV of Smart's book [Sm]). The best modern versions of the LLL algorithm have worst-case running time $O(d^3(d + \log B)(\log B)M(d))$, where d denotes the dimension of the lattice, B denotes an upper bound for the Euclidean norm of the input basis vectors, and $M(k)$ denotes the time required to multiply two k-bit integers (cf. [NS]). One can safely assume $M(k) = O(k^2)$. Since the worst-case running time of the LLL algorithm (and, consequently, of the Fincke-Pohst algorithm) increases as the dimension of the lattice increases, it is not immediately clear that replacing our original approximation lattices with higher-dimensional ones will improve the efficiency of the algorithm. However, because of how m is chosen, an increase in lattice dimension will decrease the expected size of m. So, since m is proportional to the bound B, the running time of the LLL algorithm decreases

as m decreases. Moreover, smaller values of m mean smaller bounds are produced by the reduction procedures, which reduces the total number of iterations of the reduction procedure needed. This rough analysis is clearly not definitive, but, according to Smart [Sm], the overall effect of using the $(v+r+s-1)$-dimensional lattices rather than $(v+r)$-dimensional is that the computational cost of the p_l-reduction procedures is lowered.

Though we expect there are numerous improvements and optimizations that can be made in our implementation of the algorithm, we believe the implementation to be generally good. Improving the implementation of the sieving process is likely the most important improvement that could be made in the program since this process appears to be the principal bottleneck in the algorithm in practice (at least in cases when the degree of the equation and the number of primes and fundamental units is not very large).

References

[ACHP] A. K. Agrawal, J. H. Coates, D. C. Hunt, and A. J. van der Poorten, Elliptic curves of conductor 11, *Mathematics of Computation* **35** (1980), 991-1002.

[BCP] W. Bosma, J. Cannon, and C. Playoust, The Magma algebra system. I. The user language. *Journal of Symbolic Computation* **24** (1997), no. 3-4, 235-265. `http://magma.maths.usyd.edu.au/magma/`

[BD] M. A. Bennett and S. Dahmen, Klein forms and the generalized superelliptic equation, *submitted*.

[BS] Z. I. Borevich and I. R. Shafarevich, *Number theory*, Academic Press, 1966.

[BL] Y. Bugeaud and M. Laurent, Minoration effective de la distance p-adique entre puissances de nombres algébriques (French) [Effective lower bound for the p-adic distance between powers of algebraic numbers], *Journal of Number Theory* **61** (1996), 311-342.

[Ca] J. W. S. Cassels, *Local fields*, Cambridge University Press, 1986.

[Cr] J. Cremona, The elliptic curve database for conductors to 130000, *Algorithmic Number Theory : 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, 11-29, Lecture Notes in Computer Science **4076**, Springer, 2007.

[Coa1] J. H. Coates, An effective p-adic analogue of a theorem of Thue, *Acta Arithmetica* **15** (1969), 279-305.

[Coa2] J. H. Coates, An effective p-adic analogue of a theorem of Thue II: The greatest prime factor of a binary form, *Acta Arithmetica* **16** (1970), 399-412.

[Coh1] H. Cohen, *A course in computational algebraic number theory*, Springer-Verlag, 1995.

[Coh2] H. Cohen, *Number theory volume I: tools and diophantine equations*, Springer Science + Business Media, LLC, 2007.

[FP] U. Fincke and M. Pohst, Improved methods for calculating vectors of short length in a lattice, including a complexity analysis, *Mathematics of Computation* **44** (1985), no. 170, 463-471.

[Ha] H. Hasse, *Number theory*, Springer-Verlag, 1980.

[Ko] N. Koblitz, *p-adic numbers, p-adic analysis, and zeta-functions*, Springer-Verlag, 1977.

[La] M. Laurent, Linear forms in two logarithms and interpolation determinants II, *Acta Arithmetica* **133** (2008), no. 4, 325-348.

[LLL] A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász, Factoring polynomials with rational coefficients, *Mathematische Annalen* **261** (1982), 515-534.

[Mah] K. Mahler, Zur approximation algebraischer zahlen. I., *Mathematische Annalen*, **107** (1933), 691-730.

[Mat] E. M. Matveev, An explicit lower bound for a homogeneous rational linear form in the logarithms of algebraic numbers. II., *Izvestiya: Mathematics*, **64** (2000), no. 6, 1217-1269.

[Mi1] M. Mignotte, Linear forms in two and three logarithms and interpolation determinants, *Diophantine Equations*, 151-166, Tata Institute of Fundamental Research Studies in Mathematics **20**, Tata Institute of Fundamental Research, 2008.

[Mi2] M. Mignotte, A kit on linear forms in three logarithms, *preprint*.

[Na] W. Narkiewicz, *Elementary and analytic theory of algebraic numbers*, 3rd ed., Springer-Verlag, 2004.

[NS] P. Nguyen and D. Stehlé, An LLL algorithm with quadratic time complexity, *SIAM Journal on Computing* **39** (2009), no. 3, 874-903.

[Sm] N.P. Smart, *The algorithmic resolution of diophantine equations*, Chapman and Hall, Cambridge University Press, 1998.

[St] D. Stehlé, Floating-point LLL: theoretical and practical aspects, *The LLL Algorithm, Survey and Application*, Springer-Verlag, 2010.

[ST] I. N. Stewart and D. O. Tall, *Algebraic number theory*, Chapman and Hall, Cambridge University Press, 1979.

[SV] V. G. Sprindžuk and A.I. Vinogradov, The representation of numbers by binary forms (Russian), *Matematicheskie Zametki* **3** (1968), 369-376.

[SW] D. Stehlé and M. Watkins, On the extremality of an 80-dimensional lattice, *Algorithmic Number Theory: 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010, Proceedings*, 340-356, Lecture Notes in Computer Science **6197**, Springer, 2010.

[TW1] N. Tzanakis and B. M. M. de Weger, Solving a specific Thue-Mahler equation, *Mathematics of Computation* **57** (1991), no. 196, 779-815.

[TW2] N. Tzanakis and B. M. M. de Weger, How to explicitly solve a Thue-Mahler equation, *Compositio Mathematica* **84** (1992), 223-288.

[dW1] B. M. M. de Weger, *Algorithms for diophantine equations*, CWI Tract No. 65, Centre for Math and Computer Science, Amsterdam, 1989.

[dW2] B. M. M. de Weger, On the practical solution of Thue-Mahler equations, an outline, *Colloquia Mathematica Societatis János Bolyai*, **51** (1990), 1037-1050.

[Yu1] K. Yu, Linear forms in p-adic logarithms II, *Compositio Mathematica* **74** (1990), 15-113.

[Yu2] K. Yu, P-adic logarithmic forms and group varieties III, *Forum Mathematicum* **19** (2007), 187-280.

Appendix: The implementation in Magma

```
function ThueMahlerSolver(c,p,a)
////////////////////////////////////////////////
//Description of Input
////////////////////////////////////////////////
/*
c is the (integer) sequence of coefficients of F(x,y).
c[i]=c_i from (3). The
coefficient c_0 of x^n is not to be entered.
It is assumed that c_0=1.
It is also assumed that F(x,y) is irreducible.

p is the sequence of rational primes. p[i]=p_i from (3)

a is the integer a from (3).

It is assumed that (a,p_i)=1 for i=1,..,v.

For the example equation solved in [TW2], we would use
c:=[-23,5,24];
p:=[2,3,5,7];
a:=1;
*/

////////////////////////////////////////////
//Description of Output:
////////////////////////////////////////////
/*
A list of solutions [X,Y,z_1,...,z_v] of the Thue-Mahler
equation
*/

////////////////////////////////////////////////////
```

```
//Define the n = degree of g(t) and v = number of primes
////////////////////////////////////////////////////
n:=#c;  //degree of F(t,1)=g(t)
v:=#p;  //number of primes

////////////////////////////////////////////////////
//List of Solutions
////////////////////////////////////////////////////
Solutions:=[]; //to be filled
//SolutionsTest:=[];
//ExceptionalTuplesTest:=[];



//////////////////////////////////////////////
//Define g and compute its discriminant
//////////////////////////////////////////////
Zt<t>:=PolynomialRing(Integers());
g:=t^n+c[n];
for i:=1 to n-1 do
g:=g+c[i]*t^(n-i);
end for;

DiscriminantOfg:=Discriminant(g);

////////////////////////////////////////////////
//Define K and compute number field data
////////////////////////////////////////////////
//define K=Q(theta)
K<theta>:=NumberField(g);

//Compute the ring of integers OK of K
OK:=MaximalOrder(K);

//Compute an integral basis for K
IntegralBasisK:=IntegralBasis(K);
```

```
//For each element of the integral basis for K, compute its
//coefficients on the power basis of K: 1,theta,...,theta^{n-1}
CoefficientsOfIntegralBasisElement:=[];
for i:=1 to n do
CoefficientsOfIntegralBasisElement[i]:=
Eltseq(IntegralBasisK[i]);
end for;

//s = number of real embeddings, 2t = number of complex
//embeddings
s,t:=Signature(K);

//number of fundamental units
r:=s+t-1;

//Compute a set of fundamental units and
//express them as elements in OK in terms of the integral basis
//they are represented by their coefficients on the integral basis
U,psi:=UnitGroup(OK);
eps:=[];
for i:= 1 to r do
eps[i]:=psi(U.(i+1));
end for;
eps[s+t]:=psi(U.1); //generator for units of finite order

//Compute the divisors of the class number of K
DivisorsOfClassNumberOfK:=Divisors(ClassNumber(K));

////////////////////////////////////////////////////////////////////
/*
Compute the splitting field F of g over Q (FFF), the roots of g in F
(rootsofginFFF), and the ring of integers of F (OF)
*/
////////////////////////////////////////////////////////////////////
```

```
FFF,rootsofginFFF:=SplittingField(g);
OF:=MaximalOrder(FFF);

////////////////////////////////////////////////////////////////
/*
Set precision for real/complex and p-adic computations

We want the precision for real/complex computations (realprecision) to
be about four time as large as the largest value of Log(C)/Log(10) (the
number of decimal digits in C)  that we expect will occur in the real
reduction step (Section 19).  C will about the size of H_0^(v+r), so we
want to take realprecision \approx 2*(v+r)*Log(H_0)/Log(10).  (We would
be fine if instead of "four times as large" it was just "larger" than.
We go four times as large as a safety factor to avoid having to back up
and do all the calculations again with an increased precision.)

We want to choose the precision for p_i-adic computations
(padicprecision[i]) is about four times as large as the largest value of
m that we expect will occur in the p_i-adic reduction step (Section 18).
m will be of a size that makes p_i^m \approx H_0^(v+r), so we want to
take padicprecision[i] \approx 2*(v+r)*Log(H_0)/Log(p[i]).  (We would be
fine if instead of "four times as large" it was just "larger" than.  We
go four times as large as a safety factor to avoid having to back up and
do all the calculations again with an increased precision.)

We cannot calculate H_0 a priori.  But we know H_0 will be essentially of
the size of that largest of the constants coming from using Yu's and
Matveev's theorems to bound the relevant linear forms in logarithms.

The constant c_2 \cdot \min{ c_3^{\prime}, c_3^{prime}  } from Yu's
theorem will be bounded by something of the size

(16*Exp(1)*Degree(F))^(2*(2+v+r)) * (1+v+r)^3 * Log(Degree(F)*(1+v+r))
* Max(p[1],...,p[v])^(Degree(F)) * (10)^(1+v+r)
```

```
(see [Yu2] p.190).

The constant c_17 from Matveev's Theorem will be bounded by something of
the size

2^(6*(1+v+r)+20) * Degree(F)^(3+v+r) * Log(Degree(F)) * 10^(1+v+r)

Here we are assuming that the absolute logarithmic heights involved are
<= 10.

We set the up a loop so that, if the precisions are not high enough,
we back up and start again with more precision.
*/
////////////////////////////////////////////////////////////////////////

RealPrecisionMultiplier:=1;
PadicPrecisionMultiplier:=[];
for i:=1 to v do
PadicPrecisionMultiplier[i]:=1;
end for;

for PrecisionLoopVariable:=1 to 5 do

if PrecisionLoopVariable eq 5 then
print("Something is wrong.
The required precision for the computation seems to be too large.");
break PrecisionLoopVariable;
end if;

H0estimate:=Max([
(16*Exp(1)*Degree(FFF))^(2*(2+v+r)) * (1+v+r)^3 *
Log(Degree(FFF)*(1+v+r)) * Max(p)^(Degree(FFF)) * (10)^(1+v+r),
2^(6*(1+v+r)+20) * Degree(FFF)^(3+v+r) * Log(Degree(FFF)) * 10^(1+v+r)
]);
```

```
realprecision:=4*Ceiling((v+r)*Log(H0estimate)/Log(10));
realprecision:=realprecision*RealPrecisionMultiplier;

padicprecision:=[];
for i:=1 to v do
padicprecision[i]:=4*Ceiling((v+r)*Log(H0estimate)/Log(p[i]));
padicprecision[i]:=padicprecision[i]*PadicPrecisionMultiplier[i];
end for;

SetDefaultRealField(RealField(realprecision));
PI:=Pi(RealField());

/*
print("padicprecision");
padicprecision;
print("realprecision");
realprecision;
*/


////////////////////////////////////////////////////////////////////////
/*
For each prime p[i], compute the primes of OK dividing p[i] (pp[i][j]),
their number (m[i]), their ramification indices (e[i][j]), and their
residue degrees (f[i][j]).

For each pp[i][j] with e[i][j]=f[i][j]=1 (i.e. for each unramified degree
one prime ideal above p[i]), compute the least positive integer h[i][j]
such that pp[i][j]^h[i][j] is principal and the generator pi[i][j] of
pp[i][j]^h[i][j].

Also store the indices j of the unramified degree one primes pp[i][j]
above p[i].
*/
////////////////////////////////////////////////////////////////////////
```

```
pp:=[];
m:=[];
e:=[];
f:=[];
IndicesOfUnramifiedDegreeOnePrimesAbovep:=[];
h:=[];
pi:=[];
DecompositionOfp:=[];
for i:=1 to v do
pp[i]:=[];
e[i]:=[];
f[i]:=[];
IndicesOfUnramifiedDegreeOnePrimesAbovep[i]:=[];
h[i]:=[];
pi[i]:=[];
DecompositionOfp[i]:=Decomposition(OK,p[i]);
m[i]:=#DecompositionOfp[i]; //number of distinct prime factors of p[i]
for j:=1 to m[i] do
pp[i][j]:=DecompositionOfp[i][j][1];
e[i][j]:=DecompositionOfp[i][j][2];
f[i][j]:=InertiaDegree(pp[i][j]);
if e[i][j]*f[i][j] eq 1 then
Append(~IndicesOfUnramifiedDegreeOnePrimesAbovep[i],j);
for k:=1 to #DivisorsOfClassNumberOfK do
if IsPrincipal(pp[i][j]^DivisorsOfClassNumberOfK[k]) then
h[i][j]:=DivisorsOfClassNumberOfK[k];
temp,pi[i][j]:=IsPrincipal(pp[i][j]^DivisorsOfClassNumberOfK[k]);
end if;
end for;
else
h[i][j]:=0;            //initialization, never used
pi[i][j]:=eps[s+t];   //initialization, never used
end if;
end for; //end j loop
```

```
end for; //end i loop

////////////////////////////////////////////////
/*
Compute completion of K at each pp[i][j] (Kpp[i][j]) and the embedding of
K into Kpp[i][j] (mKpp[i][j]).  Also, for each pp[i][j], compute the
corresponding factor of g(t) in \QQ_{p[i]}[t] (gp[i][j])

gp[i][j] = g_j(t) from Section 3 with p=p_i
*/
////////////////////////////////////////////////
Kpp:=[];
mKpp:=[];
gp:=[**];
for i:=1 to v do
Kpp[i]:=[];
mKpp[i]:=[**];
gp[i]:=[];
for j:=1 to m[i] do //m[i]:=number of distinct prime factors of p[i]
Kpp[i][j], mKpp[i][j]:=Completion(K,pp[i][j] :
Precision :=padicprecision[i] );
gp[i][j]:=MinimalPolynomial( mKpp[i][j](theta), PrimeField(Kpp[i][j]));
end for;
end for;
/*
LocalFactorization(PolynomialRing(pAdicRing(p[1]
: Precision:=20)) ! g : Ideals:=true);
*/
////////////////////////////////////////////////////////////////////
/*
For each p[i], compute the completion FFFppF of FFF at a prime ideal ppF
of OF.  Also compute the embedding of FFF into FFFppF (mFFFppF).

Note FFFppF contains the splitting field of g over Q_{p[i]}.
```

```
MAGMA currently does not support the direct computation of the splitting
field of g over Q_{p[i]}.  The function that purports to do this fails
for p[i] that ramify in the K.

Note that since F/Q is Galois, every prime ideal of OF above p[i] has the
same ramification index and residue degree
*/
////////////////////////////////////////////////////////////////////
ppF:=[];  //ppF[i] is a prime of FFF above p[i] selected essentially
          //arbitrarily
eF:=[];   //eF[i] is the ramification index of ppF over Q
fF:=[];   //fF[i] is the residue degree of ppF over Q
FFFppF:=[];
mFFFppF:=[**];
SS:=[];    //SS[i] is the degree of FFFppF[i] over Q_p[i]
DecompositionInFFFOfp:=[];

for i:=1 to v do
DecompositionInFFFOfp[i]:=Decomposition(OF,p[i]);
eF[i]:=DecompositionInFFFOfp[i][1][2];
fF[i]:=InertiaDegree(DecompositionInFFFOfp[i][1][1]);
ppF[i]:=DecompositionInFFFOfp[i][1][1];
FFFppF[i],mFFFppF[i]:=Completion(FFF,ppF[i]:Precision:=padicprecision[i]);
SS[i]:=AbsoluteDegree(FFFppF[i]);
end for;  //end i loop


////////////////////////////////////////////////////////////////
/*
Magma's Completion() function will make FFFppF[l] as one the following
types of field extensions:
1. an unramified extension over the p_l-adic field
2. a totally ramified extension over a p_l-adic field
3. an unramified extension over an unramified extension over a p_l-adic
field
```

4. a totally ramified extension over an unramified extension over a p-adic field

We will later want to find the coefficients of elements of FFFppF[l] on the (canonical) power basis of FFFppF[l] over Q_{p_l}. For this, it will be important to know what type of extension FFFppF[l] is. The variable FFFppFType will indicate this.

The first two types are easy to identify and they come as simple extensions, which will let us use MAGMA built-in function for finding the coefficients of elements on the power basis.

The third type seems to be a quirk of Magma. The top extension always has degree 1 relative to the intermediate extension. Moreover, the minimal polynomial of the top extension over the intermediate extension is always x. Finding the coefficients of elements on the power basis will be only slighlty harder than in cases 1. and 2.

For the fourth type, the top extension will always have relative degree greater than one. We will have to work a bit harder to find coefficients of elements here.

First we find an element generator[l] that generates FFFppF[l] over Q_{p_l}. Then we construct a matrix A[l] containing the information necessary to find the coefficients of elements of FFppF[l] on the basis 1,generator[l],...,generator[l]^(d3 - 1), where d3 = degree of FFFpFF[l] over Q_{p_l}. We will actually need the inverse of this matrix, so we take it now.

