UTRECHT UNIVERSITY

MASTER THESIS

Computation of Units in Number Fields

Author:
Bas JACOBS

Supervisor:
Prof. dr. Frits BEUKERS
Second reviewer:
Prof. dr. Gunther CORNELISSEN

June 14, 2016

Abstract

We discuss three algorithms to find small norm elements in number fields. One of these algorithms is a continued fraction-like algorithm based on the LLL-reduction of positive definite quadratic forms as suggested by Beukers. The other two algorithms are adaptations of that algorithm. We discuss how to find units from these small norm elements and how to extract a system of independent units from that. We discuss properties of these algorithms and compare them to algorithms by Cohen, Diaz y Diaz, Olivier, by Buchmann, Pethő and by Pohst, Zassenhaus. We run tests on an implementation of these algorithms in Mathematica.

Acknowledgements

I would like to thank Frits Beukers for helping me pick the subject and for his help and support along the way. I also thank Gunther Cornelissen for being the second reviewer.

Contents

1 Introduction **1**

2 Quadratic Forms **3**
- 2.1 Lattices 3
- 2.2 Reduced lattices and forms 4
  - 2.2.1 Minkowski reduced 4
  - 2.2.2 HKZ reduced 5
- 2.3 LLL reduction 5
  - 2.3.1 LLL reduced lattice bases 5
  - 2.3.2 LLL reduced forms 7

3 Implemented unit algorithms **9**
- 3.1 Geodesic algorithm 9
- 3.2 Another quadratic form 12
- 3.3 Reduction in directions 14

4 Existing unit algorithms **17**
- 4.1 Cohen, Diaz y Diaz, Olivier 17
- 4.2 Buchmann, Pethő 20
- 4.3 Pohst, Zassenhaus 23

5 Computing the unit group **26**
- 5.1 Computing the roots of unity 26
- 5.2 Computing fundamental units 27
  - 5.2.1 Generating units 28
  - 5.2.2 Constructing a set of independent units 28

6 Implementation and results **30**
- 6.1 Description of code 30
- 6.2 Tests and comparisons 31
  - 6.2.1 Choosing the parameters 31
  - 6.2.2 Statistics 34
  - 6.2.3 High degree number fields 36

A Mathematica Code **42**

Chapter 1

Introduction

In this thesis, $K = \mathbb{Q}(\alpha)$ will denote an algebraic number field obtained by adjoining to $\mathbb{Q}$ a root α of an irreducible polynomial f of degree n. Let $\sigma_1, \ldots, \sigma_n$ be the embeddings of K in $\mathbb{C}$ of which r_1 are real and $2r_2$ are complex. Then the *degree* of K is $[K : \mathbb{Q}] = n = r_1 + 2r_2$. We will write $\mathcal{O}_K$ for the ring of integers of K, which has integral basis $\alpha_1, \ldots, \alpha_n \in \mathcal{O}_K$. The *discriminant* of K, denoted Δ_K, is the square of the determinant of the $n \times n$ matrix with entries $\sigma_i(\alpha_j)$. An element $x \in \mathcal{O}_K$ can be written uniquely as

$$x = \sum_{i=1}^{n} x_i \alpha_i,$$

with $x_i \in \mathbb{Z}$. We will write $x^{(i)} = \sigma_i(x)$ for its ith conjugate.
By the *norm* of x, denoted $N(x)$, we mean

$$N(x) = N_{K/\mathbb{Q}}(x) = \prod_{i=1}^{n} x^{(i)}.$$

We are interested in the elements x of $\mathcal{O}_K$ with $N(x) = \pm 1$, which are called *units*. The units of $\mathcal{O}_K$ form a group, which we will denote by $\mathcal{O}_K^*$. In 1846, Dirichlet [9] showed that this group is finitely generated of rank $r_1 + r_2 - 1$.

Theorem 1.1 (Dirichlet's Unit Theorem)**.** *Let $\mathcal{O}_K$ be the ring of integers of a field K, admitting r_1 real and $2r_2$ complex embeddings, and write μ for the group of roots of unity in $\mathcal{O}_K$. Then μ is finite, and $\mathcal{O}_K/\mu$ is a free abelian group of rank $r_1 + r_2 - 1$.*

A proof can be found in [26]. The theorem implies that we can write

$$\mathcal{O}_K^* = \mu \times \langle \eta_1 \rangle \times \cdots \times \langle \eta_{r_1+r_2-1} \rangle,$$

where μ is the group of roots of unity in $\mathcal{O}_K^*$ and $\eta_1, \ldots, \eta_{r_1+r_2-1}$ are so-called *fundamental units*. This system of fundamental units is unique up to coordinate transformations and multiplication by roots of unity.
We will give a small example to illustrate this theorem.

Example 1.2. Consider $f(X) = X^5 - 19$, and let $\alpha = 19^{1/5}$ be a root of f. Then $K = \mathbb{Q}(\alpha)$ has degree 5 over $\mathbb{Q}$. Now $\mathcal{O}_K$ has 1 real and 4 complex embeddings, so Dirichlet's Unit Theorem tells us that $\mathcal{O}_K^* = \mu \times \langle \eta_1 \rangle \times \langle \eta_2 \rangle$. It turns out that $\mu = \{\pm 1\}$ and that $\eta_1 = 1 + \alpha + \alpha^3$ and $\eta_2 = 4 + 2\alpha + \alpha^4$ are fundamental units.

In general, if $r_1 > 0$, then we have $\mu = \{\pm 1\}$, since $\mathbb{R}$ contains no other roots of unity. In the totally complex case $r_1 = 0$, the roots of unity are also easily calculated. In Chapter 5, we will discuss how that could be done.

We will give another example, which shows that the ability of finding units can solve certain Diophantine equations.

Example 1.3. The *Pell equation* is the equation $x^2 - dy^2 = 1$, to be solved for positive integers x, y for a given nonzero integer d. This equation has a rich history, and solving it boils down to finding units in a number field. To see this, note that we can rewrite the equation as $(x+y\sqrt{d})(x-y\sqrt{d}) = 1$. Hence, a solution to the Pell equation corresponds to a unit in the ring $\mathbb{Z}[\sqrt{d}]$. On the other hand, if we can find a unit of norm 1 in $\mathbb{Z}[\sqrt{d}]$, we have found a solution to the Pell equation. Here the units ± 1 of $\mathbb{Z}[\sqrt{d}]$ are considered trivial. If $d = 19$, then $(x, y) = (170, 39)$ found in the example above solve the Pell equation. The other solutions are given by powers of $170 + 39\sqrt{19}$.

Note that the above example can be solved using a simple continued fraction algorithm like in [16, p. 11]. However, for larger degree number fields, finding a system of fundamental units is a more difficult task.
In this thesis, we will focus on finding elements of small norm, which we can then combine to create units. From these units, we then try to find a system of fundamental units. In Chapter 3 we discuss three algorithms that we have also implemented. These algorithms are designed to return many elements of bounded norm using quadratic form reductions. We will also discuss some existing algorithms developed by others in Chapter 4. We compare these algorithms and discuss differences and similarities. In Chapter 5, we explain how to obtain a system of fundamental units from elements of small norm. This is implemented together with the algorithms of Chapter 3 in Chapter 6. Before we do all that, we need to introduce the notions of quadratic forms and LLL reduction. This will be the topic of Chapter 2.

Chapter 2

Quadratic Forms

A *quadratic form* is a polynomial of the form

$$Q(\mathbf{x}) = \sum_{i,j=1}^{n} q_{ij} x_i x_j$$

in the variables $x_1, x_2, \ldots, x_n$ and with $q_{ij} = q_{ji} \in \mathbb{R}$ for $1 \leq i, j \leq n$. To such a quadratic form we associate a matrix $Q = (q_{ij})_{1 \leq i,j \leq n}$. The absolute value of the determinant of Q is called the determinant of the form, which we will denote by $D(Q)$. We call a quadratic form positive definite if $Q(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$ and $Q(\mathbf{x}) = 0$ if and only if $\mathbf{x} = \mathbf{0}$. From now on, with *form* we will mean a positive definite quadratic form. Two forms Q, Q' will be called *equivalent* if there exists $P \in \mathrm{GL}_n(\mathbb{Z})$ such that $Q' = P^t Q P$. Note that $D(Q') = D(Q)$, since $\det P = \pm 1$.
A central task with quadratic forms is finding minima. This is also our goal, because, as we will show later, by finding minima of forms we are able to find elements of small norm in number fields. By $\mu(Q)$ we will denote the minimal non-zero value of the set $\{Q(\mathbf{x}) | \mathbf{x} \in \mathbb{Z}^n\}$. The following theorem by Hermite gives a bound on this minimum in terms of the determinant of Q.

Theorem 2.1 (Hermite). *For every $n \geq 2$ there exists γ_n such that $\mu(Q) \leq \gamma_n D(Q)^{1/n}$ for all positive definite quadratic forms Q in n variables.*

The smallest possible values of γ_n are called *Hermite's constants*. We have in general $\gamma_n \leq 2n/3$.
We will show below that quadratic forms are related to lattices.

2.1 Lattices

A *lattice* L in a vector space V is a subgroup of V of the form

$$L = \mathbb{Z} \cdot \mathbf{b}_1 + \mathbb{Z} \cdot \mathbf{b}_2 + \cdots + \mathbb{Z} \cdot \mathbf{b}_n$$

for linearly independent $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in V$. The integer n is called the *rank* of L. We will look at lattices L of *maximal rank*, meaning that the vector space V has dimension n. Because of our interests, from now on we will look at lattices in $V = \mathbb{R}^n$.
Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in V$ be a $\mathbb{Z}$-basis of L. We define the matrix $Q = (\mathbf{b}_i \cdot \mathbf{b}_j)_{1 \leq i,j \leq n}$, which we will call the *Gram matrix* of the $\mathbf{b}_i$. We define the *determinant* $d(L)$ of L to be $d(L) = \sqrt{\det Q}$. A geometric interpretation is that $d(L)$ is the volume of the parallelepiped spanned by the basis vectors. Let A be the matrix with columns $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$. Then $Q = A^T A$, so $\det Q = (\det A)^2$. Hence if $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in V$ form a $\mathbb{Z}$-basis of L, then $\det Q > 0$. Moreover, Q is positive definite and symmetric, hence it corresponds to a

positive definite quadratic form. In particular, $Q(x_1,\ldots,x_n) = |x_1\mathbf{b_1} + \cdots + x_n\mathbf{b_n}|^2$. If we change the $\mathbb{Z}$-basis, this amounts to replacing A with AP for a $P \in \mathrm{GL}_n(\mathbb{Z})$, hence we get $Q' = P^T QP$. Hence equivalence classes of lattices correspond to equivalence classes of forms.
Conversely, if Q is the $n \times n$ matrix of a positive definite quadratic form, we can decompose Q as $Q = F^{-1}DF$ where F is the matrix of eigenvectors and D the diagonal matrix whose diagonal elements are the corresponding eigenvalues, which are real and positive. Let E be the matrix with entries $E_{ii} = \sqrt{D_{ii}}$, so $E^2 = D$. If we let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be the columns of $A = EF$, then Q is their Gram matrix. Indeed, $A^T A = F^T E^T EF = F^{-1}EEF = F^{-1}DF = Q$ Moreover, the vectors $\mathbf{b}_i$ are linearly independent, so they span a lattice in $\mathbb{R}^n$. Similar to before, if we consider a form Q' equivalent to Q, then we have $Q' = P^T QP = P^T A^T AP$ for $P \in \mathrm{GL}_n(\mathbb{Z})$, so we have replaced A by AP and obtained an equivalent lattice.
Hence we have shown that there is a natural correspondence between forms and lattices. There are many reduction algorithms for lattices, which we can now translate to reductions of forms.