More explictly, we construct A[l] as follows.
Let \alpha be the generator of FFFpFF[l] over its coefficient field
Let \beta be the generator of the coefficient field of FFFpFF[l] over Q_{p_l}
d1 = degree of FFFpFF[l] over its coefficient field
d2 = degree of the coefficient field over Q_{p_l}

```
d3 = d1*d2 = degree of FFFpFF[l] over Q_{p_l}
Let c_{ijk} be the numbers in Q_{p_l} such that
generator^{k-1} = sum_{i,j} c_{ijk} \alpha^{i-1} \beta^{j-1}
Here i ranges from 1 to d1, j ranges from 1 to d2, k ranges from 1 to d3.
A[l] =
c_{1 1 1} ... c_{1 1 d3}
.                  .
.                  .
.                  .
c_{1 d2 1} ... c_{1 d2 d3}
.                  .
.                  .
.                  .
c_{d1 1 1} ... c_{d1 1 d3}
.                  .
.                  .
.                  .
c_{d1 d2 1} ... c_{d1 d2 d3}


We also define here u_l = (1/2) * \ord_{p_l}(Discr G(t)) from Section 8

u[l]:=(1/2)*Valuation(Discriminant(DefiningPolynomial(
CoefficientField(FFFppF[l]))));
*/
////////////////////////////////////////////////////////////

FFFppFType:=[];
generator:=[**];
Ainverse:=[];
d1:=[];
d2:=[];
d3:=[];
u:=[RationalField() | ];
for l:=1 to v do
```

```
FFFppFType[l]:=5;           //initialize
generator[l]:=FFFppF[l].1;   //initialize, only used in case
                             //FFFppFType[l]=4
Ainverse[l]:=1;                    //initialize
d1[l]:=Degree(FFFppF[l],CoefficientField(FFFppF[l]));
d2[l]:=Degree(CoefficientField(FFFppF[l]),PrimeField(FFFppF[l]));
d3[l]:=Degree(FFFppF[l],PrimeField(FFFppF[l]));
u[l]:=1;

if AbsoluteRamificationIndex(FFFppF[l]) eq 1 then
FFFppFType[l]:=1;
u[l]:=(1/2)*Valuation(Discriminant(DefiningPolynomial(FFFppF[l])));
if Valuation(FFFppF[l].1) lt 0 then print("Error. Generator has negative
valuation.  Results invalid. l="); l; end if;
end if;

if AbsoluteInertiaDegree(FFFppF[l]) eq 1 then
FFFppFType[l]:=2;
/*FFFppF[l];
DefiningPolynomial(FFFppF[l]);
Discriminant(DefiningPolynomial(FFFppF[l]));
Parent((1/2)*Valuation(Discriminant(DefiningPolynomial(FFFppF[l]))));
(1/2)*Valuation(Discriminant(DefiningPolynomial(FFFppF[l])));*/
u[l]:=(1/2)*Valuation(Discriminant(DefiningPolynomial(FFFppF[l])));
if Valuation(FFFppF[l].1) lt 0 then print("Error. Generator has negative
valuation.  Results invalid. l="); l; end if;
end if;

if AbsoluteInertiaDegree(FFFppF[l]) gt 1 and d1[l] eq 1 then
FFFppFType[l]:=3;
u[l]:=(1/2)*Valuation(Discriminant(DefiningPolynomial(
CoefficientField(FFFppF[l]))));
if Valuation(CoefficientField(FFFppF[l]).1) lt 0 then print("Error.
Generator has negative valuation.  Results invalid. l="); l; end if;
end if;
```

```
if d1[l] gt 1 and d2[l] gt 1 then

FFFppFType[l]:=4;

k:=1;
while true do
generator[l] := k*FFFppF[l].1
+ (FFFppF[l] ! CoefficientField(FFFppF[l]).1);
if Degree(MinimalPolynomial(generator[l],PrimeField(FFFppF[l])))
eq d3[l] then
break;
end if;
k +:= 1;
end while;

u[l]:=(1/2)*Valuation(Discriminant(MinimalPolynomial(
generator[l],PrimeField(FFFppF[l]))));

if Valuation(generator[l]) lt 0 then print("Error. Generator has negative
valuation.  Results invalid. l="); l; end if;

//Note: d3[l]:=d1[l]*d2[l];
A:=ZeroMatrix( PrimeField(FFFppF[l]), d3[l], d3[l]); //initialize
temp1:=Coefficients(generator[l]); //initialize
temp2:=Coefficients(temp1[1]); //initialize
for k:=1 to d3[1] do
temp1:=Coefficients(generator[l]^(k-1));
for i:=1 to d1[1] do
temp2:=Coefficients(temp1[i]);
for j:=1 to d2[1] do
A[(i-1)*d2 + j,k]:=temp2[j];
end for;
end for;
end for;
```

```
Ainverse[l]:=A^(-1);

end if;

if FFFppFType[l] eq 5 then print("Error in FFFppFType. l:"); l; end if;
end for; //end l loop


////////////////////////////////////////////////////////////////////////
/*
Define a function to compute the coefficients of an element x in FFFppF[l]
*/
////////////////////////////////////////////////////////////////////////

function GetCoefficients(x,l)
/*
Input: l in {1,...,v}, x = an element of FFFppF[l]
Output:  The coefficients of x on the basis power basis for FFFppF[l]
over Q_{p_l}.
*/
output:=Coefficients(x);
if FFFppFType[l] eq 3 then output:=Coefficients(Coefficient(x,1)); end if;
if FFFppFType[l] eq 4 then
B:=ZeroMatrix( PrimeField(FFFppF[l]), d3, 1);
temp1:=Coefficients(x);
temp2:=Coefficients(temp1[1]); //initialize
for i:=1 to d1[l] do
temp2:=Coefficients(temp1[i]);
for j:=1 to d2[l] do
B[(i-1)*d2[l] + j,1]:=temp2[j];
end for;
end for;
C:=Ainverse[l]*B;
output:=[];
for i:=1 to d3[l] do
```

```
output[i]:=C[i][1];
end for;
end if;
return output;
end function;


//////////////////////////////////////////////////////////////////////
/*
thetap[l][i][j]:= theta_i^{(j)} from Section 3 with p=p_l
*/
//////////////////////////////////////////////////////////////////////
thetap:=[];
for l:=1 to v do
thetap[l]:=[];
for i:=1 to m[l] do
thetap[l][i]:=[**];
temp:=Roots(gp[l][i],FFFppF[l]);
for j:=1 to #temp do             //#temp = degree of gp[l][i] =
                                 //      = e[l][i]*f[l][i]
thetap[l][i][j]:=temp[j][1];
end for;
end for;
end for;

//////////////////////////////////////////////////////////////////
/*
ImageOfIntegralBasisElementp[L][i][j][k] = image of the kth element in
the integral basis for K under the embedding of K into
\overline{\QQ_{p_L}} defined by the map
theta --> thetap[L][i][j]
*/
//////////////////////////////////////////////////////////////////
ImageOfIntegralBasisElementp:=[];
for l:=1 to v do
```

```
ImageOfIntegralBasisElementp[l]:=[];
for i:=1 to m[l] do
ImageOfIntegralBasisElementp[l][i]:=[];
for j:=1 to e[l][i]*f[l][i] do
ImageOfIntegralBasisElementp[l][i][j]:=[**];
for k:=1 to n do
ImageOfIntegralBasisElementp[l][i][j][k]:=0;
for ii:= 1 to n do
ImageOfIntegralBasisElementp[l][i][j][k]:=
ImageOfIntegralBasisElementp[l][i][j][k] +
CoefficientsOfIntegralBasisElement[k][ii]*thetap[l][i][j]^(ii-1);
end for; //ii
end for; //k
end for; //j
end for; //i
end for; //l

////////////////////////////////////////////////////////////////////////
/*
ImageOfpip[L][k][l][i][j] := the image of pi[L][k] under the embedding of
K into \overline{\QQ_{p_l}} defined by the map theta --> thetap[l][i][j].
Recall pi[L][k] is the generator of pp_{Lk}^h[L][k].

ImageOfepsp[L][l][i][j] := the image of eps[L] under the embedding of K
into \overline{\QQ_{p_l}} defined by the map theta --> thetap[l][i][j].
*/
////////////////////////////////////////////////////////////////////////
ImageOfpip:=[];
for L:=1 to v do
ImageOfpip[L]:=[];
for k:=1 to m[L] do
if e[L][k]*f[L][k] eq 1 then
ImageOfpip[L][k]:=[];
for l:=1 to v do
ImageOfpip[L][k][l]:=[];
```

```
for i:=1 to m[l] do
ImageOfpip[L][k][l][i]:=[**];
for j:=1 to e[l][i]*f[l][i] do
ImageOfpip[L][k][l][i][j]:=0;
for ii:= 1 to n do
ImageOfpip[L][k][l][i][j]:=ImageOfpip[L][k][l][i][j]
+ pi[L][k][ii]*ImageOfIntegralBasisElementp[l][i][j][ii];
end for; //ii
end for; //j
end for; //i
end for; //l
else
ImageOfpip[L][k]:=[[[*0*]]];
end if;
end for; //k
end for; //L

ImageOfepsp:=[];
for L:=1 to r do
ImageOfepsp[L]:=[];
for l:=1 to v do
ImageOfepsp[L][l]:=[];
for i:=1 to m[l] do
ImageOfepsp[L][l][i]:=[**];
for j:=1 to e[l][i]*f[l][i] do
ImageOfepsp[L][l][i][j]:=0;
for ii:= 1 to n do
ImageOfepsp[L][l][i][j]:=ImageOfepsp[L][l][i][j]
+ eps[L][ii]*ImageOfIntegralBasisElementp[l][i][j][ii];
end for; //ii
end for; //j
end for; //i
end for; //l
end for; //L
```

```
////////////////////////////////////////////////////////////////////////
/*
thetaC:=
the conjugates of theta as elements in the field of complex numbers
= the roots of the minimal polynomial of theta (i.e. of g) in the field
of complex numbers
*/
////////////////////////////////////////////////////////////////////////
thetaC:=Conjugates(theta);

////////////////////////////////////////////////////////////////
/*
ImageOfIntegralBasisElementC[i][k] = image of the kth element in the
integral basis for K under the embedding of K into \CC defined by the map
theta --> thetaC[i]
*/
////////////////////////////////////////////////////////////////
ImageOfIntegralBasisElementC:=[];
for i:=1 to n do
ImageOfIntegralBasisElementC[i]:=[**];
for k:=1 to n do
ImageOfIntegralBasisElementC[i][k]:=0;
for ii:= 1 to n do
ImageOfIntegralBasisElementC[i][k]:=ImageOfIntegralBasisElementC[i][k]
+ CoefficientsOfIntegralBasisElement[k][ii]*thetaC[i]^(ii-1);
end for; //ii
end for; //k
end for; //i

////////////////////////////////////////////////////////////////////////
/*
ImageOfpiC[L][k][i] := the image of pi[L][k] under the embedding of K
into \CC defined by the map theta --> thetaC[i].  Recall pi[L][k] is the
generator of pp_{Lk}^h[L][k].
```

```
ImageOfepsC[L][i] := the image of eps[L] under the embedding of K into
\CC defined by the map theta --> thetaC[i].
*/
////////////////////////////////////////////////////////////////////////

ImageOfpiC:=[];
for L:=1 to v do
ImageOfpiC[L]:=[];
for k:=1 to m[L] do
if e[L][k]*f[L][k] eq 1 then
ImageOfpiC[L][k]:=[**];
for i:=1 to n do
ImageOfpiC[L][k][i]:=0;
for ii:= 1 to n do
ImageOfpiC[L][k][i]:=ImageOfpiC[L][k][i]
+ pi[L][k][ii]*ImageOfIntegralBasisElementC[i][ii];
end for; //ii
end for; //i
else
ImageOfpiC[L][k]:=[*0*];
end if;
end for; //k
end for;  //L

ImageOfepsC:=[];
for L:=1 to r do
ImageOfepsC[L]:=[**];
for i:=1 to n do
ImageOfepsC[L][i]:=0;
for ii:= 1 to n do
ImageOfepsC[L][i]:=ImageOfepsC[L][i]
+ eps[L][ii]*ImageOfIntegralBasisElementC[i][ii];
end for; //ii
end for; //i
```

```
end for; //L




////////////////////////////////////////////////////////////////////////
/*
Recall F = the splitting field of g over Q

thetaF:= the conjugates of theta as elements in F
= the roots of the minimal polynomial of theta (i.e. of g) in F
*/
////////////////////////////////////////////////////////////////////////
thetaF:=rootsofginFFF;

////////////////////////////////////////////////////////////////
/*
ImageOfIntegralBasisElementF[i][k] = image of the kth element in the
integral basis for K under the embedding of K into F defined by the map
theta --> thetaF[i]
*/
////////////////////////////////////////////////////////////////
ImageOfIntegralBasisElementF:=[];
for i:=1 to n do
ImageOfIntegralBasisElementF[i]:=[**];
for k:=1 to n do
ImageOfIntegralBasisElementF[i][k]:=0;
for ii:= 1 to n do
ImageOfIntegralBasisElementF[i][k]:=ImageOfIntegralBasisElementF[i][k]
+ CoefficientsOfIntegralBasisElement[k][ii]*thetaF[i]^(ii-1);
end for; //ii
end for; //k
end for; //i

////////////////////////////////////////////////////////////////////////
```

```
/*
ImageOfpiF[L][k][i] := the image of pi[L][k] under the embedding of K
into F defined by the map theta --> thetaF[i].
Recall pi[L][k] is the generator of pp_{Lk}^h[L][k].

ImageOfepsF[L][i] := the image of eps[L] under the embedding of K into F
defined by the map theta --> thetaF[i].
*/
////////////////////////////////////////////////////////////////////////

ImageOfpiF:=[];
for L:=1 to v do
ImageOfpiF[L]:=[];
for k:=1 to m[L] do
if e[L][k]*f[L][k] eq 1 then
ImageOfpiF[L][k]:=[**];
for i:=1 to n do
ImageOfpiF[L][k][i]:=0;
for ii:= 1 to n do
ImageOfpiF[L][k][i]:=ImageOfpiF[L][k][i]
+ pi[L][k][ii]*ImageOfIntegralBasisElementF[i][ii];
end for; //ii
end for; //i
else
ImageOfpiF[L][k]:=[*0*];
end if;
end for; //k
end for;  //L

ImageOfepsF:=[];
for L:=1 to r do
ImageOfepsF[L]:=[**];
for i:=1 to n do
ImageOfepsF[L][i]:=0;
for ii:= 1 to n do
```

```
ImageOfepsF[L][i]:=ImageOfepsF[L][i]
+ eps[L][ii]*ImageOfIntegralBasisElementF[i][ii];
end for; //ii
end for; //i
end for; //L



////////////////////////////////////////////////////////////////////////
/*
Compute the constant constant c10 = c_{10} from Section 12

Note: If kappa = {k_1,...,k_r} with 1 <= k_1 < ... < k_r <= s+t, then
there is a unique index k* in {1,...,s+t}-{k_1,...,k_r} and U_kappa from
Section 12 is equal to the matrix formed by starting with the r by s+t
matrix.
U = (log|\eps_j^{(i)}|) and removing the k*th row.

We use this equivalent definition of U_kappa below.
*/
////////////////////////////////////////////////////////////////////////

//Function to compute maximum absolute row sum of an r by r matrix U
function MaximumAbsoluteRowSum(U,r)
max:=0;
for i:=1 to r do
sum:=0;
for j:=1 to r do
sum:=sum+Abs(U[i][j]);
end for;
if sum gt max then max:=sum; end if;
end for;
return max;
end function;
```

```
U:=ZeroMatrix(ComplexField(),s+t,r);
for i:=1 to s+t do
for j:=1 to r do
U[i][j]:=Log(Abs(ImageOfepsC[j][i]));
end for;
end for;

currentmin:=MaximumAbsoluteRowSum( RemoveRow(U,1)^(-1),r );
for skippedindex:=1 to s+t do  //skipped index = k*
potentialmin:=MaximumAbsoluteRowSum( RemoveRow(U,skippedindex)^(-1),r );
if potentialmin lt currentmin then
currentmin:=potentialmin;
end if;
end for;

c10:=1/currentmin;


//Now we go essentially in the order of the thesis starting at Section 6.

////////////////////////////////////////////////////////////////
//Apply Prime Ideal Removing Lemma
////////////////////////////////////////////////////////////////
/*
The elements of ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] will
be of the form [[k],aaa].

If k <= m[L], then k has the meaning that pp[L][k] is the unconstrained
prime above p[L] (pp[L][k] must have e[L][k]=f[L][k]=1).

If k = m[L]+1, there is no prime above p[L] with ramification index and
residue degree equal to one, hence no unconstrained prime above p[L].

In either case, aaa is a tuple of nonnegative integers such that that
PowerProduct(pp[L],aaa) is the factor of bb=\mathfrak{b} consisting of
```

```
primes above p[L].

Note aaa[k]=0 if k <= m[L].

*/
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////
/*
Define the procedures we need
*/
////////////////////////////////////////////////////////////////////////
procedure BuildListOfValidTuplesWhenppLkIsTheUnconstrainedPrimeAbovepL(
i,L,k,~aaa,~D,~m,~CC,~LIST)
for l:=0 to D[i] do
aaa[i]:=l;
if i eq m[L] then
//aaa[k]:=0;
Include(~LIST,[[k],aaa]);
else
for j:=i+1 to m[L] do
if aaa[i] gt CC[i][j] then D[j]:=Min(D[j],CC[i][j]); end if;
end for;
BuildListOfValidTuplesWhenppLkIsTheUnconstrainedPrimeAbovepL(
i+1,L,k,~aaa,~D,~m,~CC,~LIST);
end if;
end for;
end procedure;

procedure BuildListOfValidTuplesWhenNoUnramifiedDegreeOnePrimesAbovepL(
i,L,k,~aaa,~D,~m,~CC,~LIST)
for l:=0 to D[i] do
aaa[i]:=l;
if i eq m[L] then
```

```
Include(~LIST,[[k],aaa]);
else
for j:=i+1 to m[L] do
if aaa[i] gt CC[i][j] then D[j]:=Min(D[j],CC[i][j]); end if;
end for;
BuildListOfValidTuplesWhenNoUnramifiedDegreeOnePrimesAbovepL(
i+1,L,k,~aaa,~D,~m,~CC,~LIST);
end if;
end for;
end procedure;

//////////////////////////////////////////////////////////////////////
//The algorithm
//////////////////////////////////////////////////////////////////////
ListOfTuplesOfExponentsForPrimesDividingbbAbovep:=[**];
for L:=1 to v do
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]:=[];


////////////////////////////////////////////////////////
/*
CC[i][j]:= \max\cbr{e_i,e_j} \cdot
\min_{k,l} \cbr{ord_p(theta_{i}^{(k)} - theta_{j}^{(l)})}
*/
////////////////////////////////////////////////////////
CC:=[];
for i:=1 to m[L] do
CC[i]:=[];
for j:=1 to i-1 do
temp:=[];
for k:=1 to #thetap[L][i] do
for l:=1 to #thetap[L][j] do
Append(~temp, Valuation( thetap[L][i][k] - thetap[L][j][l] ));
end for;
end for;
```

```
CC[i][j]:=Max([e[L][i],e[L][j]])*Min(temp);
end for;
end for;

for i:=1 to m[L] do
if e[L][i]*f[L][i] eq 1 then
CC[i][i]:=2^15; //placeholder, never used
else
temp:=[];
for k:=1 to #thetap[L][i] do
for l:=k+1 to #thetap[L][i] do
Append(~temp,Valuation( thetap[L][i][k] - thetap[L][i][l] ) );
end for;
end for;
CC[i][i]:=e[L][i]*Min(temp);
end if;
end for;

for i:=1 to m[L] do
for j:=i+1 to m[L] do
CC[i][j]:=CC[j][i];
end for;
end for;

DD:=Floor((1/2)*Max(e[L])*Valuation(DiscriminantOfg, p[L]));
if m[L] gt 1 then
DD:=0;
for i:=1 to m[L] do
for j:=i+1 to m[L] do
if CC[i][j] gt DD then DD:=CC[i][j]; end if;
end for;
end for;
end if;

//////////////////////////////////////////////////////
```

```
/*
For each unramified degree 1 prime pp[L][k] above p[L], list all valid
tuples assuming pp[L][k] is the unconstrained prime ideal above p[L]
dividing (x-y\theta).

The assumption means that, for i=1,...,m[L],i \neq k, we assume the
exponent of the prime ideal pp[L][i] above p[L] in the factorization of
(x-y\theta) is \leq (1/2)*max(e[L][1],...,e[L][m[L]])*ord_p(D_g).  We
assume nothing about the exponent ord_pp[L][k](x-y\theta) of pp[L][k] in
the factorization of (x-y\theta).

For i=1,...,m[L],i \neq k, D[i] is an upper bound on the exponent of the
prime ideal pp[L][i] above p[L] in the factorization of bb=\mathfrak{b}

The exponent of pp[L][k] in the factorization of (x-y\theta) is
ord_pp[L][k](x-y\theta) (obviously). Suppose A < B.  The valid tuples aaa
when ord_pp[L][k](x-y\theta) = B are a subset of the valid tuples aaa
when ord_pp[L][k](x-y\theta) = A.  So we only need to consider
ord_pp[L][k](x-y\theta) <= DD+1 to find all valid tuples in when
ord_pp[L][k](x-y\theta) >= DD+1 and when ord_pp[L][k](x-y\theta) <= DD.
Hence the definition of D[k] below.
*/
//////////////////////////////////////////////////////

for kk:=1 to #IndicesOfUnramifiedDegreeOnePrimesAbovep[L] do
k:=IndicesOfUnramifiedDegreeOnePrimesAbovep[L][kk];
//k will go through all the indices of primes ideals above p=p[L] with
//e_i=f_i=1

D:=[];

for i:=1 to k-1 do
if e[L][i]*f[L][i] eq 1 then
D[i]:=DD;
else
```

```
D[i]:=Min(CC[i][i],DD);
end if;
end for;

D[k]:=DD+1;

for i:=k+1 to m[L] do
if e[L][i]*f[L][i] eq 1 then
D[i]:=DD;
else
D[i]:=Min(CC[i][i],DD);
end if;
end for;

/*
Populate ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] assuming
pp[L][k]is the unconstrained prime ideal above p[L] dividing (x-y\theta)
*/
aaa:=[];
BuildListOfValidTuplesWhenppLkIsTheUnconstrainedPrimeAbovepL(
1,L,k,~aaa,~D,~m,~CC,
~ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]);


/*
Remove redundant tuples from
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]
*/
CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL:=
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L];
i1:=1;
while i1 le #CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL do
bbb:=CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL[i1];
if bbb[1][1] ne k then i1:=i1+1; continue; end if;
IndexOfTupleToRemove:=-1;
```

```
for i2:=1 to #ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] do
ccc:=ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i2];

if ccc[1][1] ne k then continue i2; end if;
for i:=1 to k-1 do
if bbb[2][i] ne ccc[2][i] then continue i2; end if;
end for; //end i loop
for i:=k+1 to #bbb[2] do
if bbb[2][i] ne ccc[2][i] then continue i2; end if;
end for; //end i loop
if bbb[2][k] le ccc[2][k] then continue i2; end if;
//know we know that bbb covers ccc
IndexOfTupleToRemove:=i2;
break i2;

end for;
if IndexOfTupleToRemove ne -1 then
Remove(~ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
IndexOfTupleToRemove);
else
i1:=i1+1;
end if;
end while;
/*
Done removing tuples
*/



end for; //end kk loop



/*
Remove more redundant tuples from
```

```
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]
*/
CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL:=
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L];
i1:=1;
while i1 le #CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL do
bbb:=CopyOfListOfTuplesOfExponentsForPrimesDividingbbAbovepL[i1];
if bbb[2][bbb[1][1]] ne DD+1 then i1:=i1+1; continue; end if;
IndexOfTupleToRemove:=-1;
for i2:=1 to #ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] do
ccc:=ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i2];

if ccc[2][ccc[1][1]] eq DD+1 then continue i2; end if;
//now we know ccc[2][ccc[1][1]] le DD
for i:=1 to bbb[1][1]-1 do
if bbb[2][i] ne ccc[2][i] then continue i2; end if;
end for;
for i:=bbb[1][1]+1 to #bbb[2] do
if bbb[2][i] ne ccc[2][i] then continue i2; end if;
end for;
//know we know that bbb covers ccc
IndexOfTupleToRemove:=i2;
break i2;

end for;
if IndexOfTupleToRemove ne -1 then
Remove(~ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
IndexOfTupleToRemove);
else
i1:=i1+1;
end if;
end while;

/*
Go through and set aaa[k]=0 for each [[k],aaa] in
```

```
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]
*/
for i1:=1 to #ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] do
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i1][2][

ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i1][1][1]

]:=0;
end for;




//////////////////////////////////////////////////////////////////////
/*
List all valid tuples in the case there are no unramified degree one
primes above p[L].

For i=1,...,m[L], D[i] is an upper bound on the exponent of the prime
ideal pp[L][i] above p[L] in the factorization of bb=\mathfrak{b}

Should remove p[L] from consideration in certain things below.
*/
//////////////////////////////////////////////////////////////////////

if #IndicesOfUnramifiedDegreeOnePrimesAbovep[L] eq 0 then

k:=m[L]+1;

D:=[];

for i:=1 to m[L] do
if e[L][i]*f[L][i] eq 1 then
D[i]:=DD;
else
```

```
D[i]:=Min(CC[i][i],DD);
end if;
end for;

//Populate ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]
aaa:=[];
BuildListOfValidTuplesWhenNoUnramifiedDegreeOnePrimesAbovepL(
1,L,k,~aaa,~D,~m,~CC,
~ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]);

end if;


////////////////////////////////////////////////////////////

end for; //end L loop

////////////////////////////////////////////////////////////
//END Application of Prime Ideal Removing Lemma
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//Build ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep
////////////////////////////////////////////////////////////
/*
For each p[L], if there are unramifed degree one prime ideals of OK above
p[L], we augment ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] by
taking a given choice of unbounded prime p[L][k] and allowing for all the
possible values of the parameter s_{Lk}=s[L][k], which are
0,...,h[L][k]-1 (cf. Section 6)

More precisely, for L = 1,...,v we do the following:
For each element [[k],aaa] of
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
we form the elements [[k],aaa,[0]],...,[[k],aaa,[h[L][k]-1]] and adjoin
```

```
them to a new list
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[L].

If there are no unramifed degree one prime ideals of OK above p[L], then
for each element [[k],aaa] of
ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
we form the element [[k],aaa,0] and adjoin them to
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[L].

As a minor optimization, all this could be done as part of building
ListOfTuplesOfExponentsForPrimesDividingbbAbovep.
*/
//////////////////////////////////////////////////////
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep:=[**];
for L:=1 to v do
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[L]:=[];
for i:=1 to #ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L] do
k:=ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i][1][1];
if k ne m[L]+1 then
for s:=0 to h[L][k]-1 do
Append(~ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
Append(ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i],[s]));
end for;
else //k eq m[L]+1
Append(~ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[L],
Append(ListOfTuplesOfExponentsForPrimesDividingbbAbovep[L][i],[0]));
end if;
end for;
end for;

delete ListOfTuplesOfExponentsForPrimesDividingbbAbovep;

////////////////////////////////////////////////////////////////////
/*
Compute
```

```
ListOfIdealsOfNorma = list containing a representation of each ideal in
OK having norm a.

The representation is such that the ideal corresponding to
ListIdealsOfNorma[i]ideals is PowerProduct(pI,ListIdealsOfNorma[i]),
where pI is the sequence of the prime ideals of OK that divide a.

We follow the strategy layed out in the Example on p.125 of [ST]
*/
//////////////////////////////////////////////////////////////////////
procedure BuildListOfIdealsOfNorma(k,~bbb,~epI,~NpI,~ListOfIdealsOfNorma)
for j:=0 to epI[k] do
bbb[k]:=j;
if k eq #epI then
prod:=1;
for i:=1 to #epI do
prod:=prod*NpI[i]^bbb[i];
end for;
if prod eq a or prod eq -a then Append(~ListOfIdealsOfNorma,bbb); end if;
else
BuildListOfIdealsOfNorma(k+1,~bbb,~epI,~NpI,~ListOfIdealsOfNorma);
end if;
end for;
end procedure;

ListOfIdealsOfNorma:=[];
if a eq 1 or a eq -1 then
pI:=[ideal<OK|1>];
ListOfIdealsOfNorma:=[[1]];
else
//I = ideal<OK|a> = principal ideal generated by a
FI:=Factorization(ideal<OK|a>);
pI:=[];
epI:=[];
NpI:=[];
```

```
for i:=1 to #FI do
pI[i]:=FI[i][1]; //ith prime ideal in the factorization of I
epI[i]:=FI[i][2]; //ramification index of pI[i]
NpI[i]:=Norm(FI[i][1]);
end for;
bbb:=[];
BuildListOfIdealsOfNorma(1,~bbb,~epI,~NpI,~ListOfIdealsOfNorma);
end if;


////////////////////////////////////////////////
/*
Form the list of all units of finite order \zeta in OK (i.e, the list of
all roots of unity \zeta in OK) with the understanding that we include
only one of \zeta and -\zeta.
*/
////////////////////////////////////////////////
ListOfUnitsOfFiniteOrder:=[eps[s+t]];
zeta:=eps[s+t]^2;
while(zeta ne eps[s+t]) do
Append(~ListOfUnitsOfFiniteOrder,zeta);
zeta:=zeta*eps[s+t];
end while;
CopyOfListOfUnitsOfFiniteOrder:=ListOfUnitsOfFiniteOrder;
NumberOfUnitsOfFiniteOrder:=#ListOfUnitsOfFiniteOrder;
for i:=1 to NumberOfUnitsOfFiniteOrder do
if CopyOfListOfUnitsOfFiniteOrder[i] in ListOfUnitsOfFiniteOrder
and -CopyOfListOfUnitsOfFiniteOrder[i] in ListOfUnitsOfFiniteOrder then
Exclude(~ListOfUnitsOfFiniteOrder,-CopyOfListOfUnitsOfFiniteOrder[i]);
end if;
end for;

delete CopyOfListOfUnitsOfFiniteOrder;
delete NumberOfUnitsOfFiniteOrder;
```

```
/*
T:=TorsionUnitGroup(K);
a:=[];
for u in T do
Append(~a,u);
end for;
a;
*/

////////////////////////////////////////////////////////////////////////
/*
Construct
ListOfCases = the list of all cases of the factorized Thue-Mahler
equation (11) that we need to solve.

Each element in ListOfCases will be a sequence.  Consider ListOfCases[i].
For L = 1 to v, ListOfCases[i][L] is of the form [[k],aaa,[s]].  The
meaning here is that pp[L][k] is the unbounded prime above p[L],
PowerProduct(pp[L],aaa) is the factor of bb consisting of primes above
pp[L] and s is the value of the parameter s_{kl}.

ListOfCases[i][v+1] has the meaning that the value of the parameter aa is
PowerProduct(pI,ListOfCases[i][v+1]).

Finally, ListOfCases[i][v+2] is the value of the parameter \zeta
*/
////////////////////////////////////////////////////////////////////////
procedure BuildListOfCases(L,~bbb,~LIST,~ListOfCases)
for i:=1 to #LIST[L] do
bbb[L]:=i;
if L eq v+2 then
temp:=[**];
for j:=1 to v+2 do
temp[j]:=LIST[j][bbb[j]];
end for;
```

```
Append(~ListOfCases,temp);
else
BuildListOfCases(L+1,~bbb,~LIST,~ListOfCases);
end if;
end for;
end procedure;

////////////////////////////////////////////////////////////
//preprocessing
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[v+1]:=
ListOfIdealsOfNorma;
ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep[v+2]:=
ListOfUnitsOfFiniteOrder;

//Populate the list of cases
bbb:=[];
ListOfCases:=[];
BuildListOfCases(1,~bbb,
~ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep,~ListOfCases);

delete ModifiedListOfTuplesOfExponentsForPrimesDividingbbAbovep;

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////
/*
Now we are ready to start the loop through the cases.
We first declare the procedures that we will need in the loop.
We also initialize some variables that we will use in the loop.  We
initialize them now to avoid having to add additional inputs for the
procedures.
*/
////////////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////

function MaxAbsLog(a)
/*
Input: a = number field element
Output: The maximum of the numbers |Log(a^{(1)})|,..., =|Log(a^{(n)})|,
where the a^{(i)} are the conjugates of a in \CC and Log denotes the
principal branch of the complex logarithm
*/
minpoly:=MinimalPolynomial(a,IntegerRing());
ConjugatesOfa:=Roots(PolynomialRing(ComplexField()) ! minpoly);
deg:=Degree(minpoly);
max:=Modulus(Log(ConjugatesOfa[1][1]));
for i:=2 to deg do
temp:=Modulus(Log(ConjugatesOfa[i][1]));
if temp gt max then max:=temp; end if;
end for;
return max;
end function;

////////////////////////////////////////////////////////////////

function SmallestNonnegativeRemainderModpLToThem(x,l,m)
/*
Input: l = integer in {1,...,v}, m = positive integer,
x = an element in the p_l-adic field Q_{p_l} that belongs to the subring
Z_{p_l} (the ring of p_l-adic integers in Q_{p_l}).
Output: The unique integer x^{m} in [0,p_l^m - 1] with
ord_{p_l}(x - x^{m}) >= m
*/
y:=pAdicRing(p[l] : Precision:=padicprecision[l]) ! x;
y:=pAdicQuotientRing(p[l], m) ! y;
y:=IntegerRing() ! y;
```

```
if y lt 0 then y:=y+p[l]^m; end if;
return y;
end function;

////////////////////////////////////////////////////////////////

function LengthOfVector(v)
/*
Input: v = n by 1 column matrix
Output: The length of v = the square root of the sum of the squares of
the entries of v.
*/
size:=NumberOfRows(v);
length:=0;
for i:=1 to size do
length +:= v[i][1]*v[i][1];
end for;
length:=length^(1/2);
return length;
end function;

////////////////////////////////////////////////////////////////////

function pAdicLog(x,p)
/*
Input: p = rational prime, x = p-adic unit belonging to a finite
extension of Q_p
Output: the p-adic logarithm of x
*/
e:=AbsoluteRamificationIndex(Parent(x));
f:=AbsoluteInertiaDegree(Parent(x));
CandidateOrders:=Divisors(p^f - 1);

o:=1;
for oo in CandidateOrders do
```

```
if Valuation(x^oo - 1) gt 0 then o:=oo; break; end if;
end for;
j:=1;
while p^j le e do
j +:= 1; //j:=j+1;
end while;

return Log( x^(o*p^j) )/(o*p^j);
end function;

////////////////////////////////////////////////////////////////////////

function DistanceToNearestInteger(x)
return Min( [x-Floor(x), Ceiling(x)-x] );
end function;

////////////////////////////////////////////////////////////////////////

function RoundP(x)
/*
Input: x = a real number
Ouput: The nearest positive integer to x.  If two postive integers are
the same distance to x, take the largest of the two.
*/
y:=Round(x);
if y eq 0 then y:=1; end if;
return y;
end function;

////////////////////////////////////////////////////////////////////////

procedure Choosettt(~yyy,~ttt,~potentialttt,~Floorsss,~min,~Bm,~mm,k)
/*
Assumption:  sss is a mm by 1 column vector with real entries
Input: Floorsss is a mm by 1 column vector with Floorsss[i][1]=
```

```
Floor(sss[i][1])
Input: yyy is a given mm by 1 vector with real entries
Input: Bm is a given mm by mm matrix with real entries
Input: potentialttt is a mm by 1 vector used as a placeholder by this
procedure
ttt is a mm by 1 column vector
Input: min is a real number
Assumption: This procedure calls itself.  When it is called by the user,
we assume min is equal to the length of the column vector
Bm*Floorsss - yyy
Input: k is a postive integer
Assumption: This procedure calls itself.  When this procedure is called
by the user, we assume k=1.
Result:  If the three assumptions above hold, then, after this procedure
is completed ttt, will be a mm by 1 column vector with integer entries
such that
|ttt[i][1] - sss[i][1]| <= 1
and such that the length of Bm*ttt - yyy is minimal
(among all choices of ttt with integral and |ttt[i][1] - sss[i][1]| <= 1).
Also, min will be the length of Bm*ttt - yyy.
*/
if k eq mm+1 then

potentialmin:=LengthOfVector(Bm*potentialttt - yyy);
if potentialmin lt min then
ttt:=potentialttt;
min:=potentialmin;
end if;

else

for j:=0 to 1 do
potentialttt[k][1]:=Floorsss[k][1]+j;
Choosettt(~yyy,~ttt,~potentialttt,~Floorsss,~min,~Bm,~mm,k+1);
end for;
```

```
end if;

end procedure;

////////////////////////////////////////////////////////////////////////

procedure CongruenceTest(~bbb,hhh,~passes,~Q,~J)
/*
Input: bbb is a sequence of #JJJ-1 integers.
Input: hhh is a positive integer
Assumption: Q[hhh] is a sequence of #JJJ+1 elements of the finite field
Z/qZ, where q is a rational prime.
Input: A boolean variable
Assumption: bbb[i] = b_{JJJ[1+i]}, i=1 to #JJJ-1
Note: bbb[i] = b_{JJJ[1+i]} = b_{1+J[i]}=n_{J[i]} for i:=1 to #J
Note: bbb[#J+i] = b_{JJJ[1+#J+i]} = b_{1+v+i}=a_i for i:=1 to r
Assumption: q has at least three residue degree one prime ideals of OK
above it. Assumption: For i=1,2,3,
Qh[i]=[P_{i,JJJ[2]-1}, ..., P_{i,JJJ[1+#J]-1}, E_i1, ..., E_ir, A_i, m_i]
     =[P_{i,J[1]}, ..., P_{i,J[#J]}, E_i1, ..., E_ir, A_i, m_i]
where m_i, A_i, P_ij, E_ij are the numbers from Section 22 corresponding
to three residue degree one prime ideals of OK above q.
Assumption: passes = true on entry to this procedure
Result:  Provided the assumptions hold, passes will have the value true
if b_{JJJ[i+1]} satisfies the congruence (25) (with those Pij,n_j with j
not in J removed) and will have the value false otherwise.
*/
prod1:=Q[hhh][1][#J+r+1];
for ii:=1 to #J+r do
prod1 *:= Q[hhh][1][ii]^bbb[ii];
end for;

prod2:=Q[hhh][2][#J+r+1];
for ii:=1 to #J+r do
```

```
prod2 *:= Q[hhh][2][ii]^bbb[ii];
end for;

prod3:=Q[hhh][3][#J+r+1];
for ii:=1 to #J+r do
prod3 *:= Q[hhh][3][ii]^bbb[ii];
end for;

if not IsZero(
(Q[hhh][2][#J+r+2]-Q[hhh][3][#J+r+2])*prod1 +
(Q[hhh][3][#J+r+2]-Q[hhh][1][#J+r+2])*prod2 +
(Q[hhh][1][#J+r+2]-Q[hhh][2][#J+r+2])*prod3
) then
passes:=false;
end if;

end procedure;

////////////////////////////////////////////////////////////////////////

procedure ThueMahlerEquationTest(~bbb,~Solutions,~ImageOfzetaC,
~ImageOfalphaC,~J,~ValueOfn,~kk,~hh,~ss,~tt);
/*
Use the tuple bbb (where bbb[i]=b_{JJJ[1+i]}, i=1 to #JJJ-1) and the
sequence ValueOfn to get the tuple bb (where bb[i] = b_{1+i}=n_i, i=1
to v).

Compute the corresponding X,Y.  We can actually compute X,Y before the
getting bb because \pi_{l}^{n_{l}} for l not in J is already part of
\alpha.  We do this.

Test if (X,Y,b_{2},...,b_{1+v+r}) gives solution of the Thue-Mahler
equation (1) and record it if so.

Note: bbb[i]=b_{JJJ[1+i]} = b_{1+J[i]}=n_{J[i]}=bb[J[i]] for i:=1 to #J
```

```
Note: bbb[#J+i]=b_{JJJ[1+#J+i]} = b_{1+v+i}=a_i=bb[v+i] for i:=1 to r
*/

//Calculate X,Y
if t gt 0 then

BETA1:= ImageOfzetaC[s+1] * ImageOfalphaC[s+1];
for i:=1 to #J do
BETA1 *:= ImageOfpiC[J[i]][kk[J[i]]][s+1]^bbb[i];
end for;
for i:=1 to r do
BETA1 *:= ImageOfepsC[i][s+1]^bbb[#J+i];
end for;
THETA1:=thetaC[s+1];

Y:= -Imaginary(BETA1)/Imaginary(THETA1);
X:=Round(  Real( BETA1 + THETA1*Y )  );
Y:=Round(Y);

else //t=0

BETA1:= ImageOfzetaC[1] * ImageOfalphaC[1];
for i:=1 to #J do
BETA1 *:= ImageOfpiC[J[i]][kk[J[i]]][1]^bbb[i];
end for;
for i:=1 to r do
BETA1 *:= ImageOfepsC[i][1]^bbb[#J+i];
end for;
THETA1:=thetaC[1];

BETA2:= ImageOfzetaC[2] * ImageOfalphaC[2];
for i:=1 to #J do
BETA2 *:= ImageOfpiC[J[i]][kk[J[i]]][2]^bbb[i];
end for;
for i:=1 to r do
```

```
BETA2 *:= ImageOfepsC[i][2]^bbb[#J+i];
end for;
THETA2:=thetaC[2];

Y:= Real(-(BETA1 - BETA2)/(THETA1 - THETA2));
X:=Round(Real(BETA1 + Y*THETA1));
Y:=Round(Y);

end if;

//Get the tuple bb from bbb and ValueOfn
bb:=[];
for i:=1 to v do
bb[i]:=ValueOfn[i]; //initialize
end for;
for i:=1 to #J do
bb[J[i]]:=bbb[i];
end for;
/*for i:=1 to r do
bb[v+i]:=bbb[#J+i];
end for;*/


//Build RHS and LHS expressions
RHSexpression:=a;
for i:=1 to v do
RHSexpression *:= p[i]^(bb[i]*hh[i] + ss[i] + tt[i]);
end for;
LHSexpression:= X^n + c[n]*Y^n;
for i:=1 to n-1 do
LHSexpression:=LHSexpression + c[i]*X^(n-i)*Y^i;
end for;

/*test if (X,Y,bb[1],...,bb[v])=(X,Y,b_{2},...,b_{1+v}) gives a solution
of the sign-relaxed Thue-Mahler equation (in the form |LHS|-|RHS| = 0) */
```

```
if IsZero(Abs(LHSexpression) - Abs(RHSexpression)) and Gcd(X,Y) eq 1 then
temp:=[X,Y];
for i:=1 to v do
Append( ~temp, bb[i]*hh[i] + ss[i] + tt[i] );
end for;
Include(~Solutions,temp);
end if;

end procedure;

/////////
procedure SieveBox(~bbb,~Solutions,~count,~Blower,~Bupper,~Q,~J,
~JJJ,~ValueOfn,~kk,~hh,~ss,~tt,~ImageOfzetaC,~ImageOfalphaC,k)
/*
This procedure calls itself.

When this procedure is called by the user, we assume that k=1 and
Solutions is a (possibly empty) list of solutions for the Thue-Mahler
equation.

After this procedure is finished executing its call by the user,
Solutions will contain all the solutions (X,Y,n_1,...,n_v,a_1,...,a_r) of
the sign-relaxed version of the Thue-Mahler equation that satisfy the
currently known bounds on the n_i and the a_i and that satisfy the
current case of (11).  Note that these bounds miss finitely many
explicitly known exceptional tuples.

Recall:  Every solution of the Thue-Mahler equation satisfies some case
of (11).
*/

if k gt #J+r then //done k-1 loops and k-1 >= #J+r loops. So the tuple is
                  //built

//Test the tuple bbb for the congruences (25)
```

```
passes:=true;
for hhh:=1 to #Q do
//test the tuple for the q=q_hhh congruence (25)
CongruenceTest(~bbb,hhh,~passes,~Q,~J);
if passes eq false then break hhh; end if;
end for;

if passes eq true then

/* Use the tuple bbb (where bbb[i]=b_{JJJ[1+i]}, i=1 to #JJJ-1) to get
the tuplebb (where bb[i] = b_{1+i}, i=1 to v).  Compute the corresponding
X,Y. Test if (X,Y,b_{2},...,b_{1+v}) gives a solution of the sign-relaxed
version of the Thue-Mahler equation (1) and record it if so. */
ThueMahlerEquationTest(~bbb,~Solutions,~ImageOfzetaC,~ImageOfalphaC,~J,
~ValueOfn,~kk,~hh,~ss,~tt);

end if;


count +:= 1;
if count eq 1000000 then print("done a millon tuples"); count:=0; end if;


else //k le #J+r. Done k-1 loops and k-1 < #J+r, so the tuple is not
     //built.  Only k-1 entries of the tuple have been picked.
for i:=Blower[k] to Bupper[k] do
bbb[k]:=i;
SieveBox(~bbb,~Solutions,~count,~Blower,~Bupper,~Q,~J,~JJJ,~ValueOfn,
~kk,~hh,~ss,~tt,~ImageOfzetaC,~ImageOfalphaC,k+1);
end for;

end if;

end procedure;
```

```
////////////////////////////////////////////////////////////////
/*
Start the loop over cases.
The loop variable is iiii.
*/
////////////////////////////////////////////////////////////////
//Write("Cases.txt",#ListOfCases);
//Write("Cases.txt",ListOfCases);

print("Number of Cases:");
#ListOfCases;
print("List of Cases:");
ListOfCases;


NumberOfCases:=#ListOfCases;
time for iiii:=1 to NumberOfCases do
//if iiii eq 3 then continue iiii; end if;
//time for iiii:=3 to 3 do



////////////////////////////////////////////////////////////////
/*
Give easy-to-use names to the parameters determining the case of the
factorized Thue-Mahler equation (11) that we are working with.  Do the
same for some numbers derived from these parameters.  These names
essentially coincide with the names given to them in Section 6.
*/
////////////////////////////////////////////////////////////////
zeta:=ListOfCases[iiii][v+2];
```

```
kk:=[];
ppp:=[];
ppi:=[];
hh:=[];
ss:=[];
tt:=[];
for L:=1 to v do
kk[L]:=ListOfCases[iiii][L][1][1]; //index of the unconstrained prime
                                   //above p[L]

if kk[L] ne m[L] + 1 then
ppp[L]:=pp[L][kk[L]]; //the unconstrained prime above p[L]
hh[L]:=h[L][kk[L]]; //smallest positive integer such that ppp[L]^hh[L] is
                    //principal
ppi[L]:=pi[L][kk[L]]; //generator of ppp[L]^hh[L]
else //kk[L] eq m[L] + 1
//there are no unramified degree one primes above p[L]
ppp[L]:=ideal<OK | 1>;
hh[L]:=1;
ppi[L]:=OK ! 1;
end if;
ss[L]:=ListOfCases[iiii][L][3][1]; // the parameter s_L
tt[L]:=0;
for j:=1 to m[L] do
tt[L]:=tt[L] + f[L][j]*ListOfCases[iiii][L][2][j]; //tt[L] =
                                                   //ord_p[L](N(bb))
end for;
end for; //end L loop

/*
The last thing to do is compute a generator alpha for the ideal
\mathfrak{a} \mathfrak{b} \mathfrak{p}_{1}^{s_1} \ldots
\mathfrak{p}_{v}^{s_v}
from Section 6, provided it is principle.
If the case of (11) we are considering has a solution, this ideal must be
```

```
principal.  So if it is not principle, we no there are no solutions to
case of (11) under consideration and skip directly to the next case (by
using the continue command)
*/

aa:=PowerProduct(pI,ListOfCases[iiii][v+1]);

aatimesbb:=aa;
for L:=1 to v do
aaa:=ListOfCases[iiii][L][2];
aatimesbb:=aatimesbb*PowerProduct(pp[L],aaa);
end for;

TheIdeal:=aatimesbb;
for L:=1 to v do
TheIdeal:=TheIdeal*(ppp[L]^ss[L]);
end for;

IsTheIdealPrinciple,alpha:=IsPrincipal(TheIdeal);
//print("alpha");
//alpha;

if IsTheIdealPrinciple eq false then continue iiii; end if;

delete aa;
delete aatimesbb;
delete TheIdeal;


////////////////////////////////////////////////////////////////////////
/*
ImageOfzetap[l][i][j] := the image of zeta under the embedding of K into
\overline{\QQ_{p_l}} defined by the map theta --> thetap[l][i][j].

ImageOfalphap[l][i][j] := the image of alpha under the embedding of K
```

```
into \overline{\QQ_{p_l}} defined by the map theta --> thetap[l][i][j].
*/
////////////////////////////////////////////////////////////////////////

ImageOfzetap:=[];
for l:=1 to v do
ImageOfzetap[l]:=[];
for i:=1 to m[l] do
ImageOfzetap[l][i]:=[**];
for j:=1 to e[l][i]*f[l][i] do
ImageOfzetap[l][i][j]:=0;
for ii:= 1 to n do
ImageOfzetap[l][i][j]:=ImageOfzetap[l][i][j]
+ zeta[ii]*ImageOfIntegralBasisElementp[l][i][j][ii];
end for; //ii
end for; //j
end for; //i
end for; //l

ImageOfalphap:=[];
for l:=1 to v do
ImageOfalphap[l]:=[];
for i:=1 to m[l] do
ImageOfalphap[l][i]:=[**];
for j:=1 to e[l][i]*f[l][i] do
ImageOfalphap[l][i][j]:=0;
for ii:= 1 to n do
ImageOfalphap[l][i][j]:=ImageOfalphap[l][i][j]
+ alpha[ii]*ImageOfIntegralBasisElementp[l][i][j][ii];
end for; //ii
end for; //j
end for; //i
end for; //l

////////////////////////////////////////////////////////////////////////
```

```
/*
ImageOfzetaC[i] := the image of zeta under the embedding of K into \CC
defined by the map theta --> thetaC[i].

ImageOfalphaC[i] := the image of alpha under the embedding of K into \CC
defined by the map theta --> thetaC[i].
*/
////////////////////////////////////////////////////////////////////////

ImageOfzetaC:=[ComplexField() | ];
for i:=1 to n do
ImageOfzetaC[i]:=0;
for ii:= 1 to n do
ImageOfzetaC[i]:=ImageOfzetaC[i]
+ zeta[ii]*ImageOfIntegralBasisElementC[i][ii];
end for; //ii
end for; //i

ImageOfalphaC:=[ComplexField() | ];
for i:=1 to n do
ImageOfalphaC[i]:=0;
for ii:= 1 to n do
ImageOfalphaC[i]:=ImageOfalphaC[i]
+ alpha[ii]*ImageOfIntegralBasisElementC[i][ii];
end for; //ii
end for; //i

////////////////////////////////////////////////////////////////////////
/*
For each p[l]:

If we are NOT in the case where there are no unramified degree one primes
above p[l], we fix the index i0 as in Section 7.  We will represent i0 by
the unique pair of integers (i,j) such that
theta^{(i_0)} = theta_i^{(j)}.
```

```
i0[l] will be a sequence of two integers such that
theta^{(i_0)} is represented by thetap[l][i0[l][1]][i0[l][2]]

If there are no unramified degree one primes above p[l], i0[l] will still
be a sequence of two integers, but it will have no real meaning.
*/
////////////////////////////////////////////////////////////////////////
i0:=[];
for l:=1 to v do
i0[l]:=[kk[l],1];
end for;


////////////////////////////////////////////////////////////////////////
/*
For each p[l]:

If we are NOT in the case where there are no unramified degree one primes
above p[l], we choose the indices j,k according to the guidelines in
Section 9.  As with i0 we represent j by a sequence of two integers
jjj[l] in such a way that theta^{(j)} is represented by
thetap[l][jjj[l][1]][jjj[l][2]].  Similarly, for k.

Suppose we find a choice of j,k that lets us use Lemma 7.4 to compute
n_l immediately.  If we compute a value for n_l that is not a nonnegative
integer, we move onto the next case using the continue command.  If we
compute a value for n_l that is nonnegative, then we absorb pi_l^n_l =
ppi[l]^n_l into alpha and stop considering the index l in the rest of the
algorithm, except in the last steps.

If there are no unramified degree one primes above p[l], we know that the
value of n_l is zero and we will stop considering the index l in the
algorithm, except in the last steps.

For l = 1,...,v:
```