2.2 Reduced lattices and forms

We saw before that there are equivalence classes of lattices and forms. For determination of $\mu(Q)$ we want to be able to find "nice" representatives for an equivalence class, which are called *reduced*. The process of finding such representatives is called *reduction*. There are different definitions of reduced, among which there are Hermite, Minkowski, Hermite-Korkin-Zolotarev (HKZ), Venkov and Lenstra-Lenstra-Lovasz (LLL) reduced. We will briefly discuss the notions of Minkowski and HKZ reducedness here.

2.2.1 Minkowski reduced

In 1891, Minkowski [22] came up with a notion of reduced bases, now known as Minkowski reduced bases. We will start by stating the definition in terms of lattice bases.

Definition 2.2. A lattice basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of a lattice L is called *Minkowski reduced* if for all i we have

$$|\mathbf{b}_i| \leq |\mathbf{m}| = |\sum_{j=1}^{n} m_j \mathbf{b}_j| \quad \text{for all } \mathbf{m} \in L \text{ with } \gcd(m_i, \ldots, m_n) = 1.$$

Another way of putting it, is that for all i, $\mathbf{b}_i$ is a shortest lattice element that can be extended to a basis with $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$.
For forms, as one might expect, we have the following definition.

Definition 2.3. A form $Q(\mathbf{x})$ is called *Minkowski reduced* if for all i we have

$$Q(\mathbf{e}_i) \leq Q(\mathbf{m}) \quad \text{for all } \mathbf{m} \in \mathbb{Z}^n \text{ with } \gcd(m_i, \ldots, m_n) = 1.$$

We can see that if we have a Minkowski reduced form $Q = (q_{ij})_{ij}$, then q_{11} is the smallest non-zero value of Q restricted to $\mathbb{Z}^n$. Hence if we are able to calculate a Minkowski reduced form equivalent to a given form Q, we know $\mu(Q)$. In [15], an algorithm to construct a Minkowski reduced lattice basis is discussed. However, the number of inequalities that characterise Minkowski reducedness grows very quickly, making the

algorithm run in exponential time with respect to n. Hence the algorithm is impractical for our purposes.

2.2.2 HKZ reduced

The notion of HKZ reducedness was initiated in 1850 by Hermite [17] and in 1873 by Korkin and Zolotareff [19]. Let μ_{ij} be as defined in (2.2). We have the following definition.

Definition 2.4. A lattice basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of a lattice L is called *HKZ reduced* if the following conditions hold:

1. $|\mu_{ij}| \leq \frac{1}{2}$ for $i < j$,
2. $|\mathbf{b}_1|$ is the smallest non-zero vector of L,
3. the orthogonal projection of the vectors $\mathbf{b}_2, \ldots, \mathbf{b}_n$ to $\mathbf{b}_1$ is HKZ reduced.

For forms, we have the following definition, where μ_{ij} is as defined in the recursive form of Q.

Definition 2.5. A form $Q(\mathbf{x})$ is called *HKZ reduced* if the following conditions hold:

1. $\mu_{ij} \leq \frac{1}{2}$ for $i < j$,
2. $b_1 = \mu(Q)$,
3. the form $Q - b_1(x_1 + \mu_{12}x_2 + \cdots + \mu_{1n}x_n)^2$ in $x_2, \ldots, x_n$ is HKZ reduced.

Again, it is easily seen that the two definitions are equivalent.
It turns out that when a form is HKZ reduced, then $b_i \leq b_{i+1} + \mu_{i,i+1}^2 b_i$, which is a stronger version of (2.7) which we will see in the next section.
Like with a Minkowski reduced form, if we have a HKZ reduced form, we immediately have $\mu(Q)$. However, again like with Minkowski reducedness, it is very expensive to compute an HKZ reduced basis. Several algorithms exist, for example the algorithm by Kannan [13]. However, all known algorithms run in exponential time.

2.3 LLL reduction

LLL reduction is named after Lovasz, Lenstra and Lenstra, who proposed it in 1982 [20]. Even though it does not give optimal output, it has proven to be fast and yielding good results. It will prove useful for finding units, which we will see in the next chapters. First, we will look at the notion of LLL reduced lattices, then at LLL reduced forms.

2.3.1 LLL reduced lattice bases

Given a basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ of a lattice L, define the vectors $\mathbf{b}_i^*$ ($1 \leq i \leq n$) and the numbers μ_{ij} ($1 \leq i < j \leq n$) recursively by

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ji} \mathbf{b}_j^*, \tag{2.1}$$

$$\mu_{ij} = (\mathbf{b}_i \cdot \mathbf{b}_j^*)/(\mathbf{b}_j^* \cdot \mathbf{b}_j^*). \tag{2.2}$$

This is the well-known Gram-Schmidt process and we know that the $\mathbf{b}_i^*$ form an orthogonal basis of L. Note that if we define A to be the matrix with columns $\mathbf{b}_i$ and A' the matrix with columns $\mathbf{b}_i^*$, then we have

$$A = A' \cdot \begin{pmatrix} 1 & \mu_{12} & \mu_{13} & \dots & \mu_{1n} \\ 0 & 1 & \mu_{23} & \dots & \mu_{2n} \\ 0 & 0 & 1 & \dots & \mu_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} = A'P.$$

Hence we can see that that $d(L)^2 = \prod_{1 \le i \le n} |\mathbf{b}_i^*|^2$, where $|\cdot|$ denotes the Euclidean distance in $\mathbb{R}^n$. Now, we can define the notion of LLL reducedness of a lattice base.

Definition 2.6. The basis $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$ is called *LLL reduced* if

$$|\mu_{ij}| \le \frac{1}{2} \quad \text{for } i < j, \tag{2.3}$$

$$\frac{3}{4}|\mathbf{b}_i^*|^2 \le |\mathbf{b}_{i+1}^* + \mu_{i,i+1}\mathbf{b}_i^*|^2 \quad \text{for } i < n. \tag{2.4}$$

We have the following theorem which gives bounds for the $\mathbf{b}_i$.

Theorem 2.7. *Let $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$ be an LLL reduced basis of a lattice L. Then*

1. $d(L) \le \prod_{i=1}^n |\mathbf{b}_i| \le 2^{n(n-1)/4} d(L)$.
2. $|\mathbf{b}_1| \le 2^{(n-1)/4} d(L)^{1/n}$.
3. *For every $\mathbf{x} \in L$ with $\mathbf{x} \neq 0$, we have $|\mathbf{b}_1| \le 2^{(n-1)/2}|\mathbf{x}|$.*

Proof. We start by proving part 1. We know that $d(L)^2 = \prod_{1 \le i \le n} |\mathbf{b}_i^*|^2$. We can rearrange and square (2.1), to get

$$|\mathbf{b}_i|^2 = |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \mu_{ji}^2 |\mathbf{b}_j^*|^2,$$

by orthogonality of the $\mathbf{b}_i^*$. Hence, we have $d(L)^2 \le \prod_{1 \le i \le n} |\mathbf{b}_i|^2$ and the first inequality follows. We have

$$|\mathbf{b}_{i+1}^*|^2 \ge (3/4 - \mu_{i,i+1}^2)|\mathbf{b}_i^*|^2 \ge |\mathbf{b}_i^*|^2/2,$$

where the first inequality follows from (2.4) and the orthogonality of $\mathbf{b}_i^*$, and the second inequality follows from (2.3). By induction, we now have

$$|\mathbf{b}_j^*|^2 \le 2^{i-j}|\mathbf{b}_i^*|^2 \tag{2.5}$$

for $i \ge j$. This gives

$$|\mathbf{b}_i|^2 = |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \mu_{ji}^2 |\mathbf{b}_j^*|^2 \le |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} 2^{i-j}|\mathbf{b}_i^*|^2/4 \le (2^{i-1}+1)|\mathbf{b}_i^*|^2/2, \tag{2.6}$$

giving the second inequality and proving 1.
For part 2, note that we can combine (2.5) and (2.6) to get that for $i \ge j$, we have $|\mathbf{b}_j|^2 \le (2^{i-2} + 2^{i-j-1})|\mathbf{b}_i^*|^2$. Setting $j = 1$ and multiplying the inequalities for $1 \le i \le n$,

we get part 2.
For part 3, note that there exists an i such that $\mathbf{x} = \sum_{1\le j\le i} r_j \mathbf{b}_j = \sum_{1\le j\le i} s_j \mathbf{b}_j^*$, for $r_i \in \mathbb{Z}$, $s_j \in \mathbb{R}$ and with $r_i \neq 0$. From (2.1), we see that $r_i = s_i$, so we have $|\mathbf{x}|^2 \ge s_i^2 |\mathbf{b}_i^*|^2 = r_i^2 |\mathbf{b}_i^*|^2 \ge |\mathbf{b}_i^*|^2$, since $r_i \neq 0$ is an integer. By (2.5), we have $|\mathbf{x}|^2 \ge 2^{1-j}|\mathbf{b}_1^*|^2 \ge 2^{1-n}|\mathbf{b}_1^*|^2$, from which part 3 follows. □

We can see that even though the vector $\mathbf{b}_1$ in a reduced basis of L does not have to be the shortest non-zero vector in L, it is not too far from it. Hence, indeed the results are in that sense not optimal, but the algorithmic advantage over other types of reducedness in combination with the above bounds make it a powerful notion nevertheless.
Given a basis of a lattice, the *LLL Algorithm* transforms the basis vectors so that they form an LLL reduced basis. The algorithm is simple and very efficient and is given below. It is implemented in most mathematics software packages, such as Mathematica, PARI/GP and SageMath. The algorithm requires a lattice basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ and returns an LLL reduced basis. For more details, we refer to [6].

Algorithm 2.8.

1. *Start*

   Compute $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$.

2. *Reduction*

   (a) For $i = 2$ to n
   (b) For $j = i - 1$ to 1
   (c) $\mathbf{b}_i \leftarrow \mathbf{b}_i - \mu_{ij}\mathbf{b}_j$

3. *Swap*

   (a) If $\exists i$ such that $\frac{3}{4}|\mathbf{b}_i^*|^2 > |\mathbf{b}_{i+1}^* + \mu_{i,i+1}\mathbf{b}_i^*|^2$, $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$. Go to 1.

The notion of LLL reduced forms is very similar to that of LLL reduced lattices, which is not a surprise, given the correspondence between lattices and forms discussed before. We will treat it for completeness and to be able to compare the theorems and results.