If we know the value of n_l, then ValueOfn[l]:= that value
If we don't know the value of n_l, then ValueOfn[l]:=-1

JJ = set of those indices l in {1,...,v} for which there is at least one unramified degree one prime above p[l].

By the end of this block of code,
J = the set of all indices in {1,...,v} that we need to consider in the rest of the algorithm, except the last steps. These are the indices l for which there is at least one unramified degree one prime above p[l] and there is no choice of j,k that lets us use Lemma 7.4 to compute n_l.

It will be convenient to have defined the following set of indices
JJJ := {1} union { 1+l : l in J} union { 1+v+i : i in {1,...,r}}

I = set of those l in J for which there is no choice of j,k that lets us use Lemma 8.3 to bound n_l.

I2 = J - I.

For each l in I, jl[l] is the unique index such that JJJ[jl[l]]=l+1.

We will compute delta_1 and delta_2 for each l in J

We will compute the numbers alpha_i from Section 8 for each l in J:
alpha_i = LogarithmicAlphap[l][i]

We could compute the alpha_i and delta1, delta2 from Section 8 for each l in {1,...,v} for which there is at least one unramified degree one prime above p[l] or even for each l in {1,...,v}, but we don't

For l in I:
SpecialCase[l] = true if the choice of j,k puts us in the special case of Section 17.
SpecialCase[l] = false otherwise (general case)

```
Nstar[l] = N_{l}^{*} from Section 11.

Initialize ihat[l] from Section 17.  Note ihat[l] in JJJ, ihat[l] neq 2.

Intizalize istar[l].

Eventually, istar[l] will be the index in {2,...,1+v+r} such that
ihat[l] = JJJ[istar[l]].

Possibly define ihat[l] and istar[l] for some l.
*/
////////////////////////////////////////////////////////////////////////
ValueOfn:=[];
for l:=1 to v do
ValueOfn[l]:=-1; //initialize
end for;

Nstar:=[];
for l:=1 to v do
Nstar[l]:=padicprecision[l]+1; //initialize
/*the largest possible value of Valuation(x) for x an element of
FFFppF[l] is padicprecision[l]*/
end for;

u:=[];
for l:=1 to v do
u[l]:=-1; //initialize
end for;

SpecialCase:=[];
for l:=1 to v do
SpecialCase[l]:=false; //initialize
end for;
```

```
ihat:=[];
for l:=1 to v do
ihat[l]:=0; //initialize to a value that ihat[l] can never assume
end for;

istar:=[];
for l:=1 to v do
istar[l]:=0; //initialize to a value that ihat[l] can never assume
end for;



//start with J={1,...,v}
J:=[];
for l:=1 to v do
J[l]:=l;
end for;

/*
Remove from J all the indices l such that p[l] has no unramified degree
one primes above it.  After they are removed, we know that, for every l
in J, g has at least one linear factor in Q_{p_l}.  Also set the value of
n_l for these indices l.
*/
for l:=1 to v do
if kk[l] eq m[l]+1 then /*no degree one unramified primes above p_l*/
ValueOfn[l]:=0;
/* now absorb pi_l^n_l = ppi[l]^n_l into alpha and remove the index l
from J */
alpha:=alpha*ppi[l]^ValueOfn[l];
Exclude(~J,l);
end if;
end for;

/*JJ = set of those indices l in {1,...,v} for which there is at least
one unramified degree one prime above p[l].*/
```

```
JJ:=J;
I:=J;


//initialize delta1,delta2
delta1:=[**];
delta2:=[**];
for l:=1 to v do
delta1[l]:=FFFppF[l] ! 1;
delta2[l]:=FFFppF[l] ! 1;
end for;



//Now select j,k.  We will remove indices from J and I (if possible) as
//we go.

jjj:=[];
kkk:=[];
LogarithmicAlphap:=[];
CoefficientsLogarithmicAlphap:=[];
for l:=1 to v do
LogarithmicAlphap[l]:=[**];
CoefficientsLogarithmicAlphap[l]:=[**];
for i:=1 to 1+v+r do
LogarithmicAlphap[l][i]:=0;
CoefficientsLogarithmicAlphap[l][i]:=0;
end for;
end for;


for l in JJ do
/*try to find a choice of j,k that gives ord_{p_l}(delta_1) \neq 0*/
for i:=1 to m[l] do
if i eq i0[l][1] then continue i; end if;
```

```
for j:=1 to e[l][i]*f[l][i] do
jjj[l]:=[i,j];
for ii:=i to m[l] do
if ii eq i0[l][1] then continue ii; end if;
for jj:=1 to e[l][ii]*f[l][ii] do
if ii eq i and jj le j then continue jj; end if;
kkk[l]:=[ii,jj];
/*the last if statement ensures that we never try both j=a,k=b and
j=b,k=a*/

delta1[l]:= ((thetap[l][i0[l][1]][i0[l][2]] -
thetap[l][jjj[l][1]][jjj[l][2]])
/
(thetap[l][i0[l][1]][i0[l][2]] - thetap[l][kkk[l][1]][kkk[l][2]]))
*(ImageOfzetap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

if Valuation(delta1[l]) ne 0 then
/*
print("Valuation(delta1[l]):");
Valuation(delta1[l]);
print("iiii");
iiii;
print("l:");
l;
print("jjj[l] and kkk[l]:");
jjj[l];
kkk[l];
*/

delta2[l]:= ((thetap[l][jjj[l][1]][jjj[l][2]] -
```

```
thetap[l][kkk[l][1]][kkk[l][2]])
/
(thetap[l][kkk[l][1]][kkk[l][2]] - thetap[l][i0[l][1]][i0[l][2]]))
*(ImageOfzetap[l][i0[l][1]][i0[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][i0[l][1]][i0[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);
//print("Valuation(delta2[l])");
//Valuation(delta2[l]);

ValueOfn[l]:=
(Min([Valuation(delta1[l]),0]) - Valuation(delta2[l]))/hh[l];
//print("ValueOfn[l]");
//ValueOfn[l];

if not IsIntegral(ValueOfn[l]) or ValueOfn[l] lt 0 then
continue iiii; end if; // b/c no solutions

/* now absorb pi_l^n_l = ppi[l]^n_l into alpha and remove the index l
from J */

//print("alpha");
//alpha;
//print("ppi[l]");
//ppi[l];
alpha:=alpha*ppi[l]^ValueOfn[l];
Exclude(~J,l);
//print("alpha");
//alpha;


//Recompute ImageOfalphap[l]
ImageOfalphap[l]:=[];
```

```
for i1:=1 to m[l] do
ImageOfalphap[l][i1]:=[**];
for j1:=1 to e[l][i1]*f[l][i1] do
ImageOfalphap[l][i1][j1]:=0;
for ii1:= 1 to n do
ImageOfalphap[l][i1][j1]:=ImageOfalphap[l][i1][j1]
+ alpha[ii1]*ImageOfIntegralBasisElementp[l][i1][j1][ii1];
end for; //ii1
end for; //j1
end for; //i1

//Recompute ImageOfalphaC
ImageOfalphaC:=[ComplexField()|];
for i1:=1 to n do
ImageOfalphaC[i1]:=0;
for ii1:= 1 to n do
ImageOfalphaC[i1]:=ImageOfalphaC[i1]
+ alpha[ii1]*ImageOfIntegralBasisElementC[i1][ii1];
end for; //ii1
end for; //i1


//done choosing j,k for the current l, move onto the next l
continue l;

end if;
end for; //end jj loop
end for; //end ii loop
end for; //end j loop
end for; //end i loop
end for; //end l loop

//We are now finished defining J
/*J = the set of all indices in {1,...,v} that we need to consider in the
rest of the algorithm, except the last steps.  These are the indices l
```

```
for which there is at least one unramified degree one prime above p[l]
and there is no choice of j,k that lets us use Lemma 7.4 to compute n_l.*/


////////////
/*
It will be convenient to have defined the following set of indices
JJJ := {1} union { 1+l : l in J} union { 1+v+i : i in {1,...,r}}
*/
//////////
JJJ:=[1];
for l in J do
Append(~JJJ,1+l);
end for;
for i:=1 to r do
Append(~JJJ,1+v+i);
end for;

/*we start with I = J and remove indices from I as we go.
Want to make I = set of those l in J for which there is
no choice of j,k that lets us use Lemma 8.3 to bound n_l.*/
I:=J;

for l in J do
/*
if the program makes it this far, we know that there is no choice of j,k
such that ord_{p_l}(delta_1) \neq 0.
Now we try to find a choice of j,k such that Lemma 8.3 can be used to
find an upper bound on n_l
*/
for i:=1 to m[l] do
if i eq i0[l][1] then continue i; end if;
for j:=1 to e[l][i]*f[l][i] do
jjj[l]:=[i,j];
for ii:=i to m[l] do
```

```
if ii eq i0[l][1] then continue ii; end if;
for jj:=1 to e[l][ii]*f[l][ii] do
if ii eq i and jj le j then continue jj; end if;
kkk[l]:=[ii,jj];
/*the last if statement ensures that we never try both j=a,k=b and
j=b,k=a*/

//define delta1[l] and delta2[l]
delta1[l]:= ((thetap[l][i0[l][1]][i0[l][2]] -
thetap[l][jjj[l][1]][jjj[l][2]])
/
(thetap[l][i0[l][1]][i0[l][2]] - thetap[l][kkk[l][1]][kkk[l][2]]))
*(ImageOfzetap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])*
(ImageOfalphap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

delta2[l]:= ((thetap[l][jjj[l][1]][jjj[l][2]]
- thetap[l][kkk[l][1]][kkk[l][2]])
/
(thetap[l][kkk[l][1]][kkk[l][2]] - thetap[l][i0[l][1]][i0[l][2]]))
*(ImageOfzetap[l][i0[l][1]][i0[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][i0[l][1]][i0[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

//define alpha_i's for i in JJJ
LogarithmicAlphap[l][1]:=pAdicLog(delta1[l],p[l]);
for iii in J do
LogarithmicAlphap[l][1+iii]:=
pAdicLog(ImageOfpip[iii][kk[iii]][l][kkk[l][1]][kkk[l][2]]/
```

```
ImageOfpip[iii][kk[iii]][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;
for iii:=1 to r do
LogarithmicAlphap[l][1+v+iii]:=
pAdicLog(ImageOfepsp[iii][l][kkk[l][1]][kkk[l][2]]/
ImageOfepsp[iii][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;

//get the coefficients of the alpha_i's: the alpha_{ih}'s
for iii in JJJ do
CoefficientsLogarithmicAlphap[l][iii]:=
GetCoefficients(LogarithmicAlphap[l][iii],l);
end for;

/*Check if Lemma 8.3 can be applied and, if so,
compute Nstar[l] = N_{l}^{*}.*/

min:=Valuation(LogarithmicAlphap[l][JJJ[2]]);
for iii:=2 to #JJJ do
if Valuation(LogarithmicAlphap[l][JJJ[iii]]) lt min then
min:=Valuation(LogarithmicAlphap[l][JJJ[iii]]);
end if;
end for;
if Valuation(LogarithmicAlphap[l][1]) lt min then
temp:=Max([  Floor((1/hh[l])*(1/(p[l]-1) - Valuation(delta2[l]))),
Ceiling((1/hh[l])*(min - Valuation(delta2[l])))-1  ]);
if temp lt 0 then continue iiii; end if; //no solutions in the current
                                         //case
if temp lt Nstar[l] then Nstar[l]:=temp; end if;
end if;

for hhh:=1 to SS[l] do

min:=Valuation(CoefficientsLogarithmicAlphap[l][JJJ[2]][hhh]);
for iii:=2 to #JJJ do
```

```
if Valuation(CoefficientsLogarithmicAlphap[l][JJJ[iii]][hhh]) lt min then
min:=Valuation(CoefficientsLogarithmicAlphap[l][JJJ[iii]][hhh]);
end if;
end for; //end iii loop
if Valuation(CoefficientsLogarithmicAlphap[l][1][hhh]) lt min then
temp:=Max([  Floor((1/hh[l])*(1/(p[l]-1) - Valuation(delta2[l]))),
Ceiling((1/hh[l])*(u[l] + min - Valuation(delta2[l])))-1   ]);
if temp lt 0 then continue iiii; end if; //no solutions in the current
                                         //case
if temp lt Nstar[l] then Nstar[l]:=temp; end if;
end if;

end for; //end hhh loop

if Nstar[l] lt padicprecision[l]+1 then
//we've found a choice of j,k that lets us use Lemma 8.3 to bound n_l
Exclude(~I,l);
continue l;
end if;

end for;
end for;
end for;
end for;




/*
If the program makes it this far, we know that there is no choice of j,k
such that ord_{p_l}(delta_1) \neq 0 and also there is no choice of j,k
that lets us use Lemma 8.3 to get an upper bound on n_l.  So we will
eventually need to use linear forms in logs + lattice reduction to get a
small upper bound on n_l

Now we try to find a choice of j,k according to the remaining guidlines
```

```
in Section 9.  The whole point is to try to choose j,k so that
alpha_i /  alpha_{ihat} is in Q_{p_l} for all i in J.

For each candidate j,k, we first do some simple checks for conditions
that imply alpha_i /  alpha_{ihat} is in Q_{p_l} for all i in J.  If
we are forced to check directly whether alpha_i /  alpha_{ihat} is in
Q_{p_l} for all i, we do this for all candidates for ihat.  If we find a
candidate for ihat that works, we remember it.
*/


/* check if we can take j,k so that theta^{(j)},theta^{(k)} are roots of
linear factors of g(t) in Q_{p_l}[t] */
jfound:=false;
for i:=1 to m[l] do
if i eq i0[l][1] then continue i; end if;
if Degree(gp[l][i]) eq 1 then

if jfound eq false then
jjj[l]:=[i,1];
jfound:=true;
continue i;
end if;

if jfound eq true then
kkk[l]:=[i,1];
/* we've found a choice for j,k so that theta^{(j)},theta^{(k)} are roots
of linear factors of g(t) in Q_{p_l}[t] Section 17 */
SpecialCase[l]:=true;
continue l;
end if;

end if;
end for; //end i loop
```

```
/*
check if we can take j,k so that theta^{(j)}, theta^{(k)} are the roots
of a degree 2 factor of g(t) Q_{p_l}[t]
*/
for i:=1 to m[l] do
if Degree(gp[l][i]) eq 2 then
jjj[l]:=[i,1];
kkk[l]:=[i,2];
/*
we've found a j,k so that theta^{(j)}, theta^{(k)} are the roots of a
degree 2 factor of g(t) Q_{p_l}[t]
*/
SpecialCase[l]:=true;
continue l;
end if;
end for;



/*
If there is a nonlinear factor g^{\prime}(t) of g(t) in Q_{p_l}[t] such
that the extension of Q_{p_l} that g^{\prime}(t) generates contains all
the roots of g^{\prime}(t) (equivalently, the extension is Galois), then
taking j,k so that \theta^{(j)},\theta^{(k)} are two roots of
g^{\prime}(t) will ensure that alpha_i /  alpha_{ihat} is in Q_{p_l}
for all i.

If an irreducible polynomial over Q_{p_l}[t] is neither inertial nor
eisenstein, MAGMA does not support directly constructing the extension
of Q_{p_l} generated by the polynomial.  However, we know that the
extension of Q_{p_l} generated by g^{\prime}(t) is isomorphic to the
completion of K at the prime ideal above p_l which corresponds to
g^{\prime}(t).

Note also that irreducible polynomials over p-adic fields have no
```

```
repeated roots (since p-adic fields have characteristic zero).
*/

for i:=1 to m[l] do
if Degree(gp[l][i]) gt 1 and
#Roots(gp[l][i],Kpp[l][i]) eq Degree(gp[l][i]) then
jjj[l]:=[i,1];
kkk[l]:=[i,2];
/*
we've found j,k that ensure alpha_i / alpha_{ihat} is in Z_{p_l} for all
i and all candidates for ihat.
*/
SpecialCase[l]:=true;
continue l;
end if;
end for;




/*
Now we do the more difficult checks to see if we can choose j,k such
that alpha_i /  alpha_{ihat} is in Z_{p_l} for all i,j
*/


for i:=1 to m[l] do
if i eq i0[l][1] then continue i; end if;
for j:=1 to e[l][i]*f[l][i] do
jjj[l]:=[i,j];
for ii:=i to m[l] do
if ii eq i0[l][1] then continue ii; end if;
for jj:=1 to e[l][ii]*f[l][ii] do
if ii eq i and jj le j then continue jj; end if;
kkk[l]:=[ii,jj];
```

```
//define delta1[l] and delta2[l]
delta1[l]:= ((thetap[l][i0[l][1]][i0[l][2]]
- thetap[l][jjj[l][1]][jjj[l][2]])
/
(thetap[l][i0[l][1]][i0[l][2]] - thetap[l][kkk[l][1]][kkk[l][2]]))
*(ImageOfzetap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

delta2[l]:= ((thetap[l][jjj[l][1]][jjj[l][2]]
- thetap[l][kkk[l][1]][kkk[l][2]])
/
(thetap[l][kkk[l][1]][kkk[l][2]] - thetap[l][i0[l][1]][i0[l][2]]))
*(ImageOfzetap[l][i0[l][1]][i0[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][i0[l][1]][i0[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

//define alpha_i's for i in JJJ
LogarithmicAlphap[l][1]:=pAdicLog(delta1[l],p[l]);
for iii in J do
LogarithmicAlphap[l][1+i]:=
pAdicLog(ImageOfpip[iii][kk[iii]][l][kkk[l][1]][kkk[l][2]]
/ImageOfpip[iii][kk[iii]][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;
for iii:=1 to r do
LogarithmicAlphap[l][1+v+i]:=
pAdicLog(ImageOfepsp[iii][l][kkk[l][1]][kkk[l][2]]
```

```
/ImageOfepsp[iii][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;

//get the coefficients of the alpha_i's: the alpha_{ih}'s
for iii in JJJ do
CoefficientsLogarithmicAlphap[l][iii]:=
GetCoefficients(LogarithmicAlphap[l][iii],l);
end for;

/*
Check if there is a choice of j,k such that the alpha_i (i in JJJ) all
belong to $\QQ_{p_l}$.
*/
flag:=true;
for iii in JJJ do
for hhh:=2 to SS[l] do
if not IsZero(CoefficientsLogarithmicAlphap[l][iii][hhh]) then
flag:=false; break iii; end if;
end for;
end for;
if flag eq true then
//we've found a choice of j,k with desired properties
SpecialCase[l]:=true;
continue l;
end if;


/*
Now we check directly whether we have alpha_i /  alpha_{ihat} in Q_{p_l}
for all i.  We do this for all candidates for ihat.
*/

//find all candidates for istar (hence for ihat)
candidatesforistar:=[];
min:=Valuation(LogarithmicAlphap[l][JJJ[2]]);
```

```
for iii:=JJJ[2] to JJJ[#JJJ] do
if Valuation(LogarithmicAlphap[l][iii]) lt min then
min:=Valuation(LogarithmicAlphap[l][iii]);
end if;
end for;
for iiiiii:=2 to #JJJ do
iii:=JJJ[iiiiii];
if min eq Valuation(LogarithmicAlphap[l][iii]) then
Append(~candidatesforistar,iiiiii); end if;
end for;

/* For all candidates for ihat, check whether alpha_i /  alpha_{ihat} in
Q_{p_l} for all i in JJJ */
temp:=[];
for istarstar in candidatesforistar do
ihathat:=JJJ[istarstar];
for iii in JJJ do
temp[iii]:=GetCoefficients(LogarithmicAlphap[l][iii]
/LogarithmicAlphap[l][ihathat],l);
end for;
flag:=true;
for iii in JJJ do
for hhh:=2 to SS[l] do
if not IsZero(temp[iii][hhh]) then flag:=false; break iii; end if;
end for;
end for;
if flag eq true then
/* we've found a choice of j,k with the desired properties */
SpecialCase[l]:=true;
ihat[l]:=ihathat;
istar[l]:=istarstar;
continue l;
end if;
end for;
```

```
end for; //jj
end for; //ii
end for; //j
end for; //i


/*
If the program makes it this far, it means there is no choice of j,k for
which ord_{p_l}(delta_1) = 0, no choice for which Lemma 8.3 gives an
upper bound for n_l, and no choice which puts us in the special case of
Section 17 (alpha_i /  alpha_{ihat} in Z_{p_l} for all i).

We know that there are at most two degree one factors of g(t) in
Q_{p_l}[t] and there are no degree two factors.  So there is at least one
factor of degree >= 3.  We choose a nonlinear factor of g(t) in
Q_{p_l}[t] of minimal degree and choose j,k so that \theta^{(j)},
\theta^{(k)} are roots of that factor.
*/

minimaldegreegreaterthan1:=Degree(FFFppF[l],PrimeField(FFFppF[l]));
//initialize, no gp[l][i] can have degree strictly larger than this

for i:=1 to m[l] do
if Degree(gp[l][i]) gt 1 and
Degree(gp[l][i]) lt minimaldegreegreaterthan1 then
minimaldegreegreaterthan1:=Degree(gp[l][i]);
jjj:=[i,1];
kkk:=[i,2];
end if;
end for;

end for; //end l loop

I2:=J; //J = I union I2, I intersect I2 = empty
```

```
for l in I do
Exclude(~I2,l);
end for;

//For each l in I, jl[l] is the unique index such that JJJ[jl[l]]=l+1.
jl:=[];
for i:=1 to v do
jl[i]:=0;
end for;
for l in I do
for i:=2 to 1+#J do
if JJJ[i] eq l+1 then jl[l]:=i; break i; end if;
end for;
end for;


////////////////////////////////////////////////////////////////////////
/*
For each l in I, we will find a value for the index ihat
and compute the beta_i's from Section 17: ihat[l], beta[l][i]

We also define d_l from Section 17: dd[l]

It can be that a choice of ihat[l] was already made in the last step.
If ihat[l] >0, we know this choice has been made.  Otherwise, ihat[l] = 0
and we know a choice hasn't been made yet.
*/
////////////////////////////////////////////////////////////////////////

beta:=[];
dd:=[];
for l:=1 to v do
beta[l]:=[**];
end for;
```

```
for l in I do

//define delta1[l] and delta2[l]

delta1[l]:= ((thetap[l][i0[l][1]][i0[l][2]]
- thetap[l][jjj[l][1]][jjj[l][2]])
/
(thetap[l][i0[l][1]][i0[l][2]] - thetap[l][kkk[l][1]][kkk[l][2]]))
*(ImageOfzetap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*
(ImageOfalphap[l][kkk[l][1]][kkk[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

delta2[l]:= ((thetap[l][jjj[l][1]][jjj[l][2]]
- thetap[l][kkk[l][1]][kkk[l][2]])
/
(thetap[l][kkk[l][1]][kkk[l][2]] - thetap[l][i0[l][1]][i0[l][2]]))
*(ImageOfzetap[l][i0[l][1]][i0[l][2]]
/
ImageOfzetap[l][jjj[l][1]][jjj[l][2]])
*(ImageOfalphap[l][i0[l][1]][i0[l][2]]
/
ImageOfalphap[l][jjj[l][1]][jjj[l][2]]);

//define alpha_i's
LogarithmicAlphap[l][1]:=pAdicLog(delta1[l],p[l]);
for iii in J do
LogarithmicAlphap[l][1+iii]:=
pAdicLog(ImageOfpip[iii][kk[iii]][l][kkk[l][1]][kkk[l][2]]
/ImageOfpip[iii][kk[iii]][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;
for iii:=1 to r do
```

```
LogarithmicAlphap[l][1+v+iii]:=
pAdicLog(ImageOfepsp[iii][l][kkk[l][1]][kkk[l][2]]
/ImageOfepsp[iii][l][jjj[l][1]][jjj[l][2]],p[l]);
end for;


if SpecialCase[l] eq true then

if ihat[l] eq 0 then
min:=Valuation(LogarithmicAlphap[l][JJJ[2]]);
ihat[l]:=JJJ[2];
istar[l]:=2;
for i:=3 to #JJJ do
iii:=JJJ[i];
if Valuation(LogarithmicAlphap[l][iii]) lt min then
min:=Valuation(LogarithmicAlphap[l][iii]);
ihat[l]:=iii;
istar[l]:=i;
end if;
end for;
end if;

for iii:=1 to 1+v+r do
beta[l][iii] := -LogarithmicAlphap[l][iii]/LogarithmicAlphap[l][ihat[l]];
/*
For l in I, we know that the beat_i's live in Z_{p_l}.  So their
coefficients on the power basis for FFFppF[l] are all zero, except for
the first.  We can use this to save on computations with the beta_i's:
*/
beta[l][iii] := GetCoefficients(beta[l][iii],l)[1];
end for;

dd[l]:=Valuation(delta2[l]) - Valuation(LogarithmicAlphap[l][ihat[l]]);

else //SpecialCase[l] eq false //general case
```

```
//get the coefficients of the alpha_i's: the alpha_{ih}'s
for iii:=1 to 1+v+r do
CoefficientsLogarithmicAlphap[l][iii]:=
GetCoefficients(LogarithmicAlphap[l][iii],l);
end for;

// Choose an index hhh in {1,...,S[l]} arbitrarily
hhh:=1;

if ihat[l] eq 0 then
min:=Valuation(CoefficientsLogarithmicAlphap[l][JJJ[2]][hhh]);
ihat[l]:=JJJ[2];
istar[l]:=2;
for i:=3 to #JJJ do
iii:=JJJ[i];
if Valuation(CoefficientsLogarithmicAlphap[l][iii][hhh]) lt min then
min:=Valuation(CoefficientsLogarithmicAlphap[l][iii][hhh]);
ihat[l]:=iii;
istar[l]:=i;
end if;
end for;
end if;

for iii:=1 to 1+v+r do
beta[l][iii] := -CoefficientsLogarithmicAlphap[l][iii][hhh]
/ CoefficientsLogarithmicAlphap[l][ihat[l]][hhh];
end for;

dd[l]:=Valuation(delta2[l]) - Valuation(LogarithmicAlphap[l][ihat[l]])
- u[l];

end if;

end for; //end l
```