2.3.2 LLL reduced forms

We can write a form $Q(\mathbf{x})$ in the so-called *recursive form*

$$\begin{aligned} Q(\mathbf{x}) &= b_1(x_1 + \mu_{12}x_2 + \cdots + \mu_{1n}x_n)^2 \\ &\quad + b_2(x_2 + \mu_{23}x_3 + \cdots + \mu_{2n}x_n)^2 \\ &\quad \vdots \\ &\quad + b_{n-1}(x_{n-1} + \mu_{n-1,n}x_n)^2 + b_n x_n^2. \end{aligned}$$

We can write the corresponding matrix as $Q = P^T Q' P$, with

$$P = \begin{pmatrix} 1 & \mu_{12} & \mu_{13} & \ldots & \mu_{1n} \\ 0 & 1 & \mu_{23} & \ldots & \mu_{2n} \\ 0 & 0 & 1 & \ldots & \mu_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \quad \text{and} \quad Q' = \begin{pmatrix} b_1 & 0 & 0 & \ldots & 0 \\ 0 & b_2 & 0 & \ldots & 0 \\ 0 & 0 & b_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & b_n \end{pmatrix}.$$

Hence we see that if we let A and A' be as in Section 2.3.1 and $Q' = (A')^T A'$, then $Q = P^T (A')^T A' P = A^T A$. Therefore, if we define $b_i = |\mathbf{b}_i^*|^2$ and let the μ_{ij} defined above correspond to those in Section 2.3.1, then Q' is Gram matrix of the lattice spanned by $\mathbf{b}_i^*$ and Q is Gram matrix of the lattice spanned by $\mathbf{b}_i$. We can now define LLL reducedness of a form.

Definition 2.9. We call the form Q *LLL reduced* if

$$|\mu_{ij}| \leq \frac{1}{2} \quad \text{for } i < j, \tag{2.7}$$

$$\frac{3}{4} b_i \leq b_{i+1} + \mu_{i,i+1}^2 b_i \quad \text{for } i < n. \tag{2.8}$$

As one would expect, for quadratic forms, we have a theorem similar to Theorem 2.7.

Theorem 2.10. *Let Q be an LLL reduced form in n variables and denote by $\mathbf{e}_i$ the ith basis vector of $\mathbb{R}^n$. Then*

1. $D(Q) \leq \prod_{i=1}^n Q(\mathbf{e}_i) \leq 2^{n(n-1)/2} D(Q)$.
2. $Q(\mathbf{e}_1) \leq 2^{(n-1)/2} D(Q)^{1/n}$
3. $Q(\mathbf{e}_1) \leq 2^{n-1} \mu(Q)$.

Proof. By the discussion above, it is clear that Q is the gram matrix of the basis $\mathbf{b}_i$. Hence we have $D(Q) = d(L)^2$ and $Q(\mathbf{e}_i) = |\mathbf{b}_i|^2$. Now it should be clear that this theorem is a direct translation of Theorem 2.7 to forms.
A proof for forms specifically can be found in [3]. □

This theorem will prove useful for proving the results of the algorithms of the next chapter.
Like with lattices, the process of finding a LLL reduced form is called *LLL reduction* of the form. An implementation as given in [3] uses *shift* and *swap* operations. A shift is a substitution of the form $x_r \to x_r + a x_s$ for $1 \leq r < s \leq n$ and $a \in \mathbb{Z}$ such that $|\mu_{rs}| \leq 1/2$. A swap for $1 \leq r \leq n$ interchanges the variables x_r, x_{r+1}.
In [3] and [21], explicit formulae for b_i and μ_{ij} in terms of the coefficients of Q are given, which allow for LLL reduction to be applied directly on the form Q. For certain quadratic forms, this gives a version of LLL reduction that can be used for finding units in a number field. This is explained in the next chapter and implemented later.

Chapter 3

Implemented unit algorithms

We introduce a notation which allows us to rewrite the LLL reducedness conditions in a for us more convenient way. For proofs of the statements, we refer to [3] and [21].

Theorem 3.1. *Let Q be a form in n variables with matrix $(q_{ij})_{i,j=1,\ldots,n}$ and let b_i and μ_{ij} be the coefficients of the recursive form of Q. For i, j with $1 \leq i \leq j \leq n$, we define*

$$B_{ij} = \begin{vmatrix} q_{11} & \cdots & q_{1,i-1} & q_{1,j} \\ q_{21} & \cdots & q_{2,i-1} & q_{2,j} \\ \vdots & \ddots & \vdots & \vdots \\ q_{i1} & \cdots & q_{i,i-1} & q_{i,j} \end{vmatrix},$$

with $B_{00} = 1$. Then $b_i = B_{i,i}/B_{i-1,i-1}$ for all $1 \leq i \leq n$. Also, $\mu_{ij} = B_{ij}/B_{ii}$ for all $1 \leq i < j \leq n$.
If we also define

$$C_i = \begin{vmatrix} q_{11} & \cdots & q_{1,i-1} & q_{1,i+1} \\ q_{21} & \cdots & q_{2,i-1} & q_{2,i+1} \\ \vdots & \ddots & \vdots & \vdots \\ q_{i-1,1} & \cdots & q_{i-1,i-1} & q_{i-1,i+1} \\ q_{i+1,1} & \cdots & q_{i+1,i-1} & q_{i+1,i+1} \end{vmatrix},$$

then we have $C_i/B_{i-1,i-1} = b_{i+1} + \mu_{i,i+1}^2 b_i$.

With this notation, we can rewrite the LLL reducedness conditions as follows.

Corollary 3.2. *Let Q be a quadratic form and let $B_{i,j}$ and C_i be as defined above. We call the form Q LLL reduced if*

$$\begin{aligned} 2|B_{ij}| &\leq B_{ii} \quad \text{for } i < j, \\ \frac{3}{4}B_{ii} &\leq C_i \quad \text{for } i < n. \end{aligned}$$

The shifts and swaps defined in the previous chapter can now be expressed in terms of the determinants B_{ij} and C_i. In the next sections, we will use quadratic forms and these determinants to find units.

3.1 Geodesic algorithm

Let $K = \mathbb{Q}(\alpha)$ be a number field of degree $n+1$ with a real embedding with integral basis $1, \alpha_1, \alpha_2, \ldots, \alpha_n$. In [3] and [21], the authors consider the quadratic form given by

$$Q_t = x_1^2 + x_2^2 + \cdots + x_n^2 + t(x_0 + x_1\alpha_1 + \cdots + x_n\alpha_n)^2$$

in $n+1$ variables x_i for $0 \leq i \leq n$ and a parameter t. Its matrix has the form

$$Q_t = \begin{pmatrix} t & t\alpha_1 & t\alpha_2 & \dots & t\alpha_n \\ t\alpha_1 & 1+t\alpha_1^2 & t\alpha_1\alpha_2 & \dots & t\alpha_1\alpha_n \\ t\alpha_2 & t\alpha_1\alpha_2 & 1+t\alpha_2^2 & \dots & t\alpha_2\alpha_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t\alpha_n & t\alpha_1\alpha_n & t\alpha_2\alpha_n & \dots & 1+t\alpha_n^2 \end{pmatrix}.$$

By Gaussian elimination, this can be transformed to

$$\begin{pmatrix} t & t\alpha_1 & t\alpha_2 & \dots & t\alpha_n \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

From this we see that $\det(Q_t) = t$. Assume that this form has a minimum at $\mathbf{y} = (y_0 + y_1 + \cdots + y_n)$. Then by 2.1, we have

$$Q_t(\mathbf{y}) = y_1^2 + y_2^2 + \cdots + y_n^2 + t(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)^2 \leq \gamma_{n+1}t^{1/(n+1)}.$$

This gives $t(y_0+y_1\alpha_1+\cdots+y_n\alpha_n)^2 \leq \gamma_{n+1}t^{1/(n+1)}$ and $y_1^2+y_2^2+\cdots+y_n^2 \leq \gamma_{n+1}t^{1/(n+1)}$. Multiplying the two inequalities gives

$$(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)^2 \cdot (y_1^2 + y_2^2 + \cdots + y_n^2)^n \leq \gamma_{n+1}^{n+1},$$

hence

$$|y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \cdot (y_1^2 + y_2^2 + \cdots + y_n^2)^{n/2} \leq \gamma_{n+1}^{(n+1)/2} \leq (n+1)^{n/2}. \tag{3.1}$$

This makes it tempting to hope that by finding minima of quadratic forms of the above type, we may find elements of small norm.
Using LLL reduction, we can find elements close to minima relatively easily. The following geodesic algorithm from [3] and [21] does just that.

Algorithm 3.3.

1. *Initialization*

$$\begin{aligned} Q_t^{(0)} &= x_1^2 + x_2^2 + \cdots + x_n^2 + t(x_0 + x_1\alpha_1 + \cdots + x_n\alpha_n)^2, \\ P^{(0)} &= I_{n+1}, \\ k &\leftarrow 0. \end{aligned}$$

2. *Repeat*

   (a) Determine the maximum of the set $\{t \mid Q_t^{(k)} \text{ is LLL reduced}\}$ and call this t_k.

   (b) Perform LLL reduction on $Q_{t_k+\epsilon}^{(k)}$ for infinitesimal $\epsilon > 0$ and let $A_k \in \mathrm{GL}_{n+1}(\mathbb{Z})$ be such that $\mathbf{x} \to A_k\mathbf{x}$ is the corresponding change of variables.

   (c) Define $Q_t^{(k+1)}(\mathbf{x}) = Q_t^{(k)}(A_k\mathbf{x})$ and $P^{(k+1)} = P^{(k)}A_k$.

   (d) $k \leftarrow k+1$

Since LLL reduction does not give a minimum of a quadratic form, we have a weaker version of 3.1.

Proposition 3.4. *Let* $\mathbf{y} = (y_0, y_1, \ldots, y_n)$ *be the first column of* $P^{(k)}$. *Then*

$$|y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \cdot (y_1^2 + y_2^2 + \cdots + y_n^2)^{n/2} \leq 2^{n(n+1)/4}.$$

Proof. First, note that $\det(Q_t^{(k)}) = t$, since $A_k \in \mathrm{GL}_{n+1}(\mathbb{Z})$. We have $Q_t(P^{(k)}\mathbf{x}) = Q_t^{(k)}(\mathbf{x})$ for every k. Hence $Q_t(\mathbf{y}) = Q_t^{(k)}(\mathbf{e}_1)$ and by part 2 of Theorem 2.10, we have

$$y_1^2 + y_2^2 + \cdots + y_n^2 + t(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)^2 \leq 2^{n/2}t^{1/(n+1)}.$$

Hence we have $t(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)^2 \leq 2^{n/2}t^{1/(n+1)}$ and $y_1^2 + y_2^2 + \cdots + y_n^2 \leq 2^{n/2}t^{1/(n+1)}$. Multiplying these, we get

$$|y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \cdot (y_1^2 + y_2^2 + \cdots + y_n^2)^{n/2} \leq 2^{n(n+1)/4}.$$

□

From this proposition, we can say something about the norm of $y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n$. Note that we have $|y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \leq 2^{n/4}t^{-n/(n+1)}$. As we let t get big, this gives $|y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \ll 1$, so $y_0 \approx -y_1\alpha_1 + \cdots + y_n\alpha_n$. If we write $\alpha_i^{(j)} = \sigma_j(\alpha_i)$, with $\alpha_i^{(1)} = \alpha_i$, we have

$$\prod_{i=1}^{n+1} |y_0 + y_1\alpha_1^{(i)} + \cdots + y_n\alpha_n^{(i)}| \approx |y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \prod_{i=2}^{n+1} |y_1(\alpha_1^{(i)} - \alpha_1^{(1)}) + \cdots + y_n(\alpha_n^{(i)} - \alpha_n^{(1)})|.$$

Call the right-hand side S. We know that

$$\sum_{i=2}^{n+1} |y_1(\alpha_1^{(i)} - \alpha_1^{(1)}) + \cdots + y_n(\alpha_n^{(i)} - \alpha_n^{(1)})|^2 \leq c\,(y_1^2 + \cdots + y_n^2)$$

for a constant $c \in \mathbb{R}^+$. Hence for all $i > 1$, we have

$$|y_1(\alpha_1^{(i)} - \alpha_1^{(1)}) + \cdots + y_n(\alpha_n^{(i)} - \alpha_n^{(1)})| \leq \left(c\,(y_1^2 + \cdots + y_n^2)\right)^{1/2}.$$

If we multiply these inequalities for all $i > 1$,

$$S \leq |y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n| \cdot \left(c\,(y_1^2 + y_2^2 + \cdots + y_n^2)\right)^{n/2}.$$

By the proposition, we now have

$$|N(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)| \approx S \leq 2^{n(n+1)/4}c^{-n/2}.$$

Hence the absolute norms of the elements corresponding to the first column of $P^{(k)}$ are bounded.
We have the following proposition, showing a practical aspect of the algorithm.

Proposition 3.5. *Let* $Q_t = x_1^2 + x_2^2 + \cdots + x_n^2 + t(x_0 + x_1\alpha_1 + \cdots + x_n\alpha_n)^2$. *Then the determinants* B_{ij} *and* C_i *as defined before are of the form* $ut + v$ *where* $v \in \mathbb{Z}$ *and where* u *is quadratic in* α_i.

For a proof, we refer to [21]. This proposition shows that the values of $t \geq 1$ for which $Q_t(\mathbf{x})$ is LLL reduced are closed intervals in $\mathbb{R}_{\geq 1}$. Hence it is easy to find t_k which 'break' the LLL conditions, allowing us to reduce again to find elements that may have small norm.
Moreover, it is shown in [3] that even though the rules for updating the B_{ij} and C_i are not linear, the only non-linear part consists of division by an integer. These two observations allow this algorithm to be very practical and easy to implement.
In order to find units, we perform the above algorithm, and in each iteration we look at the first column of $P^{(k)}$, which we denote by $(y_0, \ldots, y_n)$. By the discussion after Proposition 3.4, we see that $N(y_0 + y_1\alpha_1 + \cdots + y_n\alpha_n)$ is bounded. We can calculate this norm, and check if it is a unit. If it is not, we may be able to divide it by another element of the same norm to get a unit. This is further explained in Chapter 6.
Note, however, that we required the field K to have at least one real embedding. Otherwise, the quadratic form is not necessarily real valued and our theory does not apply. If we change the quadratic form to $Q_t = x_1^2 + x_2^2 + \cdots + x_n^2 + t|x_0 + x_1\alpha_1 + \cdots + x_n\alpha_n|^2$, the quadratic form stays real valued. However, Proposition 3.5 no longer holds and the determinants can become quadratic in B_{ij} and C_i. This results in forms that cannot be reduced by changing t. Moreover, the results of the algorithm can in theory (and do in practice) depend on the choice of the real embedding.
In the next section, we consider another quadratic form and discuss its properties.

3.2 Another quadratic form

Let K be a number field of degree n with integral basis $\alpha_1, \ldots, \alpha_n$, let r_1 be the number of real embeddings of K, and r_2 be the number of complex embeddings. We write $\alpha_i^{(j)} = \sigma_j(\alpha_i)$, with $\alpha_i^{(1)} = \alpha_i$. Index the conjugates such that we have $\alpha_i^{(j)} \in \mathbb{R}$ for $1 \leq j \leq r_1$ and $\overline{\alpha_i^{(j)}} = \alpha_i^{(j+r_2)}$ for $r_1 + 1 \leq j \leq r_1 + r_2$, where $\overline{x}$ denotes the complex conjugate of x.
In the quadratic form of the previous section, we use one embedding of the algebraic integer $x_0 + x_1\alpha_1 + \cdots + x_n\alpha_n$. Now, we want to look at all embeddings of such an integer, and use them all in some sense. To that end, we define

$$Q_t = t \cdot |x_1\alpha_1^{(1)} + \cdots + x_n\alpha_n^{(1)}|^2 + |x_1\alpha_1^{(2)} + \cdots + x_n\alpha_n^{(2)}|^2 + \cdots + |x_1\alpha_1^{(n)} + \cdots + x_n\alpha_n^{(n)}|^2$$

if σ_1 is a real embedding, and

$$Q_t = t \cdot |x_1\alpha_1^{(1)} + \cdots + x_n\alpha_n^{(1)}|^2 + \cdots + t|x_1\alpha_1^{(1+r_2)} + \cdots + x_n\alpha_n^{(1+r_2)}|^2 + \cdots + |x_1\alpha_1^{(n)} + \cdots + x_n\alpha_n^{(n)}|^2$$

otherwise. We want to use Algorithm 3.3 with this quadratic form. We will first show some properties of this form.
First, note that in both cases this quadratic form is real-valued. Next, we can see that in the real case, its matrix is given by

$$Q_t = \begin{pmatrix} t\mathrm{Re}(\alpha_1^{(1)}\overline{\alpha_1^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_1^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\mathrm{Re}(\alpha_1^{(1)}\overline{\alpha_n^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_1^{(k)}\overline{\alpha_n^{(k)}}) \\ t\mathrm{Re}(\alpha_2^{(1)}\overline{\alpha_1^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_2^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\mathrm{Re}(\alpha_2^{(1)}\overline{\alpha_n^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_2^{(k)}\overline{\alpha_n^{(k)}}) \\ \vdots & \ldots & \vdots \\ t\mathrm{Re}(\alpha_n^{(1)}\overline{\alpha_1^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_n^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\mathrm{Re}(\alpha_n^{(1)}\overline{\alpha_n^{(1)}}) + \sum_{k=2}^n \mathrm{Re}(\alpha_n^{(k)}\overline{\alpha_n^{(k)}}) \end{pmatrix}.$$

If K has a real embedding, then $\det(Q_t) = |\Delta_K| \cdot t$. In the complex case, we have an other matrix, with $\det(Q_t) = |\Delta_K| \cdot t^2$. We will prove this in the next section in a more general setting.
The following algorithm is simply a version of Algorithm 3.3 with the new quadratic form.

Algorithm 3.6.

1. *Initialization*

   $Q_t^{(0)} = t \cdot |x_1\alpha_1^{(1)} + \cdots + x_n\alpha_n^{(1)}|^2 + |x_1\alpha_1^{(2)} + \cdots + x_n\alpha_n^{(2)}|^2 + \cdots + |x_1\alpha_1^{(n)} + \cdots + x_n\alpha_n^{(n)}|^2$,

   $P^{(0)} = I_n$,

   $k \quad \leftarrow 0$.

2. *Repeat*

   (a) Determine the maximum of the set $\{t \,|\, Q_t^{(k)}$ is LLL reduced$\}$ and call this t_k.

   (b) Perform LLL reduction on $Q_{t_k+\epsilon}^{(k)}$ for infinitesimal $\epsilon > 0$ and let $A_k \in \mathrm{GL}_n(\mathbb{Z})$ be such that $\mathbf{x} \to A_k\mathbf{x}$ is the corresponding change of variables.

   (c) Define $Q_t^{(k+1)}(\mathbf{x}) = Q_t^{(k)}(A_k\mathbf{x})$ and $P^{(k+1)} = P^{(k)}A_k$.

   (d) $k \leftarrow k+1$

We would like to have Propositions similar to 3.4 and 3.5. Indeed, Proposition 3.4 translates to the following.

Proposition 3.7. *Let $\mathbf{y} = (y_1, \ldots, y_n)$ be the first column of $P^{(k)}$. Then,*

$$|N(y_1\alpha_1 + \cdots + y_n\alpha_n)| \leq 2^{n(n-1)/4}|\Delta_K|^{1/2}.$$

Proof. We have $Q_t(P^{(k)}\mathbf{x}) = Q_t^{(k)}(\mathbf{x})$ for every k. Hence $Q_t(\mathbf{y}) = Q_t^{(k)}(\mathbf{e}_1)$ and by part 2 of 2.10, we have $Q(\mathbf{y}) \leq 2^{(n-1)/2}\det(Q_t)^{1/n}$. First, assume that K has a real embedding. Then, we have

$$t \cdot |y_1\alpha_1^{(1)} + \cdots + y_n\alpha_n^{(1)}|^2 + \cdots + |y_1\alpha_1^{(n)} + \cdots + y_n\alpha_n^{(n)}|^2 \leq 2^{(n-1)/2}|\Delta_K|^{1/n}t^{1/n}.$$

This gives

$$|y_1\alpha_1^{(1)} + \cdots + y_n\alpha_n^{(1)}|^2 \leq 2^{(n-1)/2}|\Delta_K|^{1/n}t^{(1-n)/n}$$

for the first embedding, and for $i > 1$,

$$|y_1\alpha_1^{(i)} + \cdots + y_n\alpha_n^{(i)}|^2 \leq 2^{(n-1)/2}|\Delta_K|^{1/n}t^{1/n}.$$

If we multiply these together, we get

$$\prod_i |y_1\alpha_1^{(i)} + \cdots + y_n\alpha_n^{(i)}|^2 \leq 2^{n(n-1)/2}|\Delta_K|.$$

If K does not have a real embedding, then we have

$$|y_1\alpha_1^{(1)} + \cdots + y_n\alpha_n^{(1)}|^2 \leq 2^{(n-1)/2}|\Delta_K|^{1/n}t^{2/n}t^{-1}$$

for embedding 1 and $1 + r_2$. For $i > 1$,

$$|y_1\alpha_1^{(i)} + \cdots + y_n\alpha_n^{(i)}|^2 \leq 2^{(n-1)/2}|\Delta_K|^{1/n}t^{2/n}.$$

Multiplying these, we get

$$\prod_i |y_1\alpha_1^{(i)} + \cdots + y_n\alpha_n^{(i)}|^2 \leq 2^{n(n-1)/2}|\Delta_K|,$$

which proves the statement. □

This proposition gives a bound on the norms of the elements obtained by the algorithm. This bound depends on the degree of the field and its discriminant. For fields K with a real embedding, we have the following proposition, which is a weaker version of Proposition 3.5.