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////
/////////////////
/*
Compute c_1(l) = c1[l] for each l in I
*/
///////////////
//////////////////////////////////
////////////////////////////////////////////////////////////////////////

//////////////////////
/*

For each l in I:

Find preimages in FFF of theta^{(i_0)}, theta^{(j)}, theta^{(k)} under
the embedding sigma = mFFFppF[L] of FFF into FFFppF[l].  We choose not to
use MAGMA's bulit in preimage-finder to avoid round-off errors.

To find the preimage of theta^{(i_0)}, we actually find the index ii0 in
{1,...,n} such that mFFFppF[L](thetaF[ii0]) = thetap[l][i0[1]][i0[2]].
Similarly for theta^{(j)}, theta^{(k)}.

PreimageOfalphaj = preimage in FFF of \alpha^{(j)} under the embedding
sigma = mFFFppF[L] of FFF into FFFppF[l]

PreimageOfzetaj = preimage in FFF of \zeta^{(j)} under the embedding
sigma = mFFFppF[L] of FFF into FFFppF[l]

PreimageOfalphak = preimage in FFF of \alpha^{(k)} under the embedding
sigma = mFFFppF[L] of FFF into FFFppF[l]
```

```
PreimageOfzetak = preimage in FFF of \zeta^{(k)} under the embedding
sigma = mFFFppF[L] of FFF into FFFppF[l]

We also find the preimages of
pi_{i}^{(j)}, pi_{i}^{(k)}, eps_{i}^{(j)}, eps_{i}^{(k)}
*/
////////////////////

//intialization
c1:=[**];
rootsofginFFFppF:=[**];
PreimageOfalphaj:=[];
PreimageOfzetaj:=[];
PreimageOfalphak:=[];
PreimageOfzetak:=[];
for l:=1 to v do
c1[l]:=0;
rootsofginFFFppF[l]:=mFFFppF[l](rootsofginFFF);
end for;

for l in I do

//find preimage of \theta^{(i_0)}
max:=Valuation(thetap[l][i0[l][1]][i0[l][2]] - rootsofginFFFppF[l][1]);
ii0:=1;
for i:=2 to n do
if Valuation(thetap[l][i0[l][1]][i0[l][2]] - rootsofginFFFppF[l][i]) gt
max then
max:=Valuation(thetap[l][i0[l][1]][i0[l][2]] - rootsofginFFFppF[l][i]);
ii0:=i;
end if;
end for;

//find preimage of \theta^{(j)}
```

```
max:=Valuation(thetap[l][jjj[l][1]][jjj[l][2]] - rootsofginFFFppF[l][1]);
ijjj:=1;
for i:=2 to n do
if Valuation(thetap[l][jjj[l][1]][jjj[l][2]] - rootsofginFFFppF[l][i]) gt
max then
max:=Valuation(thetap[l][jjj[l][1]][jjj[l][2]] - rootsofginFFFppF[l][i]);
ijjj:=i;
end if;
end for;

//find preimage of \theta^{(k)}
max:=Valuation(thetap[l][kkk[l][1]][kkk[l][2]] - rootsofginFFFppF[l][1]);
ikkk:=1;
for i:=2 to n do
if Valuation(thetap[l][kkk[l][1]][kkk[l][2]] - rootsofginFFFppF[l][i]) gt
max then
max:=Valuation(thetap[l][kkk[l][1]][kkk[l][2]] - rootsofginFFFppF[l][i]);
ikkk:=i;
end if;
end for;

PreimageOfzetaj:=0;
for i:= 1 to n do
PreimageOfzetaj:=PreimageOfzetaj
+ zeta[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfalphaj:=0;
for i:= 1 to n do
PreimageOfalphaj:=PreimageOfalphaj
+ alpha[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfzetak:=0;
for i:= 1 to n do
PreimageOfzetak:=PreimageOfzetak
+ zeta[i]*ImageOfIntegralBasisElementF[ikkk][i];
```

```
end for; //i
PreimageOfalphak:=0;
for i:= 1 to n do
PreimageOfalphak:=PreimageOfalphak
+ alpha[i]*ImageOfIntegralBasisElementF[ikkk][i];
end for; //i

/*
Note that:
preimage of pi_{i}^{(j)} = ImageOfpiF[i][kk[i]][ijjj]
preimage of pi_{i}^{(k)} = ImageOfpiF[i][kk[i]][ikkk]
preimage of eps_{i}^{(j)} = ImageOfepsF[i][ijjj]
preimage of eps_{i}^{(k)} = ImageOfepsF[i][ikkk]
*/

////////////////////////////////////
/*
Define the alpha_i for i in J, construct L, and find the prime ideal of L
corresponding to sigma (see Section 11).
*/
////////////////////////////////////
alphaALGEBRAIC:=[FFF|];
for i:=1 to 1 + v + r do
alphaALGEBRAIC[i]:=1; //initialize
end for;
alphaALGEBRAIC[1]:=
(thetaF[ii0] - thetaF[ijjj] / thetaF[ii0] - thetaF[ikkk])
*(PreimageOfalphak/PreimageOfalphaj)*(PreimageOfzetak/PreimageOfzetaj);
for i in J do
alphaALGEBRAIC[1+i]:=
ImageOfpiF[i][kk[i]][ikkk]/ImageOfpiF[i][kk[i]][ijjj];
end for;
for i:=1 to r do
alphaALGEBRAIC[1+v+i]:=
ImageOfepsF[i][ikkk]/ImageOfepsF[i][ijjj];
```

```
end for;

LL:=sub<FFF | alphaALGEBRAIC>;
LLx<x>:=PolynomialRing(LL);
D:=Degree(LL);
mm:=1+#J+r;

ppL:=ppF[l] meet MaximalOrder(LL);
eppL:=RamificationIndex(ppL);
fppL:=InertiaDegree(ppL);

////////////////////////////////////////////////////////////////////////
/*
Compute all the constants that go into Yu's Theorem
*/
///
////////////////////////////////////////////////////////////////////////
d:=D;
ff:=fppL;

if p[l] eq 2 and IsIrreducible(x^2 + x + 1) then
d:=2*D;
ff:=Max([2,fppL]);
end if;

if p[l] ge 3 and ((p[l]^fppL mod 4) eq 1) and
IsIrreducible(x^3 + x^2 + x + 1) then
d:=2*D;
ff:=Max([2,fppL]);
end if;

delete x;
delete LLx;

kappa:=Floor(2*eppL*Log(p[l])/(p[l]-1));
```

```
//note d \geq 2 for us

// default case: p=2
kappa1:=160;
kappa2:=32;
kappa3:=40;
kappa4:=276;
kappa5:=16;
kappa6:=8;

if p[l] eq 3 then
kappa1:=759;
kappa2:=16;
kappa3:=20;
kappa4:=1074;
kappa5:=8;
kappa6:=4;
end if;

if p[l] ge 5 then
kappa5:=8;
kappa6:=4;

if eppL eq 1 then
kappa2:=8*(p[l]-1)/(p[l]-2);
kappa3:=10;
if p[l] mod 4 eq 1 then
kappa1:=1473;
kappa4:=394*(p[l]-1)/(p[l]-2);
end if;
if p[l] mod 4 eq 3 then
kappa1:=1282;
kappa4:=366*(p[l]-1)/(p[l]-2);
end if;
```

```
end if;

if eppL ge 2 then
kappa2:=16;
kappa3:=20;
if p[l] eq 5 then
kappa1:=319;
kappa4:=402;
end if;
if p[l] ge 7 and p[l] mod 4 eq 1 then
kappa1:=1502;
kappa4:=1372;
end if;
if p[l] ge 7 and p[l] mod 4 eq 3 then
kappa1:=2190;
kappa4:=1890;
end if;
end if;
end if;

c2:=((((mm+1)^(mm+2)) * d^(mm+2))/(Factorial(mm-1)))
* ( p[l]^ff / (ff*Log(p[l]))^3 ) * Max([1,Log(d)])
* Max( [4+Log(mm+1)+Log(d),eppL,ff*Log(p[l])] );


halphaALGEBRAIC:=[];
for i:=1 to 1+v+r do
halphaALGEBRAIC[i]:=AbsoluteLogarithmicHeight(alphaALGEBRAIC[i]);
end for;

c3prime:=kappa1 * kappa2^mm * (mm / ff*Log(p[l]) )^(mm-1);
for i in JJJ do
c3prime:=c3prime*Max([ halphaALGEBRAIC[i],
ff*Log(p[l])/(kappa3*(mm+4)*d) ]);
end for;
```

```
c3primeprime:= kappa4 * kappa5^mm * Exp(mm) * p[l]^(kappa*(mm-1));
for i in JJJ do
c3primeprime:=c3primeprime*Max([ halphaALGEBRAIC[i],
ff*Log(p[l])/(Exp(2)*kappa6*(p[l]^kappa)*d)  ]);
end for;

qu:=4; //default case: p \geq 3, p^f_pp \equiv 1 mod 4
if p[l] eq 2 then qu:=3; end if;
if p[l] ge 3 and p[l]^fppL mod 4 eq 3 then qu:=1; end if;

c1[l]:=(1/eppL)*(c2/qu)*Max([c3prime,c3primeprime]);

end for; //end l loop

//////////////////////////////////////////////////////////////////////
/*
Done computing c1[l] for all l in I
*/
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
/*
Compute c4, c5 (with primed versions), c7
*/
//////////////////////////////////////////////////////////////////////


c4prime:=0;
for l in I do
if c4prime lt c1[l]/hh[l] then c4prime:=c1[l]/hh[l]; end if;
end for;

c4primeprime:=0;
c5primeprime:=0;
```

```
for l in I2 do
if c4primeprime lt Nstar[l] then c4primeprime:=Nstar[l]; end if;
end for;

c4:=c4prime;
if c4prime lt c4primeprime then c4:=c4primeprime; end if;

c5:=0;
for l in I do
if c5 lt -Valuation(delta2[l])/hh[l] then
c5:=-Valuation(delta2[l])/hh[l];
end if;
end for;



c6:=0;
if not IsEmpty(I) then

l:=I[1];
max:=c1[l]/hh[l]; if Exp(2) gt max then max:=Exp(2); end if;
c6:=2*(-Valuation(delta2[l])/hh[l] + max*Log(max));

for l in I do
max:=c1[l]/hh[l]; if Exp(2) gt max then max:=Exp(2); end if;
if 2*( -Valuation(delta2[l])/hh[l] + max*Log(max)) gt c6 then
c6:=2*( -Valuation(delta2[l])/hh[l] + max*Log(max));
end if;
end for;
c6:=Ceiling(c6)-1;

end if;

c7:=2;
if c4primeprime gt c7 then c7:=c4primeprime; end if;
```

```
c7:=Max([2,c4primeprime,c6]);

////////////////////////////////////////////////////////////////////////
/*
Compute c_{8}^{\prime}, c_{8}^{\prime \prime}, c_{9}^{\prime},
c_{9}^{\prime \prime}
*/
////////////////////////////////////////////////////////////////////////
min:=Abs(ImageOfalphaC[1]*ImageOfzetaC[1]);
for i:=2 to n do
if Abs(ImageOfalphaC[i]*ImageOfzetaC[i]) lt min then
min:=Abs(ImageOfalphaC[i]*ImageOfzetaC[i]); end if;
end for;
c8prime:=Abs(a);
for i:=1 to v do
if i in J then
c8prime:= c8prime*p[i]^(ss[i]+tt[i]);
else
c8prime:= c8prime*p[i]^(ss[i]+tt[i]+ValueOfn[i]*hh[i]);
end if;
end for;
c8prime:=Log(c8prime/min);

prod:=1;
for i in J do
l:=i;
prod:=prod*ImageOfpiC[i][kk[i]][1];
end for;
min:=Abs(prod);
for ii:=2 to n do
prod:=1;
for i in J do
l:=i;
prod:=prod*ImageOfpiC[i][kk[i]][ii];
end for;
```

```
if Abs(prod) lt min then min:=Abs(prod); end if;
end for;
c9prime:=1;
for i in J do
c9prime:=c9prime*p[i]^hh[i];
end for;
c9prime:=Log(c9prime/min);

max:=Abs(ImageOfalphaC[1]*ImageOfzetaC[1]);
for i:=2 to n do
if Abs(ImageOfalphaC[i]*ImageOfzetaC[i]) gt max then
max:=Abs(ImageOfalphaC[i]*ImageOfzetaC[i]); end if;
end for;
c8primeprime:=Log(max);

prod:=1;
for i in J do
prod:=prod*ImageOfpiC[i][kk[i]][1];
end for;
max:=Abs(prod);
for ii:=2 to n do
prod:=1;
for i in J do
prod:=prod*ImageOfpiC[i][kk[i]][ii];
end for;
if Abs(prod) gt max then max:=Abs(prod); end if;
end for;
c9primeprime:=Log(max);

////////////////////////////////////////////////////////////////////////
/*
If s > 0:
For each i0 in {1,...,s}, compute the indices j=j(i0), k=k(i0), and the
numbers c16=c16(i0), c17=c17(i0), c18=c18(i0).
*/
```

```
////////////////////////////////////////////////////////////////////////
jjjjjj:=[];
kkkkkk:=[];
c16:=[];
c17:=[];
c18:=[];
////////////////////////////////////////////////////////////////////////
/*
s >= 3 case
*/
////////////////////////////////////////////////////////////////////////
if s ge 3 then

sFFF:=Signature(FFF); //sFFF = number of real embeddings of \CC
LogarithmicAlphaC:=[];

for i0i0i0:=1 to s do

////////////////////////
/*
Choose j,k and compute c16
*/
////////////////////////
max:=0;
for j:=1 to s do
if j eq i0i0i0 then continue j; end if;
for k:=j+1 to s do
if k eq i0i0i0 then continue k; end if;
OneOverc16:=(Abs(thetaC[i0i0i0] - thetaC[j])/2)
*(Abs(thetaC[k] - thetaC[i0i0i0])/Abs(thetaC[j] - thetaC[k]));
if OneOverc16 gt max then
max:=OneOverc16;
jjjjjj[i0i0i0]:=j;
kkkkkk[i0i0i0]:=k;
end if;
```

```
end for;
end for;
c16[i0i0i0]:=1/max;

//////////////////////
/*
Select an embedding (sigma in Section 13) of FFF into the complex numbers
by selecting a conjugate in \CC (phidot) of the generator of FFF (FFF.1).

Find the images of the roots of g in \CC under this embedding:
rootsofginC

Embedding FFF into \RR will produce a smaller constant c17[i0i0i0].

Define kappa.
*/
//////////////////////
kappa:=1;
if sFFF eq 0 then kappa:=2; end if;
phidot:=Conjugates(FFF.1)[1];
rootsofginC:=[];
for i:=1 to n do
rootsofginC[i]:=0*phidot;
for ii:=1 to Degree(FFF) do
rootsofginC[i]:=rootsofginC[i]+rootsofginFFF[i][ii]*phidot^(ii-1);
end for;
end for;

//////////////////////
/*
Find preimages in FFF of theta^{(i_0)}, theta^{(j)}, theta^{(k)} under
the embedding sigma of FFF into \CC.  We choose not to use MAGMA's bulit
in preimage-finder to avoid round-off errors.

To find the preimage of theta^{(i_0)}, we actually find the index ii0 in
```

```
{1,...,n} such that sigma(thetaF[ii0]) = thetaC[i0i0i0].
Similarly for theta^{(j)}, theta^{(k)}.

PreimageOfalphaj = preimage in FFF of \alpha^{(j)} under the embedding
sigma of FFF into \CC

PreimageOfzetaj = preimage in FFF of \zeta^{(j)} under the embedding
sigma of FFF into \CC

PreimageOfalphak = preimage in FFF of \alpha^{(k)} under the embedding
sigma of FFF into \CC

PreimageOfzetak = preimage in FFF of \zeta^{(k)} under the embedding
sigma of FFF into \CC

We also find the preimages of
pi_{i}^{(j)}, pi_{i}^{(k)}, eps_{i}^{(j)}, eps_{i}^{(k)}
*/
//////////////////////////
//find preimage of \theta^{(i_0)}
min:=Abs(thetaC[i0i0i0] - rootsofginC[1]);
ii0:=1;
for i:=2 to n do
if Abs(thetaC[i0i0i0] - rootsofginC[i]) lt min then
min:=Abs(thetaC[i0i0i0] - rootsofginC[i]);
ii0:=i;
end if;
end for;

//find preimage of \theta^{(j)}
min:=Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[1]);
ijjj:=1;
for i:=2 to n do
if Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[i]) lt min then
min:=Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[i]);
```

```
ijjj:=i;
end if;
end for;

//find preimage of \theta^{(k)}
min:=Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[1]);
ikkk:=1;
for i:=2 to n do
if Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[i]) lt min then
min:=Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[i]);
ikkk:=i;
end if;
end for;

PreimageOfzetaj:=0;
for i:= 1 to n do
PreimageOfzetaj:=PreimageOfzetaj
+ zeta[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfalphaj:=0;
for i:= 1 to n do
PreimageOfalphaj:=PreimageOfalphaj
+ alpha[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfzetak:=0;
for i:= 1 to n do
PreimageOfzetak:=PreimageOfzetak
+ zeta[i]*ImageOfIntegralBasisElementF[ikkk][i];
end for; //i
PreimageOfalphak:=0;
for i:= 1 to n do
PreimageOfalphak:=PreimageOfalphak
+ alpha[i]*ImageOfIntegralBasisElementF[ikkk][i];
end for; //i
```

```
/*
Note that:
preimage of pi_{i}^{(j)} = ImageOfpiF[i][kk[i]][ijjj]
preimage of pi_{i}^{(k)} = ImageOfpiF[i][kk[i]][ikkk]
preimage of eps_{i}^{(j)} = ImageOfepsF[i][ijjj]
preimage of eps_{i}^{(k)} = ImageOfepsF[i][ikkk]
*/

//////////////////////////////////////
/*
Define the alpha_i for i in J, construct L, and find the prime ideal of L
corresponding to sigma (see Section 11).
*/
//////////////////////////////////////
alphaALGEBRAIC:=[FFF|];
for i:=1 to 1 + v + r do
alphaALGEBRAIC[i]:=1; //initialize
end for;
alphaALGEBRAIC[1]:=(thetaF[ii0] - thetaF[ijjj]
/ thetaF[ii0] - thetaF[ikkk])
*(PreimageOfalphak/PreimageOfalphaj)
*(PreimageOfzetak/PreimageOfzetaj);
for i in J do
alphaALGEBRAIC[1+i]:=ImageOfpiF[i][kk[i]][ikkk]
/ImageOfpiF[i][kk[i]][ijjj];
end for;
for i:=1 to r do
alphaALGEBRAIC[1+v+i]:=ImageOfepsF[i][ikkk]/ImageOfepsF[i][ijjj];
end for;

LL:=sub<FFF | alphaALGEBRAIC>;
d:=Degree(LL);
mm:=1+#J+r;

//////////////////////////////////////
```

```
/*
Compute c17.