Proposition 3.8. *Let K be a number field with a real embedding and let $Q_t = t \cdot |x_1\alpha_1^{(1)} + \cdots + x_n\alpha_n^{(1)}|^2 + \cdots + |x_1\alpha_1^{(n)} + \cdots + x_n\alpha_n^{(n)}|^2$. Then the determinants B_{ij} and C_i as defined before are linear in t.*

Proof. Since $\alpha_i^{(1)} \in \mathbb{R}$ for all i, we can write

$$Q_t = \begin{pmatrix} t\alpha_1^{(1)}\alpha_1^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_1^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\alpha_1^{(1)}\alpha_n^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_1^{(k)}\overline{\alpha_n^{(k)}}) \\ t\alpha_2^{(1)}\alpha_1^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_2^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\alpha_2^{(1)}\alpha_n^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_2^{(k)}\overline{\alpha_n^{(k)}}) \\ \vdots & \ldots & \vdots \\ t\alpha_n^{(1)}\alpha_1^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_n^{(k)}\overline{\alpha_1^{(k)}}) & \ldots & t\alpha_n^{(1)}\alpha_n^{(1)} + \sum_{k=2}^n \operatorname{Re}(\alpha_n^{(k)}\overline{\alpha_n^{(k)}}) \end{pmatrix}.$$

We can subtract $\alpha_i^{(1)}/\alpha_1^{(1)}$ times the first row from the ith row to obtain a matrix where t only occurs in the first row. Since in the definition of B_{ij} and C_i, these stay in the first row, the determinants are linear in t. □

However, when K is totally imaginary, Proposition 3.5 does not translate well to our form, as we can see in the following example.

Example 3.9. Consider the totally imaginary field $K = Q(\alpha)$, where α is a root of the polynomial $X^4 + 2$. Let B_{ij} be as defined in Theorem 3.1. Then, we have $B_{22} = (17 + 14t + t^2)/\sqrt{2}$ and $C_2 = (3 + t)\sqrt{2}$, so the condition $\frac{3}{4}B_{22} \leq C_2$ becomes quadratic.

We see that in those cases it becomes a lot harder to determine t_k from Algorithm 3.6. The fact that the conditions can be quadratic, also leads to losing the linear aspect of the algorithm. Moreover, it turns out that in certain cases with totally imaginary K, at a certain point the LLL conditions cannot be satisfied at all. Therefore, we do not use this algorithm on totally complex fields in our implementation and in the tests. We will now discuss yet another quadratic form that does work for totally complex fields.

3.3 Reduction in directions

Using the number field and the indexing from before, consider the quadratic form

$$Q_{t_1,t_2,\ldots,t_n} = \sum_{i=1}^n t_i \cdot |x_1\alpha_1^{(i)} + \cdots + x_n\alpha_n^{(i)}|^2$$

in n variables x_i and n parameters t_i. Note that the quadratic form from the previous section is a special case of this one, with $t_i = 1$ for $i = 2, \ldots, n$. In this more general setting, by adjusting the parameters, we can now change the influence each embedding has.

Proposition 3.10. *The quadratic form*

$$Q_{t_1,t_2,\ldots,t_n} = \sum_{i=1}^{n} t_i \cdot |x_1\alpha_1^{(i)} + \cdots + x_n\alpha_n^{(i)}|^2$$

has determinant

$$\det(Q_{t_1\cdots t_n}) = |\Delta_K| \prod_{k=1}^{n} t_k$$

Proof. We can rewrite entry (i,j) as

$$\sum_{k=1}^{n} t_k \mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}) = \sum_{k=1}^{r_1} t_k \alpha_i^{(k)}\alpha_j^{(k)} + \sum_{k=r_1+1}^{n} t_k \mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}).$$

Since $\mathrm{Re}(z) = \mathrm{Re}(\overline{z}) = (z+\overline{z})/2$ for complex z, for every $k \geq r_1 + 1$ we have

$$t_k \mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}) + t_{k+r_2}\mathrm{Re}(\alpha_i^{(k+r_2)}\overline{\alpha_j^{(k+r_2)}}) = t_k \mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}) + t_{k+r_2}\mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}),$$

so

$$t_k \mathrm{Re}(\alpha_i^{(k)}\overline{\alpha_j^{(k)}}) + t_{k+r_2}\mathrm{Re}(\alpha_i^{(k+r_2)}\overline{\alpha_j^{(k+r_2)}}) = (t_k + t_{k+r_2})(\alpha_i^{(k)}\overline{\alpha_j^{(k)}} + \alpha_i^{(k+r_2)}\overline{\alpha_j^{(k+r_2)}})/2.$$

Hence we can write entry (i,j) as

$$\sum_{k=1}^{n} \tau_k \alpha_i^{(k)}\overline{\alpha_j^{(k)}}, \quad \text{with} \quad \tau_k = \begin{cases} t_k & \text{if} \quad 1 \leq k \leq r_1; \\ (t_k + t_{k+r_2})/2 & \text{if} \quad r_1 < k \leq r_1 + r_2; \\ (t_k + t_{k-r_2})/2 & \text{if} \quad r_1 + r_2 < k \leq r_1 + 2r_2. \end{cases}$$

Hence we can see that

$$Q_{t_1,t_2,\ldots,t_n} = A \begin{pmatrix} \tau_1 & 0 & \ldots & 0 \\ 0 & \tau_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \tau_n \end{pmatrix} A^T,$$

with

$$A = \begin{pmatrix} \alpha_1^{(1)} & \alpha_1^{(2)} & \ldots & \alpha_1^{(n)} \\ \alpha_2^{(1)} & \alpha_2^{(2)} & \ldots & \alpha_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_n^{(1)} & \alpha_n^{(2)} & \ldots & \alpha_n^{(n)} \end{pmatrix}.$$

Since $\det(AA^T) = |\Delta_K|$, we have $\det(Q_{t_1\cdots t_n}) = |\Delta_K| \prod_{k=1}^{n} \tau_k$. □

The determinant of the form Q_t from Section 3.2 follows after setting $t_i = 1$ for $i = 2, \ldots, n$.

From now on, we will set $t_{i+r_2} = t_i$ for $i > r_1$ to make sure that the conjugate embeddings get the same parameter. In that case, the determinant becomes simply $|\Delta_K| \prod_{k=1}^n t_k$. With this new quadratic form, we will reduce in "directions", which means that we will choose t_i and LLL reduce the obtained form. If we repeat this, this leads to the following algorithm.

Algorithm 3.11.

1. *Initialization*

   $Q_{t_1,t_2,\ldots,t_n} \leftarrow \sum_{i=1}^n t_i \cdot |x_1\alpha_1^{(i)} + \cdots + x_n\alpha_n^{(i)}|^2.$
   $k \leftarrow 1.$

2. *Repeat*

   (a) Choose $t'_1, t'_2, \ldots, t'_n \in \mathbb{R}$.
   (b) Perform LLL reduction on $Q_{t'_1,t'_2,\ldots,t'_n}$ and let $A_k \in \mathrm{GL}_n(\mathbb{Z})$ be such that $\mathbf{x} \to A_k\mathbf{x}$ is the corresponding change of variables.
   (c) $k \leftarrow k + 1$.

There is a freedom of choosing t'_i in step (a). One can do this in a systematic fashion, similar to the geodesic algorithms. This could be done to hope that no units are missed in the direction we are looking at. Another option is to choose random t'_i and hope that we will find units. This way, we can look at very different directions, but we can miss units that the systematic algorithm does find. Both options are implemented in Chapter 6.
The following proposition gives a bound for the norms of the elements found.

Proposition 3.12. *Let* $\mathbf{y} = (y_1, \ldots, y_n)$ *be the first column of* A_k*. Then,*

$$|N(y_1\alpha_1 + \cdots + y_n\alpha_n)| \leq 2^{n(n-1)/2}|\Delta_K|^{1/2}.$$

Proof. The proof is the same as the proofs of Propositions 3.4 and 3.7 but with another determinant and is therefore not included. □

For a discussion on the results of the three algorithms introduced above, we refer to Chapter 6.

Chapter 4

Existing unit algorithms

In this chapter, we will have a look at a couple of unit algorithms that have been developed by others. There exist more algorithms, but most of them share ideas with the algorithms we chose to discuss. A lot of these algorithms started as ideas by some researchers, and have then been changed and adapted by others. Most of the algorithms have their roots in the 1980s and have been implemented by the authors. In this chapter, we will discuss the algorithms briefly and show the similarities and differences they have to each other and to the algorithms of Chapter 3.

4.1 Cohen, Diaz y Diaz, Olivier

In 1997, Cohen, Diaz y Diaz and Olivier [7] describe the implementation of an algorithm which computes the class group and the unit group of a number field. It is based on ideas of Buchmann [4] and its implementation is used in software packages PARI/GP and SageMath. In Cohen's "A Course in Computational Algebraic Number Theory" [6], the algorithm is explained more thoroughly. The algorithm actually computes the class group of a number field, getting the fundamental units "for free". The algorithm is the fastest algorithm known asymptotically, but one needs to accept the Generalized Riemann Hypothesis (GRH) for its correctness. The GRH is assumed in step 1 and 4 of the algorithm. For more details, we refer to remark below the algorithm.
We will start explaining some number theoretic definitions needed for this algorithm.
Denote by I and J two nonzero fractional ideals of $\mathcal{O}_K$. We say that I and J are related, denoted $I \sim J$, whenever there exist nonzero elements a and b of $\mathcal{O}_K$ such that $(a)I = (b)J$. This is an equivalence relation and its equivalence classes are called the ideal classes of $\mathcal{O}_K$. Ideal classes can be multiplied and the class of principal ideals is an identity element for this multiplication. With this multiplication, the set of fractional ideal classes are an abelian group, called the *ideal class group* of $\mathcal{O}_K$, denoted $Cl(K)$. The size of this group is called the *class number* of K, and is denoted by $h(K)$.
Let $\eta_1, \ldots, \eta_{r_1+r_2-1}$ be a system of fundamental units of $\mathcal{O}_K$. Number the embeddings into $\mathbb{C}$ that are not conjugates by $1, \ldots, r_1 + r_2$. Let c_j be 1 if embedding j is real and 2 otherwise. Then the determinant of the $(r_1 + r_2 - 1) \times (r_1 + r_2)$ matrix with entries $c_j \log|\eta_i^j|$ is called the *regulator* of K, denoted by $R(K)$. If $\mathcal{O}_K^*$ is finite, we put $R(K) = 1$. Unlike the discriminant, the regulator is a positive real number which is usually expected to be transcendental.
In the algorithm, the authors work with ideals rather than elements of the field. The authors cut the algorithm in the following 5 steps, which are explained in more detail below.

Algorithm 4.1.

1. Calculate the *Minkowski bound* and consider prime ideals with norms up to this bound. Denote these by $g_1, \ldots, g_k$. It is known that they generate $Cl(K)$.

2. Find many (say l) relations in the class group among the $\overline{g}_i$. Write these relations as
$$\prod_{i=1}^{k} g_i^{m_{i,j}} = \alpha_j \mathbb{Z}_K \quad (1 \leq j \leq l),$$
where $m_{i,j} \in \mathbb{Z}$ and $\alpha_j \in K$.

3. Let $M = (m_{i,j})_{1\leq i\leq k, 1\leq j\leq l}$ be the $k \times l$ matrix of exponents, and let $V = (\alpha_j)_{1\leq j\leq l}$ be the vector of the α_j. Perform Hermite and Smith normal form reductions on M, doing the same operations on V. This way, we obtain a tentative class group and unit group. Let $h'(K)$ and $R'(K)$ be the corresponding tentative class number and regulator.

4. By using the *analytic class number and regulator formula*, check that the product $h'(K)R'(K)$ is correct up to a factor of 2. If it is not, find a few more relations and go back to step 3.

5. Now $h'(K) = h(K)$ and $R'(K) = R(K)$. From the data obtained above, compute a system of fundamental units, and output it as well as the class group.

Remark. If we are willing to accept the GRH, then in step 1 we do not need to go as far as the Minkowski bound. There are other bounds that are usually much smaller than the Minkowski bound, for example as discussed in [2]. The bound proven there is $12\ln^2|D|$. In step 4, GRH is assumed to be able to prove that if $h'(K)R'(K)$ is correct up to a factor of 2, then $h'(K) = h(K)$ and $R'(K) = R(K)$. In this proof, the authors rely on a truncation of the Euler product of the *analytic class number and regulator formula* which can only be done assuming GRH.

In step 1, we can calculate the Minkowski bound
$$M_K = \sqrt{|D|}\left(\frac{4}{\pi}\right)^{r_2}\frac{n!}{n^n},$$
and consider the prime ideals with norms up to this bound. It is known that these generate $Cl(K)$, and they can be found using the Kummer-Dedekind theorem [8]. Note that although these ideals generate $Cl(K)$, there can be ideals amongst these that correspond to the same class or that are principal. Since a priori we do not know $Cl(K)$ (this is one of the things the algorithm gives us), we just work with all prime ideals g_i we found.

In step 2, the relations are found in three ways. Firstly, there are the trivial relations obtained by factoring prime numbers of $\mathbb{Z}$ into ideals of the number field. This is already done in order to find the generators in step 1. Secondly, the authors use algorithms as in [10] to find elements $\alpha \in \mathcal{O}_K$ of small norm. This is done by enumerating elements of a lattice that lie in suitable ellipsoids, similar to Section 4.3. They hope to be able to factor these in terms of the ideals generating $Cl(K)$, obtaining new relations. Finally, the authors generate random exponents u_i and consider the ideal
$$I = \prod_{i=1}^{k} g_i^{u_i}.$$

Let $\alpha_1, \ldots, \alpha_n$ be a $\mathbb{Z}$-basis of this ideal. Choose a vector $v = (v_i)_{1 \le i \le n}$ of real numbers such that $v_{r_2+i} = v_i$ for $r_1 < i \le r_1 + r_2$. Consider the form

$$Q = \sum_{i=1}^{n} e^{v_i} \cdot |x_1 \alpha_1^{(i)} + \cdots + x_n \alpha_n^{(i)}|^2.$$

Perform LLL reduction on this form and let $\beta_1, \ldots, \beta_n$ be the LLL reduced basis of I obtained. Let $\alpha = \beta_1$, which is small by Theorem 2.10. Then, β_i/α form a $\mathbb{Z}$-basis of the fractional ideal I/α which we will call J. Next, the authors try to factor J to get

$$J = \prod_{i=1}^{k} g_i^{v_i},$$

so

$$\prod_{i=1}^{k} g_i^{u_i - v_i} = \alpha \mathcal{O}_K.$$

Note that J does not necessarily factor in g_i. If it does, the authors obtain a new relation. Otherwise, the authors try other random u_i, obtaining another ideal I.

Remark. The quadratic form Q is simply $Q_{t_1,\ldots,t_n}$ from Section 3.3 with $t_i = e^{v_i}$. Hence the reduction of the form Q we performed in Section 3.3 is used in this algorithm, too. Moreover, this step of the algorithm just boils down to finding small $\alpha \in \mathcal{O}_K$ and trying to factor an ideal above it (namely IJ^{-1}) it in terms of g_i. The method of finding these small α is the same as Algorithm 3.11.

In step 3, Hermite and Smith normal form reductions on M (doing the same operations on V) are used to identify certain subdeterminants of the matrices with multiples of the class number and regulator. In particular, certain columns of an obtained matrix correspond to elements $\alpha \in K$ such that

$$\alpha \mathcal{O}_K = \prod_{i=1}^{k} g_i^0 = \mathcal{O}_K.$$

Hence these columns corresponds to units. The details of these matrix reductions are tedious and will not be treated here. They can be found in [6]. For finding units, it boils down to trying to divide elements such that the corresponding row in the matrix of exponents M only contains zeros, such that the above relation holds.
Using a relation between the class number and regulator, it can be shown that if $h'(K)R'(K)$ is correct up to a factor of 2, then we have $h'(K) = h(K)$ and $R'(K) = R(K)$. In particular, truncate the Euler product of the analytic class number and regulator formula is at a carefully chosen place (smaller than $12\log^2|D(K)|$). Denote by z the quantity obtained. Then,

$$\frac{h(K)R(K)}{\sqrt{2}} < z < \sqrt{2}h(K)R(K).$$

To be able to show the inequalities, GRH must be assumed. Now if $h'(K)R'(K) < z\sqrt{2}$, the authors are able to show that $h'(K) = h(K)$ and $R'(K) = R(K)$.
This is checked in step 4, and in step 5, the matrices of step 3 are processed to get $Cl(K)$ and the fundamental units.
For more details, we refer to [6] and [7].
To summarise, this algorithm generates many relations between prime ideals of the

field. Then, using matrix reductions, from these relations a tentative class group and regulator are obtained. These are checked to be correct and can then be used to compute the class group and a system of fundamental units. Although this algorithm is substantially different from the algorithms of Chapter 3, a part of Step 2 coincides with a part of Section 3.3.

4.2 Buchmann, Pethő

Among the many unit group algorithms Johannes Buchmann introduced, there is one algorithm he created together with Atilla Pethő [5] that we will discuss here. The algorithm uses LLL reduction of lattices, which we will translate to quadratic forms. For more details, we refer to the article.
The algorithm finds a system of *independent* units rather than fundamental units. A system $\{\epsilon_1, \ldots, \epsilon_u\} \subseteq \mathcal{O}_K^*$ is called *independent* if $\epsilon_1^{m_1} \ldots \epsilon_u^{m_u} = 1$ implies $m_1 = \cdots = m_u = 0$ for every system of integers $\{m_1, \ldots, m_u\}$. In Chapter 5, we will explain how from such a system we can find fundamental units.
Again, let the number of real embeddings be r_1 and the number of complex embeddings r_2. We number them like before, such that σ_j is a real embedding for $1 \leq j \leq r_1$ and that σ_j and σ_{j+r_2} are conjugate embeddings for $r_1 < j \leq r_1 + r_2$. We write $\sigma_i(\alpha) = \alpha^{(i)}$. The authors construct a sequence of numbers $(\gamma_k)_{k\in\mathbb{N}}$ in $\mathcal{O}_K$ for every $i \in \{1, \ldots, r_1 + r_2\}$. This sequence satisfies

$$|\gamma_k^{(i)}| < |\gamma_{k-1}^{(i)}| \qquad \text{for} \quad k \geq 2, \tag{4.1}$$

$$|\gamma_k^{(j)}| > |\gamma_{k-1}^{(j)}| \qquad \text{for} \quad k \geq 2,\ j \in \{1, \ldots, r_1 + r_2\},\ j \neq i. \tag{4.2}$$

Hence the absolute value of the chosen embedding becomes smaller and the rest become larger. Moreover, the numbers γ_k are constructed in such a way that they are of bounded norm. The numbers γ_k are pairwise distinct. Hence after a finite amount of steps, two of them are associated to a unit, in the sense that $\gamma_{k_2}/\gamma_{k_1} = \epsilon_i$ for a certain $k_1, k_2 \in \mathbb{N}$. This unit satisfies

$$|\epsilon_i^{(i)}| < 1 \quad \text{and} \quad |\epsilon_i^{(j)}| > 1 \quad \text{for } j \neq i. \tag{4.3}$$

To construct the sequence $(\gamma_k)_{k\in\mathbb{N}}$, start with $\gamma_1 = 1$. For every k we compute a number β_k such that

$$|\beta_k^{(i)}| < 1 \quad \text{and} \quad c_1 > |\beta_k^{(j)}| > 1 \quad \text{for } j \neq i. \tag{4.4}$$

Here c_1 is a constant depending on the degree and the discriminant of K. We can now set $\gamma_{k+1} = \gamma_k\beta_k$, so $(\gamma_k)_{k\in\mathbb{N}}$ satisfies 4.1 and 4.2. We let β_k be an element of the module $R_k = \frac{1}{\gamma_k}\mathcal{O}_K$. We also want $|N(\beta_k)| \leq c_2/|N(\gamma_k)|$, where c_2 is a constant depending on the degree and the discriminant of K. This way, the conjugates of β_k are small for all k, which makes computations inexpensive. Moreover, this is used to guarantee that γ_k is of bounded norm. In fact, we only need to keep track of the small β_k and not the γ_k. Two elements γ_{k_1} and γ_{k_2} are associated if the modules R_{k_1} and R_{k_2} are the same. Then, the unit can be computed by

$$\epsilon_i = \prod_{l=k_1}^{k_2-1} \beta_l.$$

We will now explain how we can calculate β_k.
First, the authors find a reduced basis of the module R_k. A module basis is called reduced if its image under the mapping

$$\begin{array}{rcl} K & \to & \mathbb{R}^n \\ \alpha & \mapsto & (\alpha^{(1)}, \dots, \alpha^{(r_1)}, \mathrm{Re}\,\alpha^{(r_1+1)}, \dots, \mathrm{Re}\,\alpha^{(r_1+r_2)}, \mathrm{Im}\,\alpha^{(r_1+1)}, \dots, \mathrm{Im}\,\alpha^{(r_1+r_2)}) \end{array}$$

is an LLL reduced lattice basis. This way, the authors can assure that the basis elements are bounded. Specifically, they are able to bound $|\alpha_1^{(i)}|$ from above, which will prove useful later.
Secondly, the authors LLL reduce the columns of a well-chosen matrix. If the embedding i we are considering is a real embedding, the authors define

$$U = \begin{pmatrix} 0 & 0 & 0 & \dots & \delta \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \delta & \dots & 0 \\ 0 & \delta & 0 & \dots & 0 \\ \alpha_1^{(i)} & \alpha_2^{(i)} & \alpha_3^{(i)} & \dots & \alpha_n^{(i)} \end{pmatrix}. \tag{4.5}$$