Also compute the alpha_i as defined in Sections 18 and 20:
LogarithmicAlphaC[i0i0i0][i]
*/
////////////////////////////////////
LogarithmicAlphaC[i0i0i0]:=[RealField() | ];
for i:=1 to 1+v + r do
LogarithmicAlphaC[i0i0i0][i]:=0;
end for;



temp:=((thetaC[i0i0i0] - thetaC[jjjjjj[i0i0i0]])
/(thetaC[i0i0i0] - thetaC[kkkkkk[i0i0i0]]))
*(ImageOfzetaC[kkkkkk[i0i0i0]]/ImageOfzetaC[jjjjjj[i0i0i0]])
*(ImageOfalphaC[kkkkkk[i0i0i0]]/ImageOfalphaC[jjjjjj[i0i0i0]]);
LogarithmicAlphaC[i0i0i0][1]:=Log(Abs(temp));
H:=Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1]),
Abs(Log(Abs(temp))), 0.16] );

for i in J do
temp:=ImageOfpiC[i][kk[i]][kkkkkk[i0i0i0]]
/ImageOfpiC[i][kk[i]][jjjjjj[i0i0i0]];
LogarithmicAlphaC[i0i0i0][1+i]:=Log(Abs(temp));
H:=H*Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1+i]),
Abs(Log(Abs(temp))), 0.16] );
end for;

for i:=1 to r do
temp:=ImageOfepsC[i][kkkkkk[i0i0i0]]/ImageOfepsC[i][jjjjjj[i0i0i0]];
LogarithmicAlphaC[i0i0i0][1+v+i]:=Log(Abs(temp));
H:=H*Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1+v+i]),
Abs(Log(Abs(temp))), 0.16] );
end for;
```

```
c17[i0i0i0]:=
Min([(1/kappa)*((Exp(1)*mm/2)^(kappa))*(30^(mm+3))*(mm^(3.5)),
(2^(6*mm+20))])*(d^2)*(1+Log(d))*H;

//////////////////////////////
/*
Compute c18
*/
//////////////////////////////

c18[i0i0i0]:= Log(2*Log(2)*c16[i0i0i0]) + c17[i0i0i0];
end for; //end i0i0i0 loop
end if;

////////////////////////////////////////////////////////////////////////
/*
s=1,2 case
*/
////////////////////////////////////////////////////////////////////////

if s eq 1 or s eq 2 then

sFFF:=Signature(FFF); //sFFF = number of real embeddings of \CC
LogarithmicAlphaC:=[];

for i0i0i0:=1 to s do

//////////////////////
/*
Choose j,k and compute c16
*/
//////////////////////
max:=0;
for i:=1 to t do
```

```
j:=s-1 + 2*i;
k:=j+1;
OneOverc16:=(Abs(thetaC[i0i0i0] - thetaC[j])/2)*
(Abs(thetaC[k] - thetaC[i0i0i0])/Abs(thetaC[j] - thetaC[k]));
if OneOverc16 gt max then
max:=OneOverc16;
jjjjjj[i0i0i0]:=j;
kkkkkk[i0i0i0]:=k;
end if;
end for;
c16[i0i0i0]:=1/max;

//////////////////////
/*
Select an embedding (sigma in Section 13) of FFF into the complex numbers
by selecting a conjugate in \CC (phidot) of the generator of FFF (FFF.1).

Find the images of the roots of g in \CC under this embedding:
rootsofginC

Embedding FFF into \CC will produce a smaller constant c17[i0i0i0].

Define kappa.
*/
//////////////////////
kappa:=1;
if sFFF eq 0 then kappa:=2; end if;
phidot:=Conjugates(FFF.1)[1];
rootsofginC:=[];
for i:=1 to n do
rootsofginC[i]:=0*phidot;
for ii:=1 to Degree(FFF) do
rootsofginC[i]:=rootsofginC[i]+rootsofginFFF[i][ii]*phidot^(ii-1);
end for;
end for;
```

```
//////////////////////
/*
Find preimages in FFF of theta^{(i_0)}, theta^{(j)}, theta^{(k)} under
the embedding sigma of FFF into \CC.  We choose not to use MAGMA's bulit
in preimage-finder to avoid round-off errors.

To find the preimage of theta^{(i_0)}, we actually find the index ii0 in
{1,...,n} such that sigma(thetaF[ii0]) = thetaC[i0i0i0].
Similarly for theta^{(j)}, theta^{(k)}.

PreimageOfalphaj = preimage in FFF of \alpha^{(j)} under the embedding
sigma of FFF into \CC

PreimageOfzetaj = preimage in FFF of \zeta^{(j)} under the embedding
sigma of FFF into \CC

PreimageOfalphak = preimage in FFF of \alpha^{(k)} under the embedding
sigma of FFF into \CC

PreimageOfzetak = preimage in FFF of \zeta^{(k)} under the embedding
sigma of FFF into \CC

We also find the preimages of
pi_{i}^{(j)}, pi_{i}^{(k)}, eps_{i}^{(j)}, eps_{i}^{(k)}
*/
//////////////////////////
//find preimage of \theta^{(i_0)}
min:=Abs(thetaC[i0i0i0] - rootsofginC[1]);
ii0:=1;
for i:=2 to n do
if Abs(thetaC[i0i0i0] - rootsofginC[i]) lt min then
min:=Abs(thetaC[i0i0i0] - rootsofginC[i]);
ii0:=i;
end if;
```

```
end for;

//find preimage of \theta^{(j)}
min:=Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[1]);
ijjj:=1;
for i:=2 to n do
if Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[i]) lt min then
min:=Abs(thetaC[jjjjjj[i0i0i0]] - rootsofginC[i]);
ijjj:=i;
end if;
end for;

//find preimage of \theta^{(k)}
min:=Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[1]);
ikkk:=1;
for i:=2 to n do
if Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[i]) lt min then
min:=Abs(thetaC[kkkkkk[i0i0i0]] - rootsofginC[i]);
ikkk:=i;
end if;
end for;

PreimageOfzetaj:=0;
for i:= 1 to n do
PreimageOfzetaj:=PreimageOfzetaj
+ zeta[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfalphaj:=0;
for i:= 1 to n do
PreimageOfalphaj:=PreimageOfalphaj
+ alpha[i]*ImageOfIntegralBasisElementF[ijjj][i];
end for; //i
PreimageOfzetak:=0;
for i:= 1 to n do
PreimageOfzetak:=PreimageOfzetak
```

```
+ zeta[i]*ImageOfIntegralBasisElementF[ikkk][i];
end for; //i
PreimageOfalphak:=0;
for i:= 1 to n do
PreimageOfalphak:=PreimageOfalphak
+ alpha[i]*ImageOfIntegralBasisElementF[ikkk][i];
end for; //i

/*
Note that:
preimage of pi_{i}^{(j)} = ImageOfpiF[i][kk[i]][ijjj]
preimage of pi_{i}^{(k)} = ImageOfpiF[i][kk[i]][ikkk]
preimage of eps_{i}^{(j)} = ImageOfepsF[i][ijjj]
preimage of eps_{i}^{(k)} = ImageOfepsF[i][ikkk]
*/

//////////////////////////////////////
/*
Define the alpha_i for i in J, construct L, and find the prime ideal of
L corresponding to sigma (see Section 11).
*/
//////////////////////////////////////
alphaALGEBRAIC:=[FFF|];
for i:=1 to 1 + v + r do
alphaALGEBRAIC[i]:=1; //initialize
end for;
alphaALGEBRAIC[1]:=(thetaF[ii0] - thetaF[ijjj]
/ thetaF[ii0] - thetaF[ikkk])*(PreimageOfalphak/PreimageOfalphaj)
*(PreimageOfzetak/PreimageOfzetaj);
for i in J do
alphaALGEBRAIC[1+i]:=ImageOfpiF[i][kk[i]][ikkk]
/ImageOfpiF[i][kk[i]][ijjj];
end for;
for i:=1 to r do
alphaALGEBRAIC[1+v+i]:=ImageOfepsF[i][ikkk]/ImageOfepsF[i][ijjj];
```

```
end for;

LL:=sub<FFF | alphaALGEBRAIC>;
d:=Degree(LL);
mm:=2+#J+r;


////////////////////////////////////
/*
Compute c17.

Also compute the alpha_i as defined in Sections 18 and 20:
LogarithmicAlphaC[i0i0i0][i]
*/
////////////////////////////////////
LogarithmicAlphaC[i0i0i0]:=[RealField() | ];
for i:=1 to 1+v + r do
LogarithmicAlphaC[i0i0i0][i]:=0;
end for;


temp:=((thetaC[i0i0i0] - thetaC[jjjjjj[i0i0i0]])
/(thetaC[i0i0i0] - thetaC[kkkkkk[i0i0i0]]))
*(ImageOfzetaC[kkkkkk[i0i0i0]]/ImageOfzetaC[jjjjjj[i0i0i0]])
*(ImageOfalphaC[kkkkkk[i0i0i0]]/ImageOfalphaC[jjjjjj[i0i0i0]]);
LogarithmicAlphaC[i0i0i0][1]:=Imaginary(Log(temp));
H:=Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1]),
Abs(Log(temp)), 0.16] );

for i in J do
temp:=ImageOfpiC[i][kk[i]][kkkkkk[i0i0i0]]
/ImageOfpiC[i][kk[i]][jjjjjj[i0i0i0]];
LogarithmicAlphaC[i0i0i0][1+i]:=Imaginary(Log(temp));
H:=H*Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1+i]),
Abs(Log(temp)), 0.16] );
```

```
end for;

for i:=1 to r do
temp:=ImageOfepsC[i][kkkkkk[i0i0i0]]/ImageOfepsC[i][jjjjjj[i0i0i0]];
LogarithmicAlphaC[i0i0i0][1+v+i]:=Imaginary(Log(temp));
H:=H*Max( [d*AbsoluteLogarithmicHeight(alphaALGEBRAIC[1+v+i]),
Abs(Log(temp)), 0.16] );
end for;

LogarithmicAlphaC[i0i0i0][2+v+r]:=PI; // = Im Log(-1)

H:=H*Max([d*0, PI, 0.16]); //AbsoluteLogarithmicHeight(-1) = 0

c17[i0i0i0]:=
Min([(1/kappa)*((Exp(1)*mm/2)^(kappa))*(30^(mm+3))*(mm^(3.5)),
(2^(6*mm+20))])*(d^2)*(1+Log(d))*H;


//////////////////////////////
/*
Compute c18
*/
//////////////////////////////

c18[i0i0i0]:= Log(4*Arcsin(1/4)*c16[i0i0i0]) + c17[i0i0i0]
+ c17[i0i0i0]*Log(#JJJ);


end for; //end i0i0i0 loop
end if;

////////////////////////////////////////////////////////////////////////
/*
Done computing the indices j=j(i0), k=k(i0), and the numbers c16=c16(i0),
c17=c17(i0), c18=c18(i0) for every i0 in {1,...,s} (when s > 0)
```

```
*/
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//////////////////
/*
Choose c11 to minimize the value of c22, the upper bound for H.  In the
process, compute the constants that depend on c11:
c11, c12, c13, c14, c15, c19, c20, c21, c22
*/
//////////////////
////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

if s gt 0 then

c11:=(1/1000000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;

c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
```

```
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

c19:=Ceiling(Log(2*c16[1])/c11)-1;
max1:=c17[1]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[1]/c11) + max1*Log(max1) );
for i0i0i0:=2 to s do
c19:=Ceiling(Log(2*c16[i0i0i0])/c11)-1;
max1:=c17[i0i0i0]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[i0i0i0]/c11) + max1*Log(max1) );
end for;
c20:=Ceiling(c20)-1;

c21:=Max([c15,c19,c20,1]);

c22:=Max([c7,c14,c21]);

////

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;
```

```
c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

c19:=Ceiling(Log(2*c16[1])/c11)-1;
max1:=c17[1]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[1]/c11) + max1*Log(max1) );
for i0i0i0:=2 to s do
c19:=Ceiling(Log(2*c16[i0i0i0])/c11)-1;
max1:=c17[i0i0i0]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[i0i0i0]/c11) + max1*Log(max1) );
end for;

c21:=Max([c15,c19,c20,1]);

if Max([c7,c14,c21]) gt c22 then c22:=Max([c7,c14,c21]); end if;

////
```

```
for i:=1 to 999 do

c11:=(i/1000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;

c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

c19:=Ceiling(Log(2*c16[1])/c11)-1;
max1:=c17[1]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[1]/c11) + max1*Log(max1) );
for i0i0i0:=2 to s do
```

```
c19:=Ceiling(Log(2*c16[i0i0i0])/c11)-1;
max1:=c17[i0i0i0]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[i0i0i0]/c11) + max1*Log(max1) );
end for;

c21:=Max([c15,c19,c20,1]);

if Max([c7,c14,c21]) gt c22 then c22:=Max([c7,c14,c21]); end if;

end for; //end i

////////////////////////////////////////////////////////////
else //s = 0
////////////////////////////////////////////////////////////

c11:=(1/1000000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;

c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
```

```
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

c19:=Ceiling(Log(2*c16[1])/c11)-1;
max1:=c17[1]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[1]/c11) + max1*Log(max1) );
for i0i0i0:=2 to s do
c19:=Ceiling(Log(2*c16[i0i0i0])/c11)-1;
max1:=c17[i0i0i0]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[i0i0i0]/c11) + max1*Log(max1) );
end for;

c21:=Max([c15,c19,c20,1]);

c22:=Max([c7,c14,c21]);

////

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;

c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
```

```
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

c19:=Ceiling(Log(2*c16[1])/c11)-1;
max1:=c17[1]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[1]/c11) + max1*Log(max1) );
for i0i0i0:=2 to s do
c19:=Ceiling(Log(2*c16[i0i0i0])/c11)-1;
max1:=c17[i0i0i0]/c11;
if max1 lt Exp(2) then max1:=Exp(2); end if;
c20:=2*( (c18[i0i0i0]/c11) + max1*Log(max1) );
end for;

c21:=Max([c15,c19,c20,1]);

if Max([c7,c14,c21]) gt c22 then c22:=Max([c7,c14,c21]); end if;

////

for i:=1 to 999 do
```

```
c11:=(i/1000)*c10/(n-1);

c12prime:=(c8prime + c5*c9prime)/(c10 - (n-1)*c11);
c13prime:=c4*c9prime/(c10 - (n-1)*c11);
if c13prime lt Exp(2) then c13prime:=Exp(2); end if;
c12primeprime:=(c8primeprime + c5*c9primeprime)/(c10 - c11);
c13primeprime:=c4*c9primeprime/(c10 - c11);
if c13primeprime lt Exp(2) then c13primeprime:=Exp(2); end if;

c14:=2*Ceiling(c12prime + c13prime*Log(c13prime))-1;
if c14 lt 2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1
then
c14:=2*Ceiling(c12primeprime + c13primeprime*Log(c13primeprime))-1;
end if;
if c14 lt 2 then c14:=2; end if;

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

if Max([c7,c14,c15]) gt c22 then c22:=Max([c7,c14,c15]); end if;

end for; //end i

end if;

//////////////////////////////////////////////////////////////////////
```

```
/*
UpperBoundForH:= current best upper bound for H = H_0 from Section 15.

UpperBoundForA:= current best upper bound for A = A_0 from Section 15.

ConditionalUpperBoundForA[i0]:= current best i_0-conditonal upper bound
for A
= A_0(i_0) from Section 19.

UpperBoundForn[l]:= current best upper bound for n_l = N_l from Section
15.

UpperBoundForN:= current best upper bound for N

For l in I2, N_{l}^{*} bounds n_l.
For l in I, we use Corollary 14.2.

B[i]:= current best upper bound for |b_i| = B_i from section 15.
*/
////////////////////////////////////////////////////////////////////////
UpperBoundForn:=[];
ConditionalUpperBoundForA:=[];
UpperBoundForH:=c22;
UpperBoundForA:=UpperBoundForH;
for i0:=1 to s do
ConditionalUpperBoundForA[i0]:=UpperBoundForA;
end for;
for l:=1 to v do
UpperBoundForn[l]:=1; //initialize
end for;
for l in I2 do
UpperBoundForn[l]:=Nstar[l];
end for;
for l in I do
UpperBoundForn[l]:=
```

```
Ceiling( (c1[l]/hh[l])*Log(c22) - (1/hh[l])*Valuation(delta2[l]) ) - 1;
if 2 gt UpperBoundForn[l] then UpperBoundForn[l]:=2; end if;
if c22 lt UpperBoundForn[l] then UpperBoundForn[l]:=c22; end if;
end for;

UpperBoundForN:=Max(UpperBoundForn);

B:=[];
for i:=1 to 1+v+r do
B[i]:=0; //initialize
end for;
B[1]:=1;
for i in J do
B[1+i]:=UpperBoundForn[i];
end for;
for i:=1 to r do
B[1+v+i]:=UpperBoundForA;
end for;

//////////////////////////////////////////
/*
If UpperBoundForA < 0, then we know there are no solutions for the
current case of (11).

If UpperBoundForn[i] < 0 for some i in J, then we know there are no
solutions for the current case of (11).
*/
//////////////////////////////////////////

print("Starting basic reduction procedures.");

////////////////////////////////////////////////////////////////////////
//////////////////////
/*
Start Of Loop For Repeated Simple Reductions
```

```
*/
//////////////////////
////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
Improvement:=true;

while Improvement do
Improvement:=false;

//////////////////////////////////////////////////////////////////////////
/*
Basic p_l-adic reduction for each l in I (Section 18)
*/
//////////////////////////////////////////////////////////////////////////

for l in I do
//print("basic p_l, l=");
//l;

//////////////////////////////////////////////////////////////////////////
/*
Define:
W[i] = W_i (weights) for p_l (Section 18)
QQ = Q (Section 18)
mmmm = m for p_l (Section 18)

Initialize:
betam[i] = beta_i^(m) for p[L],
Tm, where Transpose(Tm) = U_m from Section 18
*/
//////////////////////////////////////////////////////////////////////////
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;
```

```
W[1]:=0;
for i:=2 to #JJJ do
W[JJJ[i]]:=RoundP(UpperBoundForH/B[JJJ[#JJJ]]);
end for;
//print("weights:");
//W;

//print("B");
//B;

QQ:=0;
for i in JJJ do
QQ:=QQ+(W[i]^2)*(B[i]^2); //recall W[1]=0
end for;

prod:=1;
for i:=2 to #JJJ do
prod:=prod*W[JJJ[i]];
end for;

/*
mmmm:=Ceiling(
( (#JJJ - 1)/(2*Log(p[l])) )*Log( 2^(#JJJ - 1)
* QQ / prod^(2/(#JJJ - 1)) )
);
*/
mmmm:=Ceiling(
( (#JJJ - 1)/(2*Log(p[l])) )*Log( QQ / prod^(2/(#JJJ - 1)) )
);

betam:=[];
for i:=1 to 1+v+r do
betam[i]:=0;
end for;
```

```
Tm:=ScalarMatrix(IntegerRing(),#JJJ-1,1);
/*initialize as #JJJ-1 by #JJJ-1 identity matrix*/

////////////////////////////////////////////////////////////////////////
/*
Start while loop that will increase m until the p_l-adic reduction yeilds
a new upper bound for n_l or until a number of increases has been made
with no success
*/
////////////////////////////////////////////////////////////////////////
flag:=true;
RunThroughNumber:=0;
while flag do
//print("RunThroughNumber");
//RunThroughNumber;

////////////////////////////////////////////////////////////////////////
/*
Increase mmmm = m (if this is not the first attempt with the basic
p_l-reduction procedure).
Calculuate betam[i] = beta_i^(m).
*/
////////////////////////////////////////////////////////////////////////
mmmm:=Ceiling( mmmm + (RunThroughNumber/20)*mmmm );
//print("mmmm");
//mmmm;

if mmmm gt (0.95)*padicprecision[l] then
PadicPrecisionMultiplier[l] := PadicPrecisionMultiplier[l] + 1;
continue PrecisionLoopVariable;
end if;

for i in JJJ do
betam[i]:= SmallestNonnegativeRemainderModpLToThem(beta[l][i],l,mmmm);
```

```
end for;

////////////////////////////////////////////////////////////////////////
/*
Define:
Am = A_m for p[L]
Gamma_m = lattice generated by columns of A_m
*/
////////////////////////////////////////////////////////////////////////


Am:=ZeroMatrix(IntegerRing(), v+r+2, v+r+2);
for i:=1 to v+r+1 do
Am[i][i]:=W[i];
end for;
for j:=1 to v+r+1 do
Am[v+r+2][j]:=W[ihat[l]]*betam[j];
end for;
Am[v+r+2][v+r+2]:=W[ihat[l]]*p[l]^mmmm;

for i:=1+v+r to 2 by -1 do
if i eq ihat[l] or not i in JJJ then
RemoveColumn(~Am,i);
RemoveRow(~Am,i);
end if;
end for;

RemoveColumn(~Am,1);
RemoveRow(~Am,1);

//Am is now a #JJJ-1 by #JJJ-1 matrix

Am:=Am*Transpose(Tm); //Transpose(Tm) = Um from Setion 15
/* If this is the first run through, Tm is the identity. If this is not
the first run through, this will save computation time.
```

```
*/


////////////////////////////////////////////////////////////////////////
/*
Compute an LLL reduced basis for the lattice Gamma_m generated by columns
of A_m.  The algorithm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_m.  Similarly, it spits out the transpose of B_m and
the transpose of U_m
*/
////////////////////////////////////////////////////////////////////////

//time
temp,Tm,rank:=LLL( Transpose(Am) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

Bm:=MatrixRing(RationalField(),#JJJ-1) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_m

/*
Here Tm*Transpose(Am) = Transpose(Bm).
So with Um=Transpose(Tm) we have Am*Um = Bm,
and With
Vm = Bm*Transpose(Tm)*Bm^(-1) = Am*Transpose(Tm)*Am^(-1), we have
Vm*Am = Bm.
*/

////////////////////////////////////////////////////////////////////////
/*
Compute the lower bound for l(Gamma_m,y) using Lemma 16.1.  If it is
large enough (bigger than sqrt(Q)), compute the new upper bound for
n[L].  Otherwise, increase m and reduce again (go back to the start of
the while loop).
```

```
*/
////////////////////////////////////////////////////////////////////////
lowerbound:=0;

//START IF
if betam[1] eq 0 then //y = 0 case (vectors)

//c1 = Transpose(Bm)[1]
lowerbound:= 2^(-(#JJJ-2)/2)*Norm(Transpose(Bm)[1])^(1/2);

else   //y neq 0 case

yyy:=ZeroMatrix(RationalField(),#JJJ-1,1);
yyy[#JJJ-1][1]:=-W[ihat[l]]*betam[1];
sss:=(Bm^(-1))*yyy;

IndicesjWithsjNotIntegral:=[];
for j:=1 to #JJJ-1 do
if not IsIntegral(sss[j][1]) then
Append(~IndicesjWithsjNotIntegral,j);
end if;
end for;

delta:=[RealField() | ];
for j:=1 to #JJJ-2 do
delta[j]:=0;
for i:=j+1 to #JJJ-1 do
if delta[j] lt
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(Bm)[i])^(1/2) then
delta[j]:=
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(Bm)[i])^(1/2);
end if;
end for;
end for;
delta[#JJJ-1]:=0;
```

```
max:=0;
for j in IndicesjWithsjNotIntegral do
if max lt
2^(-(#JJJ-2)/2) * DistanceToNearestInteger(sss[j][1])
* Norm(Transpose(Bm)[1])^(1/2) - (#JJJ - 1 - j)*delta[j]
then
max:=2^(-(#JJJ-2)/2) * DistanceToNearestInteger(sss[j][1])
* Norm(Transpose(Bm)[1])^(1/2) - (#JJJ - 1 - j)*delta[j];
end if;
end for;
lowerbound:=max;

end if;
//END IF

if lowerbound gt QQ^(1/2) then

flag:=false; //ready to get out of while loop that increases m if
             //necessary

OldUpperBoundFornl:=UpperBoundForn[l];

NewUpperBoundFornl:= Max([
Floor( (1/hh[l])*(1/(p[l] - 1) - Valuation(delta2[l])) ),
Ceiling((1/hh[l])*(mmmm-dd[l]))-1,
0
]);

if NewUpperBoundFornl lt OldUpperBoundFornl then
Improvement:=true;
UpperBoundForn[l]:=NewUpperBoundFornl;
B[1+l]:=UpperBoundForn[l];
end if;
```

```
else

RunThroughNumber+:=1;  //increase m and try again
///
/*
If increasing m 20 times (in which case m will be double its original
value) fails to produce a new upper bound for n_l, move onto the next
value of l in I.  Also, print a message indicating that the basic
p_l-adic reduction procedure was unsuccesful.
*/
/////
if RunThroughNumber eq 20 then
print("Basic p-adic reduction taking too long");
print("Case:");
iiii;
print("l:");
l;
continue l;
end if;

end if; /* IF controlled by lowerbound gt QQ^(1/2) */

end while; /*this is the loop that increases m if necessary, the while
loop controlled by flag*/

if UpperBoundForn[l] lt 0 then
//there are no solutions of the current case of (11)
continue iiii;
end if;
end for; // end l loop

UpperBoundForN:=Max(UpperBoundForn);

/*
print("Starting basic real reduction");
```

```
print("Log_10 of Upper Bound For A");
Log(UpperBoundForA)/Log(10);
*/

////////////////////////////////////////////////////////////////////////
/*
Basic Real Reduction (Section 19)
*/
////////////////////////////////////////////////////////////////////////
for i0:=1 to s do
ConditionalUpperBoundForA[i0]:=UpperBoundForA;
end for;

////////////////////////////////////////////////////////////////////////
/*
Totally Complex Case, i.e, s=0
*/
////////////////////////////////////////////////////////////////////////
if s eq 0 then

//Choose c11 to find an choose optimal upper bound for A from Lemma 19.1
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
```

```
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1
then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1 with a certain value of
//c11
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1
then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1 with a certain value of
//c11
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
```

```
c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1 with a certain value of
//c11
if max lt MIN then MIN:=max; end if;
end for;  //end i loop


OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=MIN;


if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
UpperBoundForA:=NewUpperBoundForA;
end if;

else //s > 0
////////////////////////////////////////////////////////////////////////
/*
s>0 Case
*/
```

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/*
Real Case, i.e., s \geq 3
*/
////////////////////////////////////////////////////////////////////////

if s ge 3 then // s>=3 real case

JumpToEndOfRealReduction:=false;
for i0:=1 to s do
//print("basic real, i0=");
//i0;

////////////////////////////////////////////////////////////////////////
/*
Define
weights W[i] (Section 19)
R (Section 19)
S (Section 19)
CCCCCC = C (Section 19)

Initialize:
TC, where Transpose(TC) = U_C from Section 19
phi[i]:= phi_i from Section 19
*/
////////////////////////////////////////////////////////////////////////
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;

H0prime:=0;
for i:=2 to #JJJ-1 do
if H0prime lt B[JJJ[i]] then H0prime:=B[JJJ[i]]; end if;
```

```
end for;

W[1]:=0;
for i:=2 to #JJJ-1 do
W[JJJ[i]]:=RoundP(H0prime/B[JJJ[i]]);
end for;
//print("weights");
//W;
//print("B");
//B;

R:=0;
for i in JJJ do
R:=R + B[i];
end for;
R:=(1/2)*R;
//this is just an approximation to R

S:=0;
for i:=2 to #JJJ-1 do
S:=S+(W[JJJ[i]]^2)*(B[JJJ[i]]^2);
end for;

prod:=1;
for i:=2 to #JJJ-1 do
prod:=prod*W[JJJ[i]];
end for;

/*
LogC:= ((#JJJ-1)/2)*Log(
(R^2 + S) / ( 2^(-(#JJJ-1)) *
(Abs(LogarithmicAlphaC[i0][JJJ[#JJJ]])*prod)^(2/(#JJJ-1)) )
);
*/
```

```
LogC:= ((#JJJ-1)/2)*Log(
(R^2 + S) / ( 2^(-2*(#JJJ-1))
* (Abs(LogarithmicAlphaC[i0][JJJ[#JJJ]])*prod)^(2/(#JJJ-1)) )
);


/*
LogC/Log(10);

LogC:=430;

LogC := ((#JJJ-1)/2)*Log((#JJJ-1)
* Max([UpperBoundForA,Max(UpperBoundForn)])^2);

LogC/Log(10);

break iiii;
*/


//CCCCCC:=Ceiling(Exp(LogC));


TC:=ScalarMatrix(IntegerRing(),#JJJ-1,1);
//#JJJ-1 by #JJJ-1 identity matrix

phi:=[];
for i:=1 to 1+v+r do
phi[i]:=0;
end for;

////////////////////////////////////////////////////////////////////////
/*
Start the while loop where C will be increased until a new conditional
upper for A is found that improves on the uncondtional upper bound for A
```

```
or until a number of increases of C have been made with no success.
*/
////////////////////////////////////////////////////////////////////////
RunThroughNumber1:=0;
RunThroughNumber2:=0;
flag:=true;
while flag do


////////////////////////////////////////////////////////////////////////
/*
Increase CCCCCC = C (if this is not the first attempt the basic real
reduction procedure).

Calculate the phi_i for i in JJJ.

Calculate R.
*/
////////////////////////////////////////////////////////////////////////
CCCCCC:=
Ceiling(Exp(LogC + ((RunThroughNumber1 + RunThroughNumber2)/20)*LogC));
if RunThroughNumber1 eq 21 and UpperBoundForA gt 100000 then
CCCCCC:=Ceiling(Exp(LogC + 9*LogC));
end if;
if RunThroughNumber1 eq 22 and UpperBoundForA gt 100000 then
CCCCCC:=Ceiling(Exp(LogC + 9*LogC));
end if;

if LogC/Log(10) gt (0.95)*realprecision then
RealPrecisionMultiplier := RealPrecisionMultiplier + 1;
continue PrecisionLoopVariable;
end if;

//print("Log_10 (C)");
//Floor(LogC/Log(10));
```

```
for i in JJJ do
phi[i]:=Round(CCCCCC*LogarithmicAlphaC[i0][i]);
end for;
phi[1]:=Round(CCCCCC*LogarithmicAlphaC[i0][1]);

if Abs(phi[JJJ[#JJJ]]) lt 2 then
if CCCCCC*LogarithmicAlphaC[i0][JJJ[#JJJ]] ge 0 then
phi[JJJ[#JJJ]]:=2;
else
phi[JJJ[#JJJ]]:=-2;
end if;
end if;

if IsIntegral(phi[1]/phi[JJJ[#JJJ]]) then
if Abs(phi[1]+1 - CCCCCC*LogarithmicAlphaC[i0][1]) le 1 then
phi[1]:=phi[1]+1;
else
phi[1]:=phi[1]-1;
end if;
end if;

R:=0;
for i in JJJ do
R:=R + B[i]*Abs( CCCCCC*LogarithmicAlphaC[i0][i] -  phi[i] );
end for;

////////////////////////////////////////////////////////////////////////
/*
Compute:
AC = A_C
*/
////////////////////////////////////////////////////////////////////////
AC:=ZeroMatrix(IntegerRing(),v+r+1,v+r+1);
```

```
for i:=1 to v+r do
AC[i][i]:=W[i];
end for;
for j:=1 to v+r+1 do
AC[v+r+1][j]:=phi[j];
end for;

for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveColumn(~AC,i);
RemoveRow(~AC,i);
end if;
end for;

RemoveRow(~AC,1);
RemoveColumn(~AC,1);

//AC is now a #JJJ-1 by #JJJ-1 matrix

AC:=AC*Transpose(TC);
/* If this is the first run through Transpose(TC) is the identity.
Otherwise Transpose(TC) is UC.  This will time in the LLL reduction. */
//print("AC[#JJJ-1]:");
//AC[#JJJ-1];

////////////////////////////////////////////////////////////////////////
/*
Compute LLL reduced basis for the lattice Gamma_C generated by columns of
A_C.  The algoritm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_C.  Similarly, it spits out the transpose of B_C and
the transpose of U_C
*/
////////////////////////////////////////////////////////////////////////
```

```
//time
temp,TC,rank:=LLL(Transpose(AC) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

BC:=MatrixRing(RationalField(),#JJJ-1) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_C

/*
Here TC*Transpose(AC) = Transpose(BC).
So with UC=Transpose(TC) we have AC*UC = BC,
and With
VC = BC*Transpose(TC)*BC^(-1) = AC*Transpose(TC)*AC^(-1), we have
VC*AC = BC.
*/

////////////////////////////////////////////////////////////////////////
/*
Compute the lower bound for l(Gamma_C,y).  If it is large enough
(bigger than sqrt(R^2 + S)), compute the new upper bound for n[L].
Otherwise, increase C and reduce again (go back to the start of the while
loop)
*/
////////////////////////////////////////////////////////////////////////

//START IF
if phi[1] eq 0 then //y = 0 case (vectors)
                    //never happens by choice of phi[1]

// \vec{c}_1 = Transpose(BC)[1]
lowerbound:= 2^(-(#JJJ-2)/2)*Norm( Transpose(BC)[1] )^(1/2);
print("lowerbound, y=0:");
lowerbound;
else    //y neq 0 case

yyy:=ZeroMatrix(RationalField(),#JJJ-1,1);
```

```
yyy[#JJJ-1][1]:=-phi[1];
sss:=(BC^(-1))*yyy;

IndicesjWithsjNotIntegral:=[];
for j:=1 to #JJJ-1 do
if not IsIntegral(sss[j][1]) then
Append(~IndicesjWithsjNotIntegral,j);
end if;
end for;

delta:=[RealField() | ];
for j:=1 to #JJJ-2 do
delta[j]:=0;
for i:=j+1 to #JJJ-1 do
if delta[j] lt
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(BC)[i])^(1/2) then
delta[j]:=
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(BC)[i])^(1/2);
end if;
end for;
end for;
delta[#JJJ-1]:=0;

max:=0;
for j in IndicesjWithsjNotIntegral do
if max lt
2^(-(#JJJ-2)/2) * DistanceToNearestInteger(sss[j][1])
* Norm(Transpose(BC)[1])^(1/2) - (#JJJ - 1 - j)*delta[j] then
max:=2^(-(#JJJ-2)/2) * DistanceToNearestInteger(sss[j][1])
* Norm(Transpose(BC)[1])^(1/2) - (#JJJ - 1 - j)*delta[j];
end if;
end for;
lowerbound:=max;

/*
```

```
print("lowerbound, y ne 0:");
Floor(lowerbound);
print("IndicesjWithsjNotIntegral");
IndicesjWithsjNotIntegral;
*/

/*
print("DistanceToNearestInteger(sss[#JJJ-1][1]);");
RealField() ! DistanceToNearestInteger(sss[#JJJ-1][1]);
print("DistanceToNearestInteger(sss[#JJJ-2][1]);");
RealField() ! DistanceToNearestInteger(sss[#JJJ-2][1]);
*/

/*
print("Norm(Transpose(BC)[1]^(1/2))");
Norm(Transpose(BC)[1])^(1/2);
print("2^(-(#JJJ-2)/2)");
2^(-(#JJJ-2)/2);
*/

end if;
//END IF

/*
print("(R^2 + S)^(1/2):");
Floor((R^2 + S)^(1/2));
*/
/*
print("lowerbound / (R^2 + S)^(1/2)");
lowerbound / (R^2 + S)^(1/2);
*/

if lowerbound gt (R^2 + S)^(1/2) then

//////////
```

```
/*
Choose c11 to find an optimal i0-conditional upper bound for A from Lemma
19.2
*/
//////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

c11:=((1000000-1)/1000000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
```

```
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;
end for; //end i loop
////


OldUnconditionalUpperBoundForA:=UpperBoundForA;
NewConditionalUpperBoundForAi0:=MIN;

if NewConditionalUpperBoundForAi0 lt OldUnconditionalUpperBoundForA then
ConditionalUpperBoundForA[i0]:=NewConditionalUpperBoundForAi0;
flag:=false; //ready to get out of while loop that increases C if
             //necessary
else
RunThroughNumber2:=RunThroughNumber2+1; //increase C and try again
////////
/*
In case we find a new conditional upper bound that fails to improve on
the current unconditional upper bound for A, and this occurs 5 times wtih
log(C) being increased 5% each time, then we abort the real reduction
procedure.  There is no need to print a message in this case.
*/
///////
if RunThroughNumber2 eq 5 then
JumpToEndOfRealReduction:=true;
break i0;
end if;
end if; /*controlled by NewConditionalUpperBoundForAi0 lt
          OldUnconditionalUpperBoundForA*/

else //controlled by lowerbound gt (R^2 + S)^(1/2)
```

```
RunThroughNumber1:=RunThroughNumber1+1; //increase C and try again
/////
/*
If increasing log(C) 22 times (with log(C) being increased 5% the first
twenty times and by 1000% the last two times, so that log(C) will be
20 times original value) fails to produce a new conditional upper
bound for A, then we abort the real reduction procedure.  Also, print
a message to the user indicating the basic real reduction procedure
was unsuccesful for this reason.  Note: tries 21 and 22 only happen
if the upper for A has not been reduced much (or at all) from the linear
forms in logs bound.
*/
////
if RunThroughNumber1 eq 23 then
print("basic real reduction taking too long");
print("case");
iiii;
print("i0");
i0;
print("Improvement:");
Improvement;
UpperBoundForn;
UpperBoundForA;
JumpToEndOfRealReduction:=true;

/*
GammaC:=Lattice(Transpose(AC));
w:=CoordinateSpace(GammaC) ! [0,yyy[2][1]];
ClosestVectorsMatrix(GammaC,w : Max:=5);
*/

break i0;
end if;

end if; //controlled by lowerbound gt (R^2 + S)^(1/2)
```

```
end while; /* This is the loop that increases log(C) if necessary. The
while loop controlled by flag */

end for; //end i0 loop


if JumpToEndOfRealReduction eq false then
//////////////
/*
Choose c11 to find an optimal i0-conditional upper bound for A for i0 in
{s+1,...,s+2t} from Lemma 19.1
*/
///////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
```

```
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
```

```
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;
end for;  //end i loop

/////
/*
Compute the new unconditional bound for A
*/
/////

OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=Max([Max(ConditionalUpperBoundForA),MIN]);

if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
UpperBoundForA:=NewUpperBoundForA;
for i:=1 to r do
B[1+v+i]:=UpperBoundForA;
end for;
end if;

end if; //controlled by JumpToEndOfRealReduction

if UpperBoundForA lt 0 then
//there are no solutions to the current case of (11)
```

```
continue iiii;
end if;



////////////////////////////////////////////////////////////////////////
/*
Complex Case, i.e., s = 1,2
*/
////////////////////////////////////////////////////////////////////////
else //s=1,2 complex case

B[2+v+r]:=Floor(2*Arcsin(1/4)/PI);
for i in JJJ do
B[2+v+r]:=B[2+v+r] + B[i];
end for;


JumpToEndOfRealReduction:=false;
for i0:=1 to s do


////////////////////////////////////////////////////////////////////////
/*
Define
weights W[i] (Section 19)
R (Section 19)
S (Section 19)
CCCCCC = C (Section 19)

Initialize:
TC, where Transpose(TC) = U_C from Section 19
phi[i]:= phi_i from Section 19
*/
////////////////////////////////////////////////////////////////////////
```

```
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;

H0prime:=0;
for i:=2 to #JJJ do
if H0prime lt B[JJJ[i]] then H0prime:=B[JJJ[i]]; end if;
end for;

W[1]:=0;
for i:=2 to #JJJ do
W[JJJ[i]]:=RoundP(H0prime/B[JJJ[i]]);
end for;
//print("weights");
//W;
//print("B");
//B;

R:=B[2+v+r];
for i in JJJ do
R:=R + B[i];
end for;
R:=(1/2)*R;
//this is just an approximation to R

S:=0;
for i:=2 to #JJJ do
S:=S+(W[JJJ[i]]^2)*(B[JJJ[i]]^2);
end for;

prod:=1;
for i:=2 to #JJJ do
prod:=prod*W[JJJ[i]];
end for;
```

```
/*
LogC:= ((#JJJ)/2)*Log(
(R^2 + S) / ( 2^(-(#JJJ))
* (Abs(LogarithmicAlphaC[i0][2+v+r])*prod)^(2/(#JJJ)) )
);
*/
LogC:= ((#JJJ)/2)*Log(
(R^2 + S) / ( 2^(-2*(#JJJ))
* (Abs(LogarithmicAlphaC[i0][2+v+r])*prod)^(2/(#JJJ)) )
);


//CCCCCC:=Ceiling(Exp(LogC));

TC:=ScalarMatrix(IntegerRing(),#JJJ,1);
//#JJJ by #JJJ identity matrix

phi:=[];
for i:=1 to 2+v+r do
phi[i]:=0;
end for;

////////////////////////////////////////////////////////////////////////
/*
Start the while loop where C will be increased until a new conditional
upper for A is found that improves on the uncondtional upper bound for A
or until a number of increases of C have been made with no success.
*/
////////////////////////////////////////////////////////////////////////
RunThroughNumber1:=0;
RunThroughNumber2:=0;
flag:=true;
while flag do
```

```
////////////////////////////////////////////////////////////////////////
/*
Increase CCCCCC = C (if this is not the first attempt the basic real
reduction procedure).

Calculate the phi_i for i in JJJ.

Calculate R.
*/
////////////////////////////////////////////////////////////////////////
CCCCCC:=
Ceiling(Exp(LogC + ((RunThroughNumber1 + RunThroughNumber2)/20)*LogC));
if RunThroughNumber1 eq 21 and UpperBoundForA gt 100000 then
CCCCCC:=Ceiling(Exp(LogC + 9*LogC));
end if;
if RunThroughNumber1 eq 22 and UpperBoundForA gt 100000 then
CCCCCC:=Ceiling(Exp(LogC + 19*LogC));
end if;

if LogC/Log(10) gt (0.95)*realprecision then
RealPrecisionMultiplier := RealPrecisionMultiplier + 1;
continue PrecisionLoopVariable;
end if;

for i in JJJ do
phi[i]:=Round(CCCCCC*LogarithmicAlphaC[i0][i]);
end for;
phi[2+v+r]:=Round(CCCCCC*LogarithmicAlphaC[i0][2+v+r]);

/*This will never happen because LogarithmicAlphaC[i0][2+v+r] = Pi */
if Abs(phi[2+v+r]) lt 2 then
if CCCCCC*LogarithmicAlphaC[i0][2+v+r] ge 0 then
phi[2+v+r]:=2;
else
phi[2+v+r]:=-2;
```

```
end if;
end if;

if IsIntegral(phi[1]/phi[2+v+r]) then
if Abs(phi[1]+1 - CCCCCC*LogarithmicAlphaC[i0][1]) le 1 then
phi[1]:=phi[1]+1;
else
phi[1]:=phi[1]-1;
end if;
end if;

R:=B[2+v+r]*Abs( CCCCCC*LogarithmicAlphaC[i0][2+v+r] -  phi[2+v+r] );
for i in JJJ do
R:=R + B[i]*Abs( CCCCCC*LogarithmicAlphaC[i0][i] -  phi[i] );
end for;

////////////////////////////////////////////////////////////////////////
/*
Compute:
AC = A_C
*/
////////////////////////////////////////////////////////////////////////
AC:=ZeroMatrix(IntegerRing(),v+r+2,v+r+2);

for i:=1 to 1+v+r do
AC[i][i]:=W[i];
end for;
for j:=1 to v+r+2 do
AC[v+r+2][j]:=phi[j];
end for;

for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveColumn(~AC,i);
RemoveRow(~AC,i);
```

```
end if;
end for;

RemoveRow(~AC,1);
RemoveColumn(~AC,1);

//AC is now a #JJJ by #JJJ matrix

AC:=AC*Transpose(TC);
/* If this is the first run through Transpose(TC) is the identity.
Otherwise Transpose(TC) is UC.  This will time in the LLL reduction. */

////////////////////////////////////////////////////////////////////////
/*
Compute LLL reduced basis for the lattice Gamma_C generated by columns of
A_C.  The algoritm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_C.  Similarly, it spits out the transpose of B_C and
the transpose of U_C
*/
////////////////////////////////////////////////////////////////////////
//time
temp,TC,rank:=LLL(Transpose(AC) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

BC:=MatrixRing(RationalField(),#JJJ) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_C

/*
Here TC*Transpose(AC) = Transpose(BC).
So with UC=Transpose(TC) we have AC*UC = BC,
and With
VC = BC*Transpose(TC)*BC^(-1) = AC*Transpose(TC)*AC^(-1), we have VC*AC =
BC.
```

```
*/

////////////////////////////////////////////////////////////////////////
/*
Compute the lower bound for l(Gamma_C,y).  If it is large enough (bigger
than sqrt(R^2 + S)), compute the new upper bound for n[L].  Otherwise,
increase C and reduce again (go back to the start of the while loop)
*/
////////////////////////////////////////////////////////////////////////

//START IF
if phi[1] eq 0 then //y = 0 case (vectors) //never happens by choice of
                                           //phi[1]

// \vec{c}_1 = Transpose(BC)[1]
lowerbound:= 2^(-(#JJJ-1)/2)*Norm( Transpose(BC)[1] )^(1/2);

else    //y neq 0 case

yyy:=ZeroMatrix(RationalField(),#JJJ,1);
yyy[#JJJ][1]:=-phi[1];
sss:=(BC^(-1))*yyy;

IndicesjWithsjNotIntegral:=[];
for j:=1 to #JJJ do
if not IsIntegral(sss[j][1]) then
Append(~IndicesjWithsjNotIntegral,j);
end if;
end for;

delta:=[RealField() | ];
for j:=1 to #JJJ-1 do
delta[j]:=0;
for i:=j+1 to #JJJ do
if delta[j] lt
```

```
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(BC)[i])^(1/2) then
delta[j]:=
DistanceToNearestInteger(sss[i][1])*Norm(Transpose(BC)[i])^(1/2);
end if;
end for;
end for;
delta[#JJJ]:=0;

max:=0;
for j in IndicesjWithsjNotIntegral do
if max lt
2^(-(#JJJ-1)/2) * DistanceToNearestInteger(sss[j][1]) *
Norm(Transpose(BC)[1])^(1/2) - (#JJJ - j)*delta[j] then
max := 2^(-(#JJJ-1)/2) * DistanceToNearestInteger(sss[j][1]) *
Norm(Transpose(BC)[1])^(1/2) - (#JJJ - j)*delta[j];
end if;
end for;
lowerbound:=max;

end if;
//END IF

if lowerbound gt (R^2 + S)^(1/2) then

//////////
/*
Choose c11 to find an optimal i0-conditional upper bound for A from
Lemma 19.2
*/
//////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
```

```
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

c11:=((1000000-1)/1000000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
max:=Max([
Floor((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) )),
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;
end for; //end i loop
////


OldUnconditionalUpperBoundForA:=UpperBoundForA;
```

```
NewConditionalUpperBoundForAi0:=MIN;

if NewConditionalUpperBoundForAi0 lt OldUnconditionalUpperBoundForA then
ConditionalUpperBoundForA[i0]:=NewConditionalUpperBoundForAi0;
flag:=false; //ready to get out of while loop that increases C if
             //necessary
else
RunThroughNumber2:=RunThroughNumber2+1; //increase C and try again
////////
/*
In case we find a new conditional upper bound that fails to improve on
the current unconditional upper bound for A, and this occurs 5 times wtih
log(C) being increased 5% each time, then we abort the real reduction
procedure.  There is no need to print a message in this case.
*/
///////
if RunThroughNumber2 eq 5 then
JumpToEndOfRealReduction:=true;
break i0;
end if;
end if; /*controlled by NewConditionalUpperBoundForAi0 lt
        OldUnconditionalUpperBoundForA*/

else //controlled by lowerbound gt (R^2 + S)^(1/2)

RunThroughNumber1:=RunThroughNumber1+1; //increase C and try again
/////
/*
If increasing log(C) 22 times (with log(C) being increased 5% the first
twenty times and by 1000% the last two times, so that log(C) will be
20 times original value) fails to produce a new conditional upper
bound for A, then we abort the real reduction procedure.  Also, print
a message to the user indicating the basic real reduction procedure
was unsuccesful for this reason.  Note: tries 21 and 22 only happen
if the upper for A has not been reduced much (or at all) from the linear
```

```
forms in logs bound.
*/
////
if RunThroughNumber1 eq 23 then
print("basic real reduction taking too long");
print("case");
iiii;
print("i0");
i0;
print("Improvement:");
Improvement;
JumpToEndOfRealReduction:=true;
break i0;
end if;

end if; //controlled by lowerbound gt (R^2 + S)^(1/2)

end while; /* this is the loop that increases log(C) if necessary, the
while loop controlled by flag */

end for; //end i0 loop


if JumpToEndOfRealReduction eq false then
//////////////
/*
Choose c11 to find an optimal i0-conditional upper bound for A for i0 in
{s+1,...,s+2t} from Lemma 19.1
*/
///////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
```

```
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
```

```
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;
end for;  //end i loop

//MIN is now an i0-conditional upper bound for A for i0 in {s+1,...,s+2t}

/////
/*
Compute the new unconditional bound for A
```

```
*/
/////

OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=Max([Max(ConditionalUpperBoundForA),MIN]);

if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
UpperBoundForA:=NewUpperBoundForA;
for i:=1 to r do
B[1+v+i]:=UpperBoundForA;
end for;
end if;

end if; //controlled by JumpToEndOfRealReduction

if UpperBoundForA lt 0 then
//there are no solutions to the current case of (11)
continue iiii;
end if;




end if; //end of IF for distinguishing (s>=3) (s=1,2) cases

end if; //end of IF for s=0, s>=3, s=1or2 cases

//done finding new upper bound for A

UpperBoundForH:=Max([UpperBoundForN, UpperBoundForA]);
//print("at end of improvement loop:");
print("Upper bounds for the n_1 and A, respectively:");
UpperBoundForn;
```

```
Floor(UpperBoundForA);

end while;  //end loop for repeated simple reductions
            //(the improvement loop)
////////////////////////////////////////////////////////////////////////
/*
End Of Loop For Repeated Simple Reductions (the improvement loop)
*/
////////////////////////////////////////////////////////////////////////

print("Starting refined reduction procedures.");

////////////////////////////////////////////////////////////////////////
/*
Initialize the set of exceptional tuples.  It will be filled as we work
through the refined redcution procedures.

At the end, it will consist of the tuples that pass all the easy tests.
The tuples will not have been tested for (11) directly and they won't all
necessarily be soltutions of (11).  We will test them more in the final
sieve step.

Each element of ExceptionalTuples will be a sequence of the form
[b_{JJJ[2]},...,b_{JJJ[#JJJ]}].
*/
////////////////////////////////////////////////////////////////////////
ExceptionalTuples:=[];

////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
//////////////////////
/*
Start Of Loop For Repeated Refined Reductions
*/
//////////////////////
```

```
//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
Improvement:=true;
while Improvement do
Improvement:=false;


////////////////////////////////////////////////////////////////////////
/*
If the number of tuples to sieve through is small enough, jump directly
to the final sieving procedure.
If the number of exceptional tuples has grown excessively large, jump to
the final sieving procedure and inform that user that this has been done.
*/
////////////////////////////////////////////////////////////////////////
prod:=1;
for i in J do
prod:=prod*(UpperBoundForn[i]+1);
end for;
prod:=prod*(2*UpperBoundForA + 1)^r;
if prod lt 5000000 then print("The number of tuples to sieve is small.
Jumping to the final sieving procedure."); break; end if;
//prod = number of tuples to sieve

if #ExceptionalTuples gt 100000 then
print("The number of exceptional tuples is large.
Jumping to the final sieving procedure.");
print("Case:");
iiii;
print("Upper bounds bor the n_i:");
UpperBoundForn;
print("Upper bound for A:");
Floor(UpperBoundForA);
break;
end if;
```

```
////////////////////////////////////////////////////////////////////////
/*
Refined p_l-adic reduction for each l in I (Section 20)
*/
////////////////////////////////////////////////////////////////////////

for l in I do


////////////////////////////////////////////////////////////////////////
/*
Define:
W[i] = W_i (weights) for p_l (Section 20)
QQ = Q (Section 20)
mmmm = m for p_l (Section 20)

Initialize:
betam[i] = beta_i^(m) for p[L],
Tm, where Transpose(Tm) = U_m from Section 20
*/
////////////////////////////////////////////////////////////////////////
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;

W[1]:=0;
for i:=2 to #JJJ do
W[JJJ[i]]:=RoundP(UpperBoundForH/B[JJJ[i]]);
end for;
for i in J do
W[1+i]:=2*W[i+1];
end for;
```

```
//print("weights");
//W;
//print("B");
//B;

QQ:=0;
for i in JJJ do
QQ:=QQ+(W[i]^2)*(B[i]^2); //recall W[1]=0
end for;

prod:=1;
for i:=2 to #JJJ do
prod:=prod*W[JJJ[i]];
end for;

/*
mmmm:=Ceiling(
(3/4)*( (#JJJ - 1)/(2*Log(p[l])) )*Log( 2^(#JJJ - 1)
* QQ / prod^(2/(#JJJ-1)) )
);
*/

mmmm:=Min([
Ceiling(
(3/4)*( (#JJJ - 1)/(2*Log(p[l])) )*Log( QQ / prod^(2/(#JJJ-1)) )
),
Ceiling((3/4)*UpperBoundForn[l])
]);


betam:=[];
for i:=1 to 1+v+r do
betam[i]:=0;
end for;
```

```
Tm:=ScalarMatrix(IntegerRing(),#JJJ-1,1);
/*initialize as #JJJ-1 by #JJJ-1 identity matrix*/

////////////////////////////////////////////////////////////////////////
/*
Start while loop that will increase m until the p_l-adic reduction yeilds
a new upper bound for n_l or until a number of increases has been made
with no success
*/
////////////////////////////////////////////////////////////////////////
flag:=true;
RunThroughNumber:=0;
while flag do

////////////////////////////////////////////////////////////////////////
/*
Increase mmmm = m (if this is not the first attempt with the basic
p_l-reduction procedure).
Calculuate betam[i] = beta_i^(m).
*/
////////////////////////////////////////////////////////////////////////
mmmm:=Ceiling( mmmm + (RunThroughNumber/20)*mmmm );
for i in JJJ do
betam[i]:= SmallestNonnegativeRemainderModpLToThem(beta[l][i],l,mmmm);
end for;
//print("mmmm");
//mmmm;

////////////////////////////////////////////////////////////////////////
/*
Compute the (soon to be) new upper bound for n_l.  If it is better than
the old upper bound, then we do the enumeration procedure.  If it is not
better, there is no need to do the enumeration and we move on to the next
l in I
*/
```

```
////////////////////////////////////////////////////////////////////////
NewUpperBoundFornl:=Max([
Floor( (1/hh[l])*(1/(p[l]-1) - Valuation(delta2[l])) ),
Ceiling((1/hh[l])*(mmmm-dd[l]))-1
]);

OldUpperBoundFornl:=UpperBoundForn[l];

if NewUpperBoundFornl ge OldUpperBoundFornl then
continue l;

else // NewUpperBoundFornl lt OldUpperBoundFornl


////////////////////////////////////////////////////////////////////////
/*
Define:
Am = A_m for p[L]
Gamma_m = lattice generated by columns of A_m
*/
////////////////////////////////////////////////////////////////////////
Am:=ZeroMatrix(IntegerRing(), v+r+2, v+r+2);
for i:=1 to v+r+1 do
Am[i][i]:=W[i];
end for;
for j:=1 to v+r+1 do
Am[v+r+2][j]:=W[ihat[l]]*betam[j];
end for;
Am[v+r+2][v+r+2]:=W[ihat[l]]*p[l]^mmmm;

for i:=1+v+r to 2 by -1 do
if i eq ihat[l] or not i in JJJ then
RemoveColumn(~Am,i);
RemoveRow(~Am,i);
end if;
```

```
end for;

RemoveColumn(~Am,1);
RemoveRow(~Am,1);

//Am is now a #JJJ-1 by #JJJ-1 matrix

Am:=Am*Transpose(Tm); //Transpose(Tm) = Um from Setion 15
/* If this is the first run through, Tm is the identity. If this is not
the first run through, this will save computation time.
*/

////////////////////////////////////////////////////////////////////////
/*
Compute an LLL reduced basis for the lattice Gamma_m generated by columns
of A_m.  The algorithm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_m.  Similarly, it spits out the transpose of B_m and
the transpose of U_m
*/
////////////////////////////////////////////////////////////////////////
//time
temp,Tm,rank:=LLL( Transpose(Am) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

Bm:=MatrixRing(RationalField(),#JJJ-1) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_m

/*
Here Tm*Transpose(Am) = Transpose(Bm).
So with Um=Transpose(Tm) we have Am*Um = Bm,
and With
Vm = Bm*Transpose(Tm)*Bm^(-1) = Am*Transpose(Tm)*Am^(-1), we have
Vm*Am = Bm.
```

```
*/

////////////////////////////////////////////////////////////////////////
/*
Define:
yyy = \vec{y} from Section 20
sss = \vec{s} from Section 20
ttt = \vec{t} from Section 20
zzz = \vec{z} from Section 20
DDD = D from Section 20
*/
////////////////////////////////////////////////////////////////////////

yyy:=ZeroMatrix(RationalField(),2+v+r,1);
for i:=1 to v do
yyy[1+i][1]:=(1/2)*W[i+1]*B[i+1];
end for;
for i:=1+v+r to 2 by -1 do
if i eq ihat[l] or not i in JJJ then
RemoveRow(~yyy,i);
end if;
end for;
RemoveRow(~yyy,1);
//yyy is now a #JJJ-1 by 1 column vector
if ihat[l] le 1+v then
yyy[#JJJ-1][1]:=-W[ihat[l]]*betam[1]+(1/2)*W[ihat[l]]*B[ihat[l]];
else //ihat[l] gt 1+v
yyy[#JJJ-1][1]:=-W[ihat[l]]*betam[1];
end if;

sss:=(Bm^(-1))*yyy;

ttt:=sss;
for i:=1 to #JJJ-1 do
ttt[i][1]:=Floor(sss[i][1]);
```

```
end for;
Floorsss:=ttt;
potentialttt:=ttt;
min:=LengthOfVector(Bm*ttt - yyy);

mm:=#JJJ-1;
Choosettt(~yyy,~ttt,~potentialttt,~Floorsss,~min,~Bm,~mm,1);

zzz:=Bm*ttt;

DDD:=0;
for i in J do
DDD:=DDD + ((1/2)*W[1+i]*B[1+i])^2;
end for;
for i:=1 to r do
DDD:=DDD + (W[1+v+i]*B[1+v+i])^2;
end for;
DDD:=DDD^(1/2);
DDD:=DDD+min;


//////////////////////////////////////////////////////////////////////
/*
Construct the lattice Gamma_m.  The MAGMA function Lattice() assumes

Use the function EnumerationCost to compute an estimate the number of
nodes in the tree to be visited during the execution of the algorithm
that will enumerate all lattice vectors u with |u| \leq D.  The number
of nodes is essentially directly proportional to the time needed for the
enumeration.

If the number of nodes is too large for the enumration to be done in a
reasonable amount of time, we increase m and try the reduction procedure
again.  If several increases of m fail to result in a small enough
estimate for the number of nodes, then we move onto the next l in I.
```

```
In Stehle and Watkins (2006), it is asserted that MAGMA's enumeration
algorithm has a traversal rate of about 7.5 millon nodes per second.
Based on the examples in the MAGMA Handbook, the rate is appears to be
20 to 40 millon nodes per second. We will assume a traversal rate of 10
millon nodes per second.  Assuming we want the enumeration to take less
than 10 minutes, we want to abort if the estimated number of nodes is
> 10*60*10^7 = 6000000000

Note: To MAGMA, Norm(v) = |v|^2.
*/
////////////////////////////////////////////////////////////////////////

Gammam:=Lattice(Transpose(Bm));


if EnumerationCost(Gammam,RealField() ! DDD^2) gt 6000000000 then

RunThroughNumber+:=1;  //increase m and try again
///
/*
If increasing m 20 times (with m being increased by 5% each time, so
that m will be double its original value after 20 increases) fails to
result in a sufficiently small estimate for the number of nodes, then we
move on to the next value of l in I.  Also, print a message indicating
that the refined p_l-adic reduction procedure was unsuccesful.
*/
/////
if RunThroughNumber eq 20 then
print("Refined p-adic reduction taking too long.");
print("Case:");
iiii;
print("l:");
l;
continue l;
```

```
end if;

else // EnumerationCost(Gammam,UpperBoundForu^2) <= 6000000000
flag:=false; //ready to get out of while loop that increases m
             //if necessary
////////////////////////////////////////////////////////////////////////
/*
Create a process P to enumerate all the vectors u in the lattice Gamma_m
with length squared |u|^2 <= D^2 (equivalently length |u| <= D).

To enumerate the vectors, we will need to repeatedly call NextVector(P).
Calling NextVector(P) will return the next vector found in the
enumeration (along with its norm).

IsEmpty(P) returns true if the process P has finished enumerating all the
vectors.  It returns false otherwise.
*/
////////////////////////////////////////////////////////////////////////
P:=ShortVectorsProcess(Gammam,Floor(DDD^2));

////////////////////////////////////////////////////////////////////////
/*
Enumerate those lattice vectors u with ||u|| <= D, extract
the corresponding tuples, and test those tuples.

Extracted tuple = (b_{JJJ[2]},...,b_{JJJ[#JJJ]})

bbb[i] = b_{JJJ[i]}, i=1 to #JJJ

Recall: For each ll in I, jl[ll] is the unique index such that
JJJ[jl[ll]]=ll+1.
So bbb[jl[ll]] = b_{ll+1} = n_ll
*/
////////////////////////////////////////////////////////////////////////
bbb:=[RationalField() | ];
```

```
while not IsEmpty(P) do

uuu:=NextVector(P);

//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]})
bbb[1]:=1;
if ihat[l] le 1+v then

for i:=2 to istar[l]-1 do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
bbb[istar[l]]:=(uuu[#JJJ-1]-(yyy[#JJJ-1][1]-zzz[#JJJ-1][1])
+ (1/2)*W[JJJ[istar[l]]]*B[JJJ[istar[l]]])/W[JJJ[istar[l]]];
for i:=istar[l]+1 to 1+#J do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1]))/W[JJJ[i]];
end for;

else //ihat[l] gt 1+v

for i:=2 to 1+#J do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to istar[l]-1 do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
```

```
bbb[istar[l]]:=(uuu[#JJJ-1]-(yyy[#JJJ-1][1]-zzz[#JJJ-1][1]))
/W[JJJ[istar[l]]];
for i:=istar[l]+1 to #JJJ do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1]))/W[JJJ[i]];
end for;

end if;

/*
if bbb[1] ne 1 then
print("something is wrong!"); bbb[1]; istar[l]; break iiii; end if;
*/

//test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2+#J to #JJJ do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then
passes:=false; break; end if;
end for;
if passes eq false then break; end if;

/*test if tuple fits in the new box.  If so, throw it away (it's not
```

```
exceptional).  Recall bbb[jl[l]] = b_{l+1} = n_l*/
if bbb[jl[l]] le NewUpperBoundFornl then passes:=false; break; end if;


LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[l][JJJ[i]];
end for;
if SpecialCase[l] eq true then
if Valuation(LAMBDAprime) ne bbb[jl[l]]*hh[l] + dd[l] then
passes:=false; break; end if;
else //SpecialCase[l] eq false
if Valuation(LAMBDAprime) lt bbb[jl[l]]*hh[l] + dd[l] then
passes:=false; break; end if;
end if;


for ll in I do
if ll ne l then

LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;
```

```
end if;
end for;
if passes eq false then break; end if;


if SpecialCase[l] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*LogarithmicAlphap[l][JJJ[i]];
end for;
if Valuation(LAMBDA) ne bbb[jl[l]]*hh[l] + Valuation(delta2[l]) then
passes:=false; break; end if;
end if; //end IF controlled by SpecialCase[l]


for ll in I do
if ll ne l then
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
end if; //end IF controlled by SpecialCase[ll]
end if;
end for;
if passes eq false then break; end if;


break;
end while; //end hack while loop
```

```
//If the tuple passes all the tests, add it to the list of exceptional
//tuples
if passes eq true then Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;

//Since the enumeration process only spits out one of u and -u, we now
//need to do the same thing with -u.

uuu:=-uuu;

//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]})
bbb[1]:=1;

if ihat[l] le 1+v then

for i:=2 to istar[l]-1 do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
bbb[istar[l]]:=(uuu[#JJJ-1]-(yyy[#JJJ-1][1]-zzz[#JJJ-1][1])
+ (1/2)*W[JJJ[istar[l]]]*B[JJJ[istar[l]]])/W[JJJ[istar[l]]];
for i:=istar[l]+1 to 1+#J do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1]) + (1/2)*W[JJJ[i]]*B[JJJ[i]])
/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1]))/W[JJJ[i]];
end for;

else //ihat[l] gt 1+v

for i:=2 to 1+#J do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
```

```
end for;
for i:=2+#J to istar[l]-1 do
bbb[i]:=(uuu[i-1]-(yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
bbb[istar[l]]:=(uuu[#JJJ-1]-(yyy[#JJJ-1][1]-zzz[#JJJ-1][1]))
/W[JJJ[istar[l]]];
for i:=istar[l]+1 to #JJJ do
bbb[i]:=(uuu[i-2]-(yyy[i-2][1]-zzz[i-2][1]))/W[JJJ[i]];
end for;

end if;

/*
if bbb[1] ne 1 then print("something is wrong!"); end if;
*/

//test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
if passes eq false then break; end if;



for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;



for i:=2+#J to #JJJ do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then
```

```
passes:=false; break; end if;
end for;
if passes eq false then break; end if;


/*test if tuple fits in the new box.  If so, throw it away (it's not
exceptional).  Recall bbb[jl[l]] = b_{l+1} = n_l*/
if bbb[jl[l]] le NewUpperBoundFornl then passes:=false; break; end if;


LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[l][JJJ[i]];
end for;


if SpecialCase[l] eq true then
if Valuation(LAMBDAprime) ne bbb[jl[l]]*hh[l] + dd[l] then
passes:=false; break; end if;
else //SpecialCase[l] eq false
if Valuation(LAMBDAprime) lt bbb[jl[l]]*hh[l] + dd[l] then
passes:=false; break; end if;
end if;


for ll in I do
if ll ne l then

LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll])) and
Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
```

```
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll])) and
Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;

end if;
end for;
if passes eq false then break; end if;


if SpecialCase[l] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*LogarithmicAlphap[l][JJJ[i]];
end for;
if Valuation(LAMBDA) ne bbb[jl[l]]*hh[l] + Valuation(delta2[l]) then
passes:=false; break; end if;
end if; //end IF controlled by SpecialCase[l]


for ll in I do
if ll ne l then
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll])) and
Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
```

```
end if; //end IF controlled by SpecialCase[ll]
end if;
end for;
if passes eq false then break; end if;

break;
end while; //end hack while loop

//If the tuple passes all the tests, add it to the list of exceptional
//tuples
if passes eq true then Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;



end while; //enumeration loop

//Enumeration done



Improvement:=true;
UpperBoundForn[l]:=NewUpperBoundFornl;
B[1+l]:=UpperBoundForn[l];

end if; // end of IF controlled by EnumerationCost(Gammam,D^2)



end if; // end of IF controlled by NewUpperBoundFornl ge
        // OldUpperBoundFornl



end while; //this is the loop that increases m if needed

end for; // end l loop

UpperBoundForN:=Max(UpperBoundForn);
```

```
////////////////////////////////////////////////////////////////////////
/*
Refined Real Reduction (Section 21)
*/
////////////////////////////////////////////////////////////////////////
for i0:=1 to s do
ConditionalUpperBoundForA[i0]:=UpperBoundForA;
end for;

////////////////////////////////////////////////////////////////////////
/*
Totally Complex Case, i.e, s=0
*/
////////////////////////////////////////////////////////////////////////
if s eq 0 then

//Choose c11 to find an choose optimal upper bound for A from Lemma 19.1
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then
min:=Abs(Imaginary(thetaC[j])); end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
```

```
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);

c15:=0;
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
```

```
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;
end for;  //end i loop

OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=MIN;

if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
UpperBoundForA:=NewUpperBoundForA;
end if;

else //s > 0
////////////////////////////////////////////////////////////////////////
/*
s>0 Case
*/
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/*
Real Case, i.e., s \geq 3
*/
////////////////////////////////////////////////////////////////////////
if s ge 3 then // s>=3 real case
```

```
JumpToEndOfRealReduction:=false;
for i0:=1 to s do
//print("refined real, i0=");
//i0;

////////////////////////////////////////////////////////////////////////
/*
Define
weights W[i] (Section 21)
R (approximation) (Section 21)
S (Section 21)
CCCCCC = C (Section 21)

Initialize:
TC, where Transpose(TC) = U_C from Section 21
phi[i]:= phi_i from Section 21
*/
////////////////////////////////////////////////////////////////////////
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;

H0prime:=0;
for i:=2 to #JJJ-1 do
if H0prime lt B[JJJ[i]] then H0prime:=B[JJJ[i]]; end if;
end for;

W[1]:=0;
for i:=2 to #JJJ-1 do
W[JJJ[i]]:=RoundP(H0prime/B[JJJ[i]]);
end for;
for i in J do
W[1+i]:=2*W[i+1];
```

```
end for;
//print("weights");
//W;
//print("B");
//B;

R:=0;
for i in JJJ do
R:=R + B[i];
end for;
R:=(1/2)*R;
//this is just an approximation to R

S:=0;
for i:=2 to #JJJ-1 do
S:=S+(W[JJJ[i]]^2)*(B[JJJ[i]]^2);
end for;

prod:=1;
for i:=2 to #JJJ-1 do
prod:=prod*W[JJJ[i]];
end for;

/*
LogC:= (3/4) * ((#JJJ-1)/2) * Log(
(R^2 + S) / ( 2^(-(#JJJ-1))
* (Abs(LogarithmicAlphaC[i0][JJJ[#JJJ]])*prod)^(2/(#JJJ-1)) )
);
*/

LogC:= Min([
(3/4) * ((#JJJ-1)/2) * Log(
(R^2 + S)
/ (  (Abs(LogarithmicAlphaC[i0][JJJ[#JJJ]])*prod)^(2/(#JJJ-1)) )
),
```

```
(3/4)*UpperBoundForA
]);

//CCCCCC:=Ceiling(Exp(LogC));

TC:=ScalarMatrix(IntegerRing(),#JJJ-1,1);
//#JJJ-1 by #JJJ-1 identity matrix

phi:=[];
for i:=1 to 1+v+r do
phi[i]:=0;
end for;

////////////////////////////////////////////////////////////////////////
/*
Start the while loop where C will be increased until we find a new
conditional upper for A that is smaller than the uncondtional upper
bound for A or until a number of increases of C have been made with no
success.
*/
////////////////////////////////////////////////////////////////////////
RunThroughNumber1:=0;
RunThroughNumber2:=0;
flag:=true;
while flag do



////////////////////////////////////////////////////////////////////////
/*
Increase CCCCCC = C (if this is not the first attempt the basic real
reduction procedure).
*/
////////////////////////////////////////////////////////////////////////
CCCCCC:=
```

```
Ceiling(Exp(LogC + ((RunThroughNumber1 + RunThroughNumber2)/20)*LogC));
//print("CCCCCC");
//CCCCCC;

////////////////////////////////////////////////////////////////////////
/*
Compute the (soon to be) new i0-conditional upper bound for A. We choose
c11 to optimize this bound.

If it is better than the old unconditional upper bound, then we do the
enumeration procedure.  If it is not better, then we first retry several
times with larger values of C, but if that doesn't work we abandon the
refined real reduction procedure (there is no point in moving on to the
next i0 in {1,...,s} because the uncondtional upper bound on A will not
be improved).
*/
////////////////////////////////////////////////////////////////////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

c11:=((1000000-1)/1000000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
```

```
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(2*Log(2)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;
end for; //end i loop

NewConditionalUpperBoundForAi0:=MIN;
OldUnconditionalUpperBoundForA:=UpperBoundForA;

if NewConditionalUpperBoundForAi0 ge OldUnconditionalUpperBoundForA then
RunThroughNumber2:=RunThroughNumber2+1; //increase C and try again
////////
/*
In case we find a new conditional upper bound that fails to improve on
the current unconditional upper bound for A, and this occurs 5 times with
log(C) being increased 5% each time, then we abort the real reduction
procedure.  There is no need to print a message in this case.
*/
///////
if RunThroughNumber2 eq 5 then
JumpToEndOfRealReduction:=true;
print("RunThroughNumber2 = ");
```

```
RunThroughNumber2;
break i0;
end if;

else //NewConditionalUpperBoundForAi0 lt OldUnconditionalUpperBoundForA

////////////////////////////////////////////////////////////////////////
/*
Calculate the phi_i for i in JJJ.

Calculate R.
*/
////////////////////////////////////////////////////////////////////////
for i in JJJ do
phi[i]:=Round(CCCCCC*LogarithmicAlphaC[i0][i]);
end for;

if Abs(phi[JJJ[#JJJ]]) lt 2 then
if CCCCCC*LogarithmicAlphaC[i0][JJJ[#JJJ]] ge 0 then
phi[JJJ[#JJJ]]:=2;
else
phi[JJJ[#JJJ]]:=-2;
end if;
end if;

if IsIntegral(phi[1]/phi[JJJ[#JJJ]]) then
if Abs(phi[1]+1 - CCCCCC*LogarithmicAlphaC[i0][1]) le 1 then
phi[1]:=phi[1]+1;
else
phi[1]:=phi[1]-1;
end if;
end if;

R:=0;
for i in JJJ do
```

```
R:=R + B[i]*Abs( CCCCCC*LogarithmicAlphaC[i0][i] -  phi[i] );
end for;

////////////////////////////////////////////////////////////////////////
/*
Compute
AC = A_C
*/
////////////////////////////////////////////////////////////////////////
AC:=ZeroMatrix(IntegerRing(),v+r+1,v+r+1);

for i:=1 to v+r do
AC[i][i]:=W[i];
end for;
for j:=1 to v+r+1 do
AC[v+r+1][j]:=phi[j];
end for;

for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveColumn(~AC,i);
RemoveRow(~AC,i);
end if;
end for;

RemoveRow(~AC,1);
RemoveColumn(~AC,1);

//AC is now a #JJJ-1 by #JJJ-1 matrix

AC:=AC*Transpose(TC);
/* If this is the first run through Transpose(TC) is the identity.
Otherwise Transpose(TC) is UC.  This will save time in the LLL
reduction. */
```

```
////////////////////////////////////////////////////////////////////////
/*
Compute an LLL reduced basis for the lattice Gamma_C generated by columns
of A_C.  The algoritm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_C.  Similarly, it spits out the transpose of B_C and
the transpose of U_C
*/
////////////////////////////////////////////////////////////////////////
//time
temp,TC,rank:=LLL(Transpose(AC) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

BC:=MatrixRing(RationalField(),#JJJ-1) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_C

/*
Here TC*Transpose(AC) = Transpose(BC).
So with UC=Transpose(TC) we have AC*UC = BC,
and With
VC = BC*Transpose(TC)*BC^(-1) = AC*Transpose(TC)*AC^(-1), we have
VC*AC = BC.
*/

////////////////////////////////////////////////////////////////////////
/*
Define:
yyy = \vec{y} from Section 21
sss = \vec{s} from Section 21
ttt = \vec{t} from Section 21
zzz = \vec{z} from Section 21
DDD = D from Section 21
*/
////////////////////////////////////////////////////////////////////////
```

```
yyy:=ZeroMatrix(RationalField(),1+v+r,1);
for i:=1 to v do
yyy[1+i][1]:=(1/2)*W[1+i]*B[1+i];
end for;
for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveRow(~yyy,i);
end if;
end for;
RemoveRow(~yyy,1);
//yyy is now a #JJJ-1 by 1 column vector
yyy[#JJJ-1][1]:=-phi[1];

sss:=(BC^(-1))*yyy;

ttt:=sss;
for i:=1 to #JJJ-1 do
ttt[i][1]:=Floor(sss[i][1]);
end for;
Floorsss:=ttt;
potentialttt:=ttt;
min:=LengthOfVector(BC*ttt - yyy);

mm:=#JJJ-1;
Choosettt(~yyy,~ttt,~potentialttt,~Floorsss,~min,~BC,~mm,1);

zzz:=BC*ttt;

/*Now zzz is likely the closest vector in the lattice Gamma_C to yyy and
min = |zzz-yyy|*/

DDD:=((R+1)^2 + S)^(1/2);

////////////////////////////////////////////////////////////////////////
/*
```

```
Construct the lattice Gamma_C.  The MAGMA function Lattice() assumes

Use the function EnumerationCost to compute an estimate the number of
nodes in the tree to be visited during the execution of the algorithm
that will enumerate all lattice vectors u with |u| \leq D + |y-z|.  The
number of nodes is essentially directly proportional to the time needed
for the enumeration.

If the number of nodes is too large for the enumration to be done in a
reasonable amount of time, we increase C and try the reduction procedure
again.  If several increases of C fail to result in a small enough
estimate for the number of nodes, then we abort the refined real
reduction (there is no point in moving on to the next i0 in {1,...,s}
because the uncondtional upper bound on A will not be improved).

In Stehle and Watkins (2006), it is asserted that MAGMA's enumeration
algorithm has a traversal rate of about 7.5 millon nodes per second.
Based on the examples in the MAGMA Handbook, the rate is appears to be
20 to 40 millon nodes per second. We will assume a traversal rate of 10
millon nodes per second.  Assuming we want the enumeration to take less
than 10 minutes, we want to abort if the estimated number of nodes
is > 10*60*10^7 = 6000000000

Note: To MAGMA, Norm(v) = |v|^2.
*/
////////////////////////////////////////////////////////////////////////

GammaC:=Lattice(Transpose(BC));

if EnumerationCost(GammaC,RealField() ! (DDD+min)^2) gt 6000000000 then

RunThroughNumber1+:=1;  //increase C and try again
/////
/*
If increasing log(C) 20 times (with log(C) being increased 5% each time,
```

```
so that log(C) will be double its original value) fails to produce a new
conditional upper bound for A, then we abort the real reduction procedure.
Also, print a message to the user indicating the refined real reduction
procedure was unsuccesful for this reason.
*/
////
if RunThroughNumber1 eq 20 then
print("Refined real reduction taking too long.");
print("Case:");
iiii;
print("i0:");
i0;
JumpToEndOfRealReduction:=true;
break i0;
end if;

else // EnumerationCost(GammaC,(DDD+min)^2) <= 6000000000
flag:=false; //ready to get out of while loop that increases C
             //if necessary
//////////////////////////////////////////////////////////////////////
/*
Create a process P to enumerate all the vectors u in the lattice Gamma_C
with lenght squared |u|^2 <= (D+|y-z|)^2 (equivalently length
|u| <= D+|y-z|).

To enumerate the vectors, we will need to repeatedly call NextVector(P).
Callign NextVector(P) will return the next vector found in the enumeration
(along with its norm).

IsEmpty(P) returns true if the process P has finished enumerating all the
vectors.  It returns false otherwise.
*/
//////////////////////////////////////////////////////////////////////
P:=ShortVectorsProcess(GammaC,Floor((DDD+min)^2));
```

```
////////////////////////////////////////////////////////////////////////
/*
Enumerate those lattice vectors u with |u| <= D+|y-z|, extract
the corresponding tuples, and test those tuples.

Extracted tuple = (b_{JJJ[2]},...,b_{JJJ[#JJJ]})

bbb[i] = b_{JJJ[i]}, i=1 to #JJJ
*/
////////////////////////////////////////////////////////////////////////
bbb:=[RationalField() | ];



while not IsEmpty(P) do

uuu:=NextVector(P);

//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]})
bbb[1]:=1;

for i:=2 to 1+#J do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ-1 do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
sum:=0;
for i:=1 to #JJJ-1 do
sum:=sum+bbb[i]*phi[JJJ[i]];
end for;
bbb[#JJJ] := (uuu[#JJJ-1] - (yyy[#JJJ-1][1]-zzz[#JJJ-1][1]) - sum)
/phi[JJJ[#JJJ]];
```

```
//Test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2+#J to #JJJ do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then
passes:=false; break; end if;
end for;
if passes eq false then break; end if;

/*Test if the tuple fits in the new box.  If so, throw it away (it's not
exceptional). */
FitsInTheNewBox:=true;
for i:=2+#J to #JJJ do
if Abs(bbb[i]) gt NewConditionalUpperBoundForAi0 then
FitsInTheNewBox:=false; break i;
end if;
end for;
if FitsInTheNewBox eq true then passes:=false; break; end if;

for ll in I do
LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
```

```
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;

for ll in I do
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;


break;
end while; //end hack while loop

/*If the tuple passes all the tests, add it to the list of exceptional
tuples*/
```

```
if passes eq true then Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;

//

/*Since the enumeration process only spits out one of u and -u, we now
need to do the same thing with -u.*/

uuu:=-uuu;

//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]})
bbb[1]:=1;

for i:=2 to 1+#J do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ-1 do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
sum:=0;
for i:=1 to #JJJ-1 do
sum:=sum+bbb[i]*phi[JJJ[i]];
end for;
bbb[#JJJ] := (uuu[#JJJ-1] - (yyy[#JJJ-1][1]-zzz[#JJJ-1][1]) - sum)
/phi[JJJ[#JJJ]];

//Test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
```

```
if passes eq false then break; end if;

for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2+#J to #JJJ do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then
passes:=false; break; end if;
end for;
if passes eq false then break; end if;

/*Test if the tuple fits in the new box.  If so, throw it away (it's not
exceptional). */
FitsInTheNewBox:=true;
for i:=2+#J to #JJJ do
if Abs(bbb[i]) gt NewConditionalUpperBoundForAi0 then
FitsInTheNewBox:=false; break i;
end if;
end for;
if FitsInTheNewBox eq true then passes:=false; break; end if;

for ll in I do
LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
```

```
and Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;

for ll in I do
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;


break;
end while; //end hack while loop

/*If the tuple passes all the tests, add it to the list of exceptional
tuples*/
if passes eq true then Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;


end while; //enumeration loop

//Enumeration done
```

```
ConditionalUpperBoundForA[i0]:=NewConditionalUpperBoundForAi0;


end if; //end of IF controlled by EnumerationCost(GammaC,(DDD+min)^2)

end if; /* end of IF controlled by NewConditionalUpperBoundForAi0 ge
OldUnconditionalUpperBoundForA */

end while; /* this is the while loop that increases log(C) if necessary,
              the while loop controlled by flag */


end for; //end i0 loop


if JumpToEndOfRealReduction eq false then
//////////////
/*
Choose c11 to find an optimal i0-conditional upper bound for A for i0 in
{s+1,...,s+2t} from Lemma 19.1
*/
///////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
```

```
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
```

```
c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;
end for;  //end i loop

/////
/*
Compute the new unconditional bound for A
*/
/////


OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=Max([Max(ConditionalUpperBoundForA),MIN]);