Here, $\alpha_1, \dots, \alpha_n$ is an LLL reduced basis of R_k in the above sense. The positive constant δ is defined as

$$\delta = 2^{-n/4}|\alpha_1^{(i)}|\kappa^{-n}, \tag{4.6}$$

where $\kappa \geq 1$ is a constant that is chosen in the beginning of the algorithm. Although there are some restrictions on this choice of κ (for which we refer to the article), the choice is rather arbitrary. The Gram matrix of the columns of U is

$$G = \begin{pmatrix} (\alpha_1^{(i)})^2 & \alpha_1^{(i)}\alpha_2^{(i)} & \dots & \alpha_1^{(i)}\alpha_n^{(i)} \\ \alpha_2^{(i)}\alpha_1^{(i)} & \delta^2 + (\alpha_2^{(i)})^2 & \dots & \alpha_2^{(i)}\alpha_n^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_n^{(i)}\alpha_1^{(i)} & \alpha_n^{(i)}\alpha_2^{(i)} & \dots & \delta^2 + (\alpha_n^{(i)})^2 \end{pmatrix},$$

which corresponds to the quadratic form

$$Q_\delta = \delta^2 x_2^2 + \delta^2 x_3^2 + \dots + \delta^2 x_n^2 + (\alpha_1^{(i)} x_1 + \alpha_2^{(i)} x_2 + \dots + \alpha_n^{(i)} x_n)^2.$$

This corresponds to the form of Algorithm 3.3 with $t = 1/\delta^2$. The determinant of Q is $D(Q_\delta) = |\alpha_1^{(i)}|\delta^{n-1}$. Let $\mathbf{y} = (y_1, \dots, y_n)$ be the first column of the transformation matrix corresponding to the reduction. Let β be $\beta = \alpha_1 y_1 + \alpha_2 y_2 + \dots + \alpha_n y_n$. Then, by part 2 of Theorem 2.7, we see that

$$|\beta^{(i)}| \leq 2^{(n-1)/4}|\alpha_1^{(i)}|^{1/n}\delta^{n-1}.$$

Since $\alpha_1, \dots, \alpha_n$ form an LLL reduced basis of R_k, the authors can bound $|\alpha_1^{(i)}|$ from above. Therefore, they are able to bound $|\beta^{(i)}|$ in terms of only constants, n and $|N(\gamma_k)|$. We can also bound $|N(\beta)|$ similar to the discussion after Proposition 3.4. The authors find $|N(\beta)| \leq c_3|N(\gamma_k)|^{-1}$. This means that we can bound the absolute values of the embeddings $\beta^{(j)}$ for $j \neq i$ from below. Hence if we carefully choose δ, we can force 4.4 to hold, hence satisfying 4.1 and 4.2.

If i corresponds to a complex embeddding, the authors proceed in the same way, but with another matrix

$$U = \begin{pmatrix} 0 & 0 & 0 & \dots & \delta \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \delta & \dots & 0 \\ \text{Re } \alpha_1^{(i)} & \text{Re } \alpha_2^{(i)} & \text{Re } \alpha_3^{(i)} & \dots & \text{Re } \alpha_n^{(i)} \\ \text{Im } \alpha_1^{(i)} & \text{Im } \alpha_2^{(i)} & \text{Im } \alpha_3^{(i)} & \dots & \text{Im } \alpha_n^{(i)} \end{pmatrix}. \tag{4.7}$$

The Gram matrix of the columns of U is

$$G = \begin{pmatrix} \text{Re } \alpha_1^{(i)}\overline{\alpha_1^{(i)}} & \text{Re } \alpha_1^{(i)}\overline{\alpha_2^{(i)}} & \text{Re } \alpha_1^{(i)}\overline{\alpha_3^{(i)}} & \dots & \text{Re } \alpha_1^{(i)}\overline{\alpha_n^{(i)}} \\ \text{Re } \alpha_2^{(i)}\overline{\alpha_1^{(i)}} & \text{Re } \alpha_2^{(i)}\overline{\alpha_2^{(i)}} & \text{Re } \alpha_2^{(i)}\overline{\alpha_3^{(i)}} & \dots & \text{Re } \alpha_2^{(i)}\overline{\alpha_n^{(i)}} \\ \text{Re } \alpha_3^{(i)}\overline{\alpha_1^{(i)}} & \text{Re } \alpha_3^{(i)}\overline{\alpha_2^{(i)}} & \delta^2 + \text{Re } \alpha_3^{(i)}\overline{\alpha_3^{(i)}} & \dots & \text{Re } \alpha_3^{(i)}\overline{\alpha_n^{(i)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \text{Re } \alpha_n^{(i)}\overline{\alpha_1^{(i)}} & \text{Re } \alpha_n^{(i)}\overline{\alpha_2^{(i)}} & \text{Re } \alpha_n^{(i)}\overline{\alpha_3^{(i)}} & \dots & \delta^2 + \text{Re } \alpha_n^{(i)}\overline{\alpha_n^{(i)}} \end{pmatrix}.$$

The corresponding quadratic form is

$$Q_\delta = \delta^2 x_3^2 + \cdots + \delta^2 x_n^2 + |\alpha_1^{(i)} x_1 + \alpha_2^{(i)} x_2 + \cdots + \alpha_n^{(i)} x_n|^2,$$

which is similar to the form in the real case. Again, we let $\mathbf{y} = (y_1, \dots, y_n)$ be the first column of the transformation matrix of the reduction. Using arguments similar to the real case, the authors guarantee that $\beta = \alpha_1 y_1 + \cdots + \alpha_n y_n$ satisfies 4.4. For a more detailed explanation, we again refer to [5].

Now we know how to find $\gamma_k^{(i)}$ for all k, giving a unit ϵ_i for k large enough. Hence after applying this for every i, we find $r_1 + r_2$ units ϵ_i satisfying 4.3. We then check if among this set of $r_1 + r_2$ units there are $r_1 + r_2 - 1$ independent units. If this does not hold, we apply the algorithm again but with bigger κ in 4.6.

As we saw, this algorithm is rather different from the algorithms we implemented. The LLL reduction of lattice bases the authors perform corresponds to the LLL reduction of quadratic forms we also considered. However, in this algorithm the author also reduces the bases of the modules, hence resulting in different lattices. We considered geodesic algorithms and algorithms that have a certain randomness. Buchmann and Pethő chose to use a chain of modules R_k and guarantee that the units satisfy certain properties. Therefore, they are able to find an independent system of units. We will discuss how to get from an independent system to fundamental units in Chapter 5.

We conclude by summarising the algorithm for a direction $i \in \{1, \dots, r_1 + r_2\}$. This algorithm should be run for $i = 1, \dots, r_1 + r_2$. This is a very high-level summary, more details can be found in [5].

Algorithm 4.2.

1. *Initialization*

$$k \leftarrow 1,$$
$$R_1 = \mathcal{O}_K.$$

2. *Repeat*

   (a) Find an LLL reduced basis of R_k and call it $\alpha_1, \dots, \alpha_n$.

   (b) Define U as in 4.5 or 4.7 if i corresponds to a real or a complex embedding, respectively.

   (c) LLL reduce the columns of U. Let $\mathbf{y} = (y_1, \dots, y_n)$ be the first column of the transformation matrix corresponding to the reduction.

   (d) Let $\beta_k = \alpha_1 y_1 + \cdots + \alpha_n y_n$ and $R_{k+1} = (1/\beta_k)R_k$.

   (e) If $R_k = R_l$ for $l \leq k$, return $\epsilon_i = \prod_{l=l}^{k} \beta_l$.

   (f) $k \leftarrow k + 1$

4.3 Pohst, Zassenhaus

The third and final algorithm we will discuss is due to Pohst and Zassenhaus and is explained in their book *Algorithmic Algebraic Number Theory* [25]. It uses Minkowski's Convex Body Theorem to find non-trivial lattice points in certain lattices. Those points correspond to elements of bounded norm, which can be used to find units. In Chapter 5, we will discuss how we can find units using those elements. We will start by stating Minkowski's Convex Body Theorem. A proof can for example be found in [25].

Theorem 4.3 (Minkowski's Convex Body Theorem)**.** *Let $C \subseteq \mathbb{R}^n$ be a convex, $\mathbf{0}$-symmetric set and L a lattice. If $V(C) > 2^n d(L)$ or if $V(C) = 2^n d(L)$ and C is compact, then C contains a lattice point $\mathbf{x} \in L$ with $x \neq \mathbf{0}$.*

Here, $V(C)$ is the volume of the set C, which is Jordan-measurable. By $\mathbf{0}$-symmetric, we mean that $\mathbf{x} \in L$ implies $-\mathbf{x} \in L$.
The authors create these types of sets C such that they can find non-trivial lattice points. The authors call these sets *parallelotopes*. The parallelotopes are constructed in a special way which we will explain below.
Let K be a field with $[K : \mathbb{Q}] = n$ and let $\alpha_1, \dots, \alpha_n$ be a basis of $\mathcal{O}_K$. There is a bijective mapping

$$\phi : \mathcal{O}_K \to \mathbb{Z}^n : x_1\alpha_1 + \cdots + x_n\alpha_n \mapsto (x_1, \dots, x_n).$$

The authors use this mapping to go back and forth from the ring of integers to $\mathbb{Z}^n \subset \mathbb{R}^n$, and we will follow the same notation. Consider the set

$$\Pi = \{(x_1, \dots, x_n)^T \in \mathbb{R}^n \mid -1 \leq x_i \leq 1, 1 \leq i \leq n\}.$$

This set is convex, compact and $\mathbf{0}$-symmetric, and has volume $V(\Pi) = 2^n$. If we consider the lattice $L = \mathbb{Z}^n$, then we see that Π satisfies the premises of Minkowski's Convex Body Theorem. Obviously, we could have seen a priori that Π contains non-trivial points of $\mathbb{Z}^n$. However, we will now transform Π in such a way that these premises still hold, but such that we can find other lattice points that correspond to other elements

of small norm.
If we define

$$B = \max\left\{ \prod_{j=1}^{n} |y_1\alpha_1^{(j)} + \cdots + y_n\alpha_n^{(j)}| \mid \mathbf{y} \in \Pi \right\}, \tag{4.8}$$

then we have $|N(\phi^{-1}(\mathbf{x}))| \leq B$ for all $\mathbf{x} \in \mathbb{Z}^n \cap \Pi$.
Choose an element $\omega \in \mathcal{O}_K \backslash \mathbb{Z}$. More about the choice of ω can be found in the end of this section. Then its right regular representation $M_\omega \in \mathbb{Z}^{n\times n}$ is defined by

$$(\alpha_1, \ldots, \alpha_n)\omega = (\alpha_1, \ldots, \alpha_n)M_\omega.$$

We have $N(\omega) = \det M_\omega$, and we can define a linear transformation Ψ_ω by $\Psi_\omega = |N(\omega)|^{-1/n}M_\omega$. Now, Ψ_ω is linear and has determinant ± 1. Therefore,

$$\Psi_\omega(\Pi) := \{|N(\omega)|^{-1/n}M_\omega(x_1, \ldots, x_n)^T \in \mathbb{R}^n \mid -1 \leq x_i \leq 1, 1 \leq i \leq n\}$$

is a convex, $\mathbf{0}$-symmetric parallelotope of volume 2^n.
For $-1 \leq x_1, \ldots, x_n \leq 1$, we have

$$\prod_{j=1}^{n} |N(\omega)|^{-1/n} \, |(\alpha_1^{(j)}, \ldots, \alpha_n^{(j)})M_\omega\mathbf{x}| = |N(\omega)|^{-1} \prod_{j=1}^{n} |\omega^{(j)}(\alpha_1^{(j)}, \ldots, \alpha_n^{(j)})\mathbf{x}| \leq B.$$