if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
```

```
UpperBoundForA:=NewUpperBoundForA;
for i:=1 to r do
B[1+v+i]:=UpperBoundForA;
end for;
end if;

end if; //controlled by JumpToEndOfRealReduction

if UpperBoundForA lt 0 then
//there are no solutions to the current case of (11)
continue iiii;
end if;




////////////////////////////////////////////////////////////////////////
/*
Complex Case, i.e., s = 1,2
*/
////////////////////////////////////////////////////////////////////////
else //s=1,2 complex case

B[2+v+r]:=Floor(2*Arcsin(1/4)/PI);
for i in JJJ do
B[2+v+r]:=B[2+v+r] + B[i];
end for;


JumpToEndOfRealReduction:=false;
for i0:=1 to s do


////////////////////////////////////////////////////////////////////////
```

```
/*
Define
weights W[i] (Section 21)
R (approximation) (Section 21)
S (Section 21)
CCCCCC = C (Section 21)

Initialize:
TC, where Transpose(TC) = U_C from Section 21
phi[i]:= phi_i from Section 21
*/
////////////////////////////////////////////////////////////////////////
W:=[];
for i:=1 to 1+v+r do
W[i]:=0; //initialize
end for;

H0prime:=0;
for i:=2 to #JJJ do
if H0prime lt B[JJJ[i]] then H0prime:=B[JJJ[i]]; end if;
end for;

W[1]:=0;
for i:=2 to #JJJ do
W[JJJ[i]]:=RoundP(H0prime/B[JJJ[i]]);
end for;
for i in J do
W[1+i]:=2*W[i+1];
end for;

R:=B[2+v+r];
for i in JJJ do
R:=R + B[i];
end for;
R:=(1/2)*R;
```

```
//this is just an approximation to R

S:=0;
for i:=2 to #JJJ do
S:=S+(W[JJJ[i]]^2)*(B[JJJ[i]]^2);
end for;

prod:=1;
for i:=2 to #JJJ do
prod:=prod*W[JJJ[i]];
end for;

/*
LogC:= (3/4) * ((#JJJ)/2) * Log(
(R^2 + S)
/ ( 2^(-(#JJJ)) * (Abs(LogarithmicAlphaC[i0][2+v+r])*prod)^(2/(#JJJ)) )
);
*/

LogC:= Min([
(3/4) * ((#JJJ)/2) * Log(
(R^2 + S) / ( (Abs(LogarithmicAlphaC[i0][2+v+r])*prod)^(2/(#JJJ)) )
),
(3/4)*UpperBoundForA
]);

//CCCCCC:=Ceiling(Exp(LogC));

TC:=ScalarMatrix(IntegerRing(),#JJJ,1); //#JJJ by #JJJ identity matrix

phi:=[];
for i:=1 to 2+v+r do
phi[i]:=0;
end for;
```

```
////////////////////////////////////////////////////////////////////////
/*
Start the while loop where C will be increased until we find a new
conditional upper for A that is smaller than the uncondtional upper
bound for A or until a number of increases of C have been made with no
success.
*/
////////////////////////////////////////////////////////////////////////
RunThroughNumber1:=0;
RunThroughNumber2:=0;
flag:=true;
while flag do

////////////////////////////////////////////////////////////////////////
/*
Increase CCCCCC = C (if this is not the first attempt the basic real
reduction procedure).
*/
////////////////////////////////////////////////////////////////////////
CCCCCC:=
Ceiling(Exp(LogC + ((RunThroughNumber1 + RunThroughNumber2)/20)*LogC));

////////////////////////////////////////////////////////////////////////
/*
Compute the (soon to be) new i0-conditional upper bound for A. We choose
c11 to optimize this bound.

If it is better than the old unconditional upper bound, then we do the
enumeration procedure.  If it is not better, then we first retry several
times with larger values of C, but if that doesn't work we abandon the
refined real reduction procedure (there is no point in moving on to the
next i0 in {1,...,s} because the uncondtional upper bound on A will not
be improved).
*/
////////////////////////////////////////////////////////////////////////
```

```
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

c11:=((1000000-1)/1000000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);
max:=Max([
Ceiling((1/c11)*( Log(4*Arcsin(1/4)*c16[i0]) + Log(CCCCCC)
- Log((lowerbound^2 - S)^(1/2) - R) ))-1,
Ceiling((c8prime + c9prime*UpperBoundForN)/(c10 - (n-1)*c11))-1,
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10 - c11))-1,
Ceiling(Log(2*c16[i0])/c11)-1,
0
]);
if max lt MIN then MIN:=max; end if;
```

```
end for; //end i loop

NewConditionalUpperBoundForAi0:=MIN;
OldUnconditionalUpperBoundForA:=UpperBoundForA;

if NewConditionalUpperBoundForAi0 ge OldUnconditionalUpperBoundForA then
RunThroughNumber2:=RunThroughNumber2+1; //increase C and try again
////////
/*
In case we find a new conditional upper bound that fails to improve on
the current unconditional upper bound for A, and this occurs 5 times with
log(C) being increased 5% each time, then we abort the real reduction
procedure.  There is no need to print a message in this case.
*/
///////
if RunThroughNumber2 eq 5 then
JumpToEndOfRealReduction:=true;
break i0;
end if;

else //NewConditionalUpperBoundForAi0 lt OldUnconditionalUpperBoundForA

////////////////////////////////////////////////////////////////////////
/*
Calculate the phi_i for i in JJJ.

Calculate R.
*/
////////////////////////////////////////////////////////////////////////
for i in JJJ do
phi[i]:=Round(CCCCCC*LogarithmicAlphaC[i0][i]);
end for;
phi[2+v+r]:=Round(CCCCCC*LogarithmicAlphaC[i0][2+v+r]);

/*This will never happen because LogarithmicAlphaC[i0][2+v+r] = Pi */
```

```
if Abs(phi[2+v+r]) lt 2 then
if CCCCCC*LogarithmicAlphaC[i0][2+v+r] ge 0 then
phi[#JJJ]:=2;
else
phi[#JJJ]:=-2;
end if;
end if;

if IsIntegral(phi[1]/phi[2+v+r]) then
if Abs(phi[1]+1 - CCCCCC*LogarithmicAlphaC[i0][1]) le 1 then
phi[1]:=phi[1]+1;
else
phi[1]:=phi[1]-1;
end if;
end if;

R:=B[2+v+r]*Abs( CCCCCC*LogarithmicAlphaC[i0][2+v+r] -  phi[2+v+r] );
for i in JJJ do
R:=R + B[i]*Abs( CCCCCC*LogarithmicAlphaC[i0][i] -  phi[i] );
end for;

////////////////////////////////////////////////////////////////////////
/*
Compute
AC = A_C
*/
////////////////////////////////////////////////////////////////////////
AC:=ZeroMatrix(IntegerRing(),v+r+2,v+r+2);

for i:=1 to 1+v+r do
AC[i][i]:=W[i];
end for;
for j:=1 to v+r+2 do
AC[v+r+2][j]:=phi[j];
end for;
```

```
for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveColumn(~AC,i);
RemoveRow(~AC,i);
end if;
end for;

RemoveRow(~AC,1);
RemoveColumn(~AC,1);

//AC is now a #JJJ by #JJJ matrix

AC:=AC*Transpose(TC);
/* If this is the first run through Transpose(TC) is the identity.
Otherwise Transpose(TC) is UC.  This will time in the LLL reduction. */

////////////////////////////////////////////////////////////////////////
/*
Compute an LLL reduced basis for the lattice Gamma_C generated by columns
of A_C.  The algoritm used is de Weger's exact integer version of LLL.

Note: LLL(X) assumes ROWS of X span the lattice, so we need to feed it
the transpose of A_C.  Similarly, it spits out the transpose of B_C and
the transpose of U_C
*/
////////////////////////////////////////////////////////////////////////
//time
temp,TC,rank:=LLL(Transpose(AC) : Proof:=true, Method:="Integral",
Delta:=0.75, Eta:=0.5 );

BC:=MatrixRing(RationalField(),#JJJ) ! Transpose(temp);
//columns are LLL-reduced basis for Gamma_C

/*
```

```
Here TC*Transpose(AC) = Transpose(BC).
So with UC=Transpose(TC) we have AC*UC = BC,
and With
VC = BC*Transpose(TC)*BC^(-1) = AC*Transpose(TC)*AC^(-1), we have
VC*AC = BC.
*/

////////////////////////////////////////////////////////////////////////
/*
Define:
yyy = \vec{y} from Section 21
sss = \vec{s} from Section 21
ttt = \vec{t} from Section 21
zzz = \vec{z} from Section 21
DDD = D from Section 21
*/
////////////////////////////////////////////////////////////////////////
yyy:=ZeroMatrix(RationalField(),2+v+r,1);
for i:=1 to v do
yyy[1+i][1]:=(1/2)*W[1+i]*B[1+i];
end for;
for i:=1+v+r to 2 by -1 do
if not i in JJJ then
RemoveRow(~yyy,i);
end if;
end for;
RemoveRow(~yyy,1);
//yyy is now a #JJJ by 1 column vector
yyy[#JJJ][1]:=-phi[1];

sss:=(BC^(-1))*yyy;

ttt:=sss;
for i:=1 to #JJJ do
ttt[i][1]:=Floor(sss[i][1]);
```

```
end for;
Floorsss:=ttt;
potentialttt:=ttt;
min:=LengthOfVector(BC*ttt - yyy);

mm:=#JJJ;
Choosettt(~yyy,~ttt,~potentialttt,~Floorsss,~min,~BC,~mm,1);

zzz:=BC*ttt;

/*Now zzz is likely the closest vector in the lattice Gamma_C to yyy and
min = |zzz-yyy|*/

DDD:=((R+1)^2 + S)^(1/2);

////////////////////////////////////////////////////////////////////////
/*
Construct the lattice Gamma_C.  The MAGMA function Lattice() assumes

Use the function EnumerationCost to compute an estimate the number of
nodes in the tree to be visited during the execution of the algorithm
that will enumerate all lattice vectors u with |u| \leq D + |y-z|.  The
number of nodes is essentially directly proportional to the time needed
for the enumeration.

If the number of nodes is too large for the enumration to be done in a
reasonable amount of time, we increase C and try the reduction procedure
again.  If several increases of C fail to result in a small enough
estimate for the number of nodes, then we abort the refined real
reduction (there is no point in moving on to the next i0 in {1,...,s}
because the uncondtional upper bound on A will not be improved).

In Stehle and Watkins (2006), it is asserted that MAGMA's enumeration
algorithm has a traversal rate of about 7.5 millon nodes per second.
Based on the examples in the MAGMA Handbook, the rate is appears to be
```

```
20 to 40 millon nodes per second. We will assume a traversal rate of 10
millon nodes per second.  Assuming we want the enumeration to take less
than 10 minutes, we want to abort if the estimated number of nodes is
> 10*60*10^7 = 6000000000

Note: To MAGMA, Norm(v) = |v|^2.
*/
////////////////////////////////////////////////////////////////////////

GammaC:=Lattice(Transpose(BC));

if EnumerationCost(GammaC,RealField() ! (DDD+min)^2) gt 6000000000 then

RunThroughNumber1+:=1;  //increase C and try again
/////
/*
If increasing log(C) 20 times (with log(C) being increased 5% each time,
so that log(C) will be double its original value) fails to produce a new
conditional upper bound for A, then we abort the real reduction procedure.
Also, print a message to the user indicating the refined real reduction
procedure was unsuccesful for this reason.
*/
////
if RunThroughNumber1 eq 20 then
print("Refined real reduction taking too long.");
print("Case:");
iiii;
print("i0:");
i0;
JumpToEndOfRealReduction:=true;
break i0;
end if;

else // EnumerationCost(GammaC,(DDD + min)^2) <= 6000000000
flag:=false; //ready to get out of while loop that increases C
```

```
            //if necessary
////////////////////////////////////////////////////////////////////////
/*
Create a process P to enumerate all the vectors u in the lattice Gamma_C
with lenght squared |u|^2 <= (D+|y-z|)^2 (equivalently length |u| <=
D+|y-z|).

To enumerate the vectors, we will need to repeatedly call NextVector(P).
Callign NextVector(P) will return the next vector found in the
enumeration (along with its norm).

IsEmpty(P) returns true if the process P has finished enumerating all the
vectors.  It returns false otherwise.
*/
////////////////////////////////////////////////////////////////////////
P:=ShortVectorsProcess(GammaC,Floor((DDD+min)^2));

////////////////////////////////////////////////////////////////////////
/*
Enumerate those lattice vectors u with |u| <= D+|y-z|, extract
the corresponding tuples, and test those tuples.

Extracted tuple = (b_{JJJ[2]},...,b_{JJJ[#JJJ]},b_{2+v+r})

bbb[i] = b_{JJJ[i]}, i=1 to #JJJ
bbb[#JJJ+1] = b_{2+v+r}
*/
////////////////////////////////////////////////////////////////////////
bbb:=[RationalField() | ];
bbb[1]:=1;

while not IsEmpty(P) do

uuu:=NextVector(P);
```

```
//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]},b_{2+v+r})

for i:=2 to 1+#J do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
sum:=0;
for i:=1 to #JJJ do
sum:=sum+bbb[i]*phi[JJJ[i]];
end for;
bbb[#JJJ+1] := (uuu[#JJJ] - (yyy[#JJJ][1]-zzz[#JJJ][1]) - sum)
/phi[2+v+r];

//Test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

if not IsIntegral((1/2)*bbb[#JJJ+1]) then passes:=false; break; end if;


for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2+#J to #JJJ do
```

```
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then
passes:=false; break; end if;
end for;
if passes eq false then break; end if;

if bbb[#JJJ+1] gt B[2+v+r] or bbb[#JJJ+1] lt -B[2+v+r] then
passes:=false; break; end if;

/*Test if the tuple fits in the new box.  If so, throw it away (it's not
exceptional). */
FitsInTheNewBox:=true;
for i:=2+#J to #JJJ do
if Abs(bbb[i]) gt NewConditionalUpperBoundForAi0 then
FitsInTheNewBox:=false; break i;
end if;
end for;
if FitsInTheNewBox eq true then passes:=false; break; end if;

for ll in I do
LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;
end for;
```

```
if passes eq false then break; end if;

for ll in I do
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;


break;
end while; //end hack while loop

/*If the tuple passes all the tests, add it to the list of exceptional
tuples*/
if passes eq true then
Remove(~bbb,#JJJ+1); Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;

//

/*Since the enumeration process only spits out one of u and -u, we now
need to do the same thing with -u.*/

uuu:=-uuu;

//Extract tuple (b_{JJJ[2]},...,b_{JJJ[#JJJ]},b_{2+v+r})
```

```
for i:=2 to 1+#J do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1])
+ (1/2)*W[JJJ[i]]*B[JJJ[i]])/W[JJJ[i]];
end for;
for i:=2+#J to #JJJ do
bbb[i] := (uuu[i-1] - (yyy[i-1][1]-zzz[i-1][1]))/W[JJJ[i]];
end for;
sum:=0;
for i:=1 to #JJJ do
sum:=sum+bbb[i]*phi[JJJ[i]];
end for;
bbb[#JJJ+1] := (uuu[#JJJ] - (yyy[#JJJ][1]-zzz[#JJJ][1]) - sum)
/phi[2+v+r];

//Test the tuple
passes:=true;
while true do //this while loop is a hack to provide a way to
              //"jump to line X"

for i:=2 to #JJJ do
if not IsIntegral(bbb[i]) then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

if not IsIntegral((1/2)*bbb[#JJJ+1]) then passes:=false; break; end if;



for i:=2 to 1+#J do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt 0 then passes:=false; break; end if;
end for;
if passes eq false then break; end if;

for i:=2+#J to #JJJ do
if bbb[i] gt B[JJJ[i]] or bbb[i] lt -B[JJJ[i]] then passes:=false; break;
end if;
```

```
end for;
if passes eq false then break; end if;

if bbb[#JJJ+1] gt B[2+v+r] or bbb[#JJJ+1] lt -B[2+v+r] then
passes:=false; break; end if;

/*Test if the tuple fits in the new box.  If so, throw it away (it's not
exceptional). */
FitsInTheNewBox:=true;
for i:=2+#J to #JJJ do
if Abs(bbb[i]) gt NewConditionalUpperBoundForAi0 then
FitsInTheNewBox:=false; break i;
end if;
end for;
if FitsInTheNewBox eq true then passes:=false; break; end if;

for ll in I do
LAMBDAprime:=0;
for i:=1 to #JJJ do
LAMBDAprime:=LAMBDAprime + bbb[i]*beta[ll][JJJ[i]];
end for;
if SpecialCase[ll] eq true then
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) ne bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
else //SpecialCase[ll] eq false
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDAprime) lt bbb[jl[ll]]*hh[ll] + dd[ll] then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;
```

```
for ll in I do
if SpecialCase[ll] eq false then
LAMBDA:=0;
for i:=1 to #JJJ do
LAMBDA:=LAMBDA + bbb[i]*beta[ll][JJJ[i]];
end for;
if bbb[jl[ll]] gt (1/hh[ll])*(1/(p[ll]-1) - Valuation(delta2[ll]))
and Valuation(LAMBDA) ne bbb[jl[ll]]*hh[ll] + Valuation(delta2[ll]) then
passes:=false; break ll;
end if;
end if;
end for;
if passes eq false then break; end if;


break;
end while; //end hack while loop

/*If the tuple passes all the tests, add it to the list of exceptional
tuples*/
if passes eq true then
Remove(~bbb,#JJJ+1); Remove(~bbb,1); Include(~ExceptionalTuples, bbb);
end if;


end while; //enumeration loop

//Enumeration done

ConditionalUpperBoundForA[i0]:=NewConditionalUpperBoundForAi0;

end if; //end of IF controlled by EnumerationCost(GammaC,(DDD+min)^2)

end if; /* end of IF controlled by NewConditionalUpperBoundForAi0 ge
           OldUnconditionalUpperBoundForA */
```

```
end while; /* this is the while loop that increases log(C) if necessary,
              the while loop controlled by flag */


end for; //end i0 loop

if JumpToEndOfRealReduction eq false then
//////////////
/*
Choose c11 to find an optimal i0-conditional upper bound for A for i0 in
{s+1,...,s+2t} from Lemma 19.1
*/
///////////
MIN:=UpperBoundForA;

c11:=(1/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
```

```
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

c11:=((1000000 - 1)/1000000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;

for i:=1 to 999 do
c11:=(i/1000)*c10/(n-1);

c15:=0;
if t gt 0 then
min:=Abs(Imaginary(thetaC[s+1]));
for ii:=2 to t do
j:=s-1 + 2*ii;
if Abs(Imaginary(thetaC[j])) lt min then min:=Abs(Imaginary(thetaC[j]));
end if;
```

```
end for; //end ii
if Floor(-Log(min)/c11) gt 0 then c15:=Floor(-Log(min)/c11); end if;
end if;

max:=Ceiling((c8prime + c9prime*UpperBoundForN)/(c10-(n-1)*c11))-1;
if max lt
Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1 then
max:=Ceiling((c8primeprime + c9primeprime*UpperBoundForN)/(c10-c11))-1;
end if;
if max lt c15 then max:=c15; end if;
//now max is the upper bound for A in Lemma 19.1
if max lt MIN then MIN:=max; end if;
end for;  //end i loop

//MIN is now an i0-conditional upper bound for A for i0 in {s+1,...,s+2t}

/////
/*
Compute the new unconditional bound for A
*/
/////

OldUpperBoundForA:=UpperBoundForA;
NewUpperBoundForA:=Max([Max(ConditionalUpperBoundForA),MIN]);

if NewUpperBoundForA lt OldUpperBoundForA then
Improvement:=true;
UpperBoundForA:=NewUpperBoundForA;
for i:=1 to r do
B[1+v+i]:=UpperBoundForA;
end for;
end if;

end if; //controlled by JumpToEndOfRealReduction
```

```
if UpperBoundForA lt 0 then
//there are no solutions to the current case of (11)
continue iiii;
end if;



end if; //end of IF for distinguishing (s>=3) (s=1,2) cases

end if; //end of IF for s=0, s>=3, s=1or2 cases

//done finding new upper bound for A

UpperBoundForH:=Max([UpperBoundForN, UpperBoundForA]);

print("Upperbounds for the n_1 and A, respectively:");
UpperBoundForn;
Floor(UpperBoundForA);
end while;  //end loop for repeated refined reductions (the improvement
            //loop)

////////////////////////////////////////////////////////////////////////
/*
End Of Loop For Repeated Refined Reductions (the improvement loop)
*/
////////////////////////////////////////////////////////////////////////




////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////
//////////////////////
/*
Final Sieve (Section 22)
```

```
*/
//////////////////////
//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
/*
Compute
NumberOfTuplesToCheck = the number of tuples to sieve through (excluding
the [small number of] exceptional tuples)

Select rational primes q_1,...q_k such that each q_i has at least three
prime ideal factors in O_K that have residue degree one and such that the
product q_1 ... q_k is > NumberOfTuplesToCheck

For each q=q_h, pick three residue degree 1 prime ideal factors of q in
O_K: qq_1, qq_2, qq_3.  For each  qq_i, we compute the integers m_i, A_i,
P_ij, E_ij from Section 22.  Actually, we compute these not as integers
but as elements in the finite field Z/qZ.  We store these elements in the
sequence
Q[h][i]=[P_i1, ..., P_iv, E_i1, ..., E_ir, A_i, m_i]
Q[h][i]=[P_{i,JJJ[2]-1}, ..., P_{i,JJJ[1+#J]-1},
                                              E_i1, ..., E_ir, A_i, m_i]
       =[P_{i,J[1]}, ..., P_{i,J[#J]}, E_i1, ..., E_ir, A_i, m_i]

We don't store the primes q_1,...,q_k or their prime ideal factors.


*/
////////////////////////////////////////////////////////////////////////////
NumberOfTuplesToCheck:=1;
for i in J do
NumberOfTuplesToCheck := NumberOfTuplesToCheck * (UpperBoundForn[i] + 1);
end for;
NumberOfTuplesToCheck := NumberOfTuplesToCheck * (2*UpperBoundForA + 1)^r;
```

```
ProductOfAllqSelected:=1;
Q:=[**];
hhh:=0;

j:=0;
while ProductOfAllqSelected le NumberOfTuplesToCheck do

ListOfPrimesInInterval:=PrimesInInterval(150*j+1,150*(j+1));

for i:=#ListOfPrimesInInterval to 1 by -1 do
if ProductOfAllqSelected gt NumberOfTuplesToCheck then break i; end if;
q:=ListOfPrimesInInterval[i];
DecompositionOfq:=Decomposition(OK,q);
if #DecompositionOfq ge 3 then
IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered:=[];
for ii:=1 to #DecompositionOfq do
if InertiaDegree(DecompositionOfq[ii][1]) eq 1 then
Append(~IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered,ii);
end if;
if #IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered eq 3 then
break ii;
end if;
end for; //ii
if #IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered eq 3 then

hhh:=hhh+1;
Q[hhh]:=[];
ProductOfAllqSelected:=ProductOfAllqSelected*q;
ZmodqZ:=FiniteField(q, 1);
for ii in IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered do
temp:=[];
for jj:=1 to #J do
temp[jj]:=ZmodqZ ! (pi[J[jj]][kk[J[jj]]] mod DecompositionOfq[ii][1]);
end for;
```

```
for jj:=1 to r do
temp[#J+jj]:=ZmodqZ ! (eps[jj] mod DecompositionOfq[ii][1]);
end for;
temp[1+#J+r]:=ZmodqZ ! ((OK ! alpha*zeta) mod DecompositionOfq[ii][1]);
temp[2+#J+r]:=ZmodqZ ! ((OK ! theta) mod DecompositionOfq[ii][1]);
Append(~Q[hhh],temp);
end for; //ii

end if;// controlled by
       //#IndicesOfFirstThreeResidueDegreeOnePrimesAboveqEncountered eq 3
end if;// controlled by #DecompositionOfq ge 3
end for; //i

j:=j+1;
end while;



////////////////////////////////////////////////////////////////////////
/*
Suppose bbb[i] = b_{JJJ[1+i]}, for i=1 to #JJJ-1.
Note: bbb[i] = b_{JJJ[1+i]} = b_{1+J[i]}=n_{J[i]} for i:=1 to #J
Note: bbb[#J+i] = b_{JJJ[1+#J+i]} = b_{1+v+i}=a_i for i:=1 to r

We define BBBupper and BBBlower so that
BBBlower[i] <= bbb[i] <= BBBupper[i]
*/
////////////////////////////////////////////////////////////////////////
BBBupper:=[];
BBBlower:=[];
for i:=1 to #J do
BBBupper[i]:=UpperBoundForn[J[i]];
BBBlower[i]:=0;
end for;
for i:=#J+1 to #J+r do
```

```
BBBupper[i]:=UpperBoundForA;
BBBlower[i]:=-UpperBoundForA;
end for;



//Solutions:=[];

////////////////////////////////////////////////////////////////////////
/*
Sieve through all tuples in the box defined by the bounds on the n_i
(i in J) and the a_i
*/
////////////////////////////////////////////////////////////////////////
print("Starting the sieve:");
count:=0; //used in SieveBox for testing.  Can be removed.
bbb:=[]; //needed for SieveBox() to work

SieveBox(~bbb,~Solutions,~count,~BBBlower,~BBBupper,~Q,~J,~JJJ,
~ValueOfn,~kk,~hh,~ss,~tt,~ImageOfzetaC,~ImageOfalphaC,1);

////////////////////////////////////////////////////////////////////////
/*
Sieve through all exceptional tuples that we found in the refined
reduction steps.

After this procedure is done, Solutions will contain all the exceptional
solutions (X,Y,z_1,...,z_v) of the sign-relaxed version of Thue-Mahler
equation (1) that also satisfy the current case of (11).
Recall:  Every solution of the Thue-Mahler equation satisfies some case
of (11).
*/
////////////////////////////////////////////////////////////////////////
bbb:=[];
//ExceptionalTuples;
```

```
//ExceptionalTuplesTest[iiii]:=ExceptionalTuples;
for i:=1 to #ExceptionalTuples do

temp:=ExceptionalTuples[i];
for j:=1 to #temp do
bbb[j]:=IntegerRing() ! temp[j];
end for;


//Test the tuple bbb for the congruences (25)
passes:=true;
for hhh:=1 to #Q do
//test the tuple for the q=q_hhh congruence (25)
CongruenceTest(~bbb,hhh,~passes,~Q,~J);
if passes eq false then break hhh; end if;
end for;

if passes eq true then

/* Use the tuple bbb (where bbb[i]=b_{JJJ[1+i]}, i=1 to #JJJ-1) to get
the tuple bb (where bb[i] = b_{1+i}, i=1 to v).  Compute the
corresponding X,Y. Test if (X,Y,b_{2},...,b_{1+v}) gives a solution of
the Thue-Mahler equation (1) and record the solution if so.
*/
ThueMahlerEquationTest(~bbb,~Solutions,~ImageOfzetaC,~ImageOfalphaC,~J,
~ValueOfn,~kk,~hh,~ss,~tt);

end if;

end for;
```

```
////////////////////////////////////////////////////////////////////////
/*
At this point almost all of the solutions of the sign-relaxed version of
the Thue-Mahler equation have been found.  In the langauge of Section 6,
the ones we have found come from instances of (11) with \zeta equal to an
element of T^{\prime}.  The missing ones come from instances of (11) with
\zeta equal to an element of $T \setminus T^{\prime}$.  The missing ones
are of the form (-x,-y,z_1,...,z_v) where (x,y,z_1,...,z_v) is a solution
we have found.

Now we find the missing solutions.
*/
////////////////////////////////////////////////////////////////////////
SolutionsCopy:=Solutions;
for ii:=1 to #SolutionsCopy do
sol:=SolutionsCopy[ii]; //sol = (X,Y,z_1,...,z_v)
sol[1]:=-sol[1];
sol[2]:=-sol[2];
//now sol = (-X,-Y,z_1,...,z_v)

//Build RHS and LHS expressions
RHSexpression:=a;
for i:=1 to v do
RHSexpression *:= p[i]^(sol[2+i]);
end for;
LHSexpression:= sol[1]^n + c[n]*sol[2]^n;
for i:=1 to n-1 do
LHSexpression:=LHSexpression + c[i]*sol[1]^(n-i)*sol[2]^i;
end for;

/*test if sol=(-X,-Y,z_1,...,z_v) is a solution of the Thue-Mahler
equation (in the form Abs(LHSexpression)-Abs(RHSexpression) = 0).  If so,
append it to Solutions*/
if IsZero(Abs(LHSexpression)-Abs(RHSexpression)) then
Include(~Solutions,sol);
```

```
end if;

end for;



//SolutionsTest[iiii]:=Solutions;
//print("Solutions of the sign-relaxed Thue-Mahler equation");
//Solutions;


////////////////////////////////////////////////////////////////////////
/*
At this point all the solutions of the sign-relaxed version of the
Thue-Mahler equation have been found.  It is now a simple matter to go
through these solutions and eliminate the ones that are not solutions of
the Thue-Mahler equation proper.

Recall: Each element of Solutions is of the form
[X,Y,z_1,...,z_v]
*/
////////////////////////////////////////////////////////////////////////
SolutionsCopy:=Solutions;
for ii:=#SolutionsCopy to 1 by -1 do

//Build RHS and LHS expressions
RHSexpression:=a;
for i:=1 to v do
RHSexpression *:= p[i]^(SolutionsCopy[ii][2+i]);
end for;
LHSexpression:= SolutionsCopy[ii][1]^n + c[n]*SolutionsCopy[ii][2]^n;
for i:=1 to n-1 do
LHSexpression:=LHSexpression
+ c[i]*SolutionsCopy[ii][1]^(n-i)*SolutionsCopy[ii][2]^i;
end for;
```

```
/*test if =(X,Y,z_1,...,z_v) is a solution of the Thue-Mahler equation
(in the form LHSexpression-RHSexpression = 0).  If not, remove it from
Solutions*/
if not IsZero(LHSexpression-RHSexpression) then
Remove(~Solutions,ii);
end if;

end for;



print("Done case iiii=");
iiii;

////////////////////////////////////////////////////////////////////////
end for;  //end iiii loop (the loop through the cases)

break PrecisionLoopVariable;
end for; //end the precision loop

//SolutionsTest;
//ExceptionalTuplesTest;
//Write("Solutions821.txt",SolutionsTest);

print("Solutions of the Thue-Mahler equation:");
return Solutions;

end function;
```