Hence the absolute norms of elements of $\phi^{-1}(\Psi_\omega(\Pi) \cap \mathbb{Z}^n)$ are bounded by B.
This means that by using these kinds of transformations, we can find new elements of bounded norm. The only thing left to show is how to compute $\Psi_\omega(\Pi) \cap \mathbb{Z}^n$.
The authors show that there exist unimodular matrices U_ω, U_ω^{-1} such that $M_\omega^T U_\omega$ is in Hermite normal form. We will call this matrix N_ω. It is a lower triangular matrix, and since $\det M_\omega = \pm \det N_\omega$, the product of the diagonal elements of N_ω is up to sign $N(\omega)$.
The authors now create a lower triangular matrix $B_\omega \in \mathbb{Z}^{n\times n}$ such that

$$M_\omega^T U_\omega B_\omega = \begin{pmatrix} |N(\omega)| & 0 & \ldots & 0 \\ 0 & |N(\omega)| & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & |N(\omega)| \end{pmatrix}.$$

They create B_ω by recursively computing its elements from top left to bottom right.
Consider a lattice point $\mathbf{c} = (c_1, \ldots, c_n)^T$ of $\Psi_\omega(\Pi)$. Hence $\mathbf{c} = |N(\omega)|^{-1/n}M_\omega\mathbf{x}$ for $\mathbf{x} \in \mathbb{R}^n$ with $1 \leq x_i \leq 1$ for all i. We define $\mathbf{d} = U_\omega^T\mathbf{c}$. Then $\mathbf{d} \in \mathbb{Z}^n$, since $\mathbf{c} \in \mathbb{Z}^n$. We have

$$B_\omega^T\mathbf{d} = |N(\omega)|^{-1/n}B_\omega^T U_\omega^T M_\omega\mathbf{x} = |N(\omega)|^{(n-1)/n}\mathbf{x}.$$

Hence we can find $\mathbf{d} \in \mathbb{Z}^n$ by solving

$$-|N(\omega)|^{(n-1)/n} \leq \left(B_\omega^T\mathbf{d}\right)_i \leq |N(\omega)|^{(n-1)/n}$$

for $i \in \{1, \ldots, n\}$. Since B_ω^T is an upper triangular matrix this can be done recursively, starting from d_n and working to d_1. When we have $\mathbf{d}$, we calculate

$$\mathbf{c} = (U_\omega^{-1})^T\mathbf{d},$$

yielding an element of $\Psi_\omega(\Pi) \cap \mathbb{Z}^n$.
Hence the authors find elements of small norm using the following algorithm.

Algorithm 4.4.

1. *Initialization*

$$\Pi = \{(x_1, \dots, x_n)^T \in \mathbb{R}^n \mid -1 \leq x_i \leq 1, 1 \leq i \leq n\}.$$

2. *Repeat*

   (a) Choose $\omega \in \mathcal{O}_K \backslash \mathbb{Z}$ and let $M_\omega \in \mathbb{Z}^{n \times n}$ be its right regular representation. Define Ψ to be the linear transformation given by $|N(\omega)|^{-1/n} M_\omega$ with respect to $\alpha_1, \dots, \alpha_n$.

   (b) Compute unimodular matrices U_ω, U_ω^{-1} such that $N_\omega = M_\omega^T U_\omega$ is in Hermite normal form.

   (c) Compute lower triangular $B_\omega \in \mathbb{Z}^{n \times n}$ such that $M_\omega^T U_\omega B_\omega$ is a diagonal matrix with entries $|N(\omega)|$.

   (d) Find $\mathbf{d} \in \mathbb{Z}^n$ by solving $-|N(\omega)|^{(n-1)/n} \leq \left(B_\omega^T \mathbf{d}\right)_i \leq |N(\omega)|^{(n-1)/n}$ for $i \in \{1, \dots, n\}$. Calculate $\mathbf{c} = (U_\omega^{-1})^T \mathbf{d}$.

   (e) Save $\phi^{-1}(\mathbf{c}) = c_1\alpha_1 + \dots + c_n\alpha_n$.

This yields for every ω at least one element $\mathbf{c}$ of absolute norm bounded by B as defined in 4.8. Hence the output of this algorithm is similar to the result of the algorithms of Chapter 3. They all return elements of bounded norm, which must then be processed in order to obtain units. How this could be done is described in Chapter 5.
Note that the algorithm depends on the choice of basis $\alpha_1, \dots, \alpha_n$ in the beginning and the choices of ω in every step. The choice of basis influences B, hence affecting the maximum norm of the elements found. The choice of ω determines the transformation of the parallelotope, hence the elements it contains. The authors do not know how to choose the basis to make B as small as possible. Neither do they know how to choose ω wisely, but they suggest to choose ω to have small absolute norm, to consider a few powers of the element and then to switch to another ω. The algorithm returns elements of bounded norm, so these elements can then be used as ω. Moreover, if ω has small norm, then M_ω will be small too, requiring not much storage when implementing the algorithm.

Remark. This algorithm is indeed very different from the other algorithms discussed in this section, since it does not use any type of reduction to find a short element in a lattice. It depends strongly on the choices of bases and transformation elements, whereas the other algorithms discussed do not. For a more detailed description, we refer to [25].

Chapter 5

Computing the unit group

In the previous chapters, we discussed algorithms that are able to find elements of small norm, systems of independent units and systems of fundamental units. In this chapter, we will discuss how we can use these results to determine the unit group. Let K be a field of degree n with signature (r_1, r_2), so $r_1 + 2r_2 = n$. Moreover, let $\alpha_1, \ldots, \alpha_n$ be an integral basis of $\mathcal{O}_K$. Recall that by Dirichlet's Unit Theorem, the unit group can be written as

$$\mathcal{O}_K^* = \mu \times \langle \eta_1 \rangle \times \cdots \times \langle \eta_{r_1+r_2-1} \rangle.$$

We will start by explaining how to compute the group μ of the roots of unity in $\mathcal{O}_K^*$. This will be based on discussions in [6] and [25]. In Section 5.2, we will discuss how to find fundamental units $\eta_1, \ldots, \eta_{r_1+r_2-1}$ from the outputs of the algorithms of Chapters 3 and 4.

5.1 Computing the roots of unity

An nth root of unity for $n \in \mathbb{Z}^+$ is a number z satisfying $z^n = 1$. A *primitive root of unity* is an nth root of unity that is not a kth root of unity for some smaller k. The number of primitive nth roots of unity is given by $\phi(n)$, where ϕ is Euler's totient function. The zeros of the nth *cyclotomic polynomial* Φ_n are precisely the primitive nth roots of unity. It is given by

$$\Phi_n(z) = \prod_{k=1}^{\phi(n)} (z - z_k) = \prod_{d|n} (z^d - 1)^{\mu(n/d)}.$$

Here, μ is the Möbius function. For more details, we refer to any book on elementary number theory.

Note that the only real roots of unity are ± 1. Hence if $r_1 > 0$, then $\mu = \{\pm 1\}$. From now on, we will assume $r_1 = 0$, so we are in the case of a totally complex field K.

The first way of determining μ is due to [6]. Let $\beta = y_1\alpha_1 + \cdots + y_n\alpha_n$ be a root of unity in K. Then $|\beta^{(i)}|^2 = 1$ for every $i \in \{1, \ldots, n\}$. Hence if we define the form

$$Q(\mathbf{x}) = \sum_{i=1}^{n} |x_1^{(i)}\alpha_1^{(i)} + \cdots + x_n^{(i)}\alpha_n^{(i)}|^2,$$

then $Q(\mathbf{y}) = n$. Moreover, for an element $\alpha \in \mathcal{O}_K \backslash \{0\}$, we have

$$\sum_{i=1}^{n} |\alpha^{(i)}|^2 \geq n\Big(\prod_{i=1}^{n} |\alpha^{(i)}|\Big)^{2/n} \geq n.$$

Here, the first inequality follows from the inequality of arithmetic and geometric means. The second inequality follows from the fact that the norm of α is at least 1. Note that we have equalities if and only if $|\alpha^{(i)}|^2 = 1$ for all i. Hence the minimum non-zero value of $Q(\mathbf{x})$ is n and it is attained for $\mathbf{y}$ precisely when $y_1\alpha_1 + \cdots + y_n\alpha_n$ is a root of unity.
Hence finding roots of unity boils down to finding $\mathbf{y}$ minimising the form Q. Since we now need to find elements really minimising the form, using just LLL reduction will in general not work, since that does not guarantee finding minima. Hence we need to apply other algorithms that are not as fast as LLL reduction. This can be done by enumeration, but an example of a faster algorithm can be found in [10].
Another way to find μ is the following. We know that μ is a finite cyclic group and we will denote its order by g. We know that g must be even since it contains the subgroup $\{\pm 1\}$ of order 2. We will now try to find a primitive gth root of unity, which generates μ. Note that such a primitive gth root of unity generates a subfield of K, hence $\phi(g)|n$. This means that we need to find all even m with $\phi(m)|n$. Since ϕ is multiplicative, this is not hard. Once we have found a list of candidates of g, we have to check for every candidate m if Φ_m has a root in $\mathcal{O}_K$. Now, g is the maximal value for which this is true, and the corresponding root is the generator of μ. For more details, for instance about how to determine if Φ_m has a root in $\mathcal{O}_K$, we refer to [25]. We finish the section with a small example of the second method.

Example 5.1. Define the field $K = \mathbb{Q}(\alpha)$ where α is a root of $X^4 + 3$. Since we have $n = 4$, we have to find even m satisfying $\phi(m)|4$. We get $m \in \{2, 4, 6, 8, 10, 12\}$ and it turns out that

$$\Phi_6(t) = t^2 - t + 1$$

has root $1/2 + \alpha^2/2$. This is an element of $\mathcal{O}_K$ and since $\Phi_m(t)$ has no roots in $\mathcal{O}_K$ for $m \in \{8, 10, 12\}$, the group of roots of unity has order 6. It is generated by $1/2 + \alpha^2/2$ and we have

$$\mu = \left\{ \frac{1+\alpha^2}{2}, \frac{1-\alpha^2}{2}, -1, -\frac{1+\alpha^2}{2}, -\frac{1-\alpha^2}{2}, 1 \right\}.$$

5.2 Computing fundamental units

The algorithm of Cohen, Diaz y Diaz and Olivier outputs a system of fundamental units of the number field. The other algorithms discussed here output a system of independent units (Buchmann, Pethő) or only many elements of small norm (the other algorithms). In this section, we will explain how one can construct fundamental units from the outputs of these algorithms. First, we need a definition. Consider the mapping $\lambda : K\backslash\{0\} \to \mathbb{R}^{r_1+r_2-1}$ given by

$$\lambda(x) = (c_1 \log|x^{(1)}|, \ldots, c_{r_1+r_2-1} \log|x^{(r_1+r_2-1)}|),$$

where $c_i = 1$ for $1 \leq i \leq r_1$ and $c_i = 2$ otherwise. This is called the *logarithmic embedding*. It is a homomorphism from the multiplicative group of K to the additive group in $\mathbb{R}^{r_1+r_2-1}$. The kernel of its restriction to $\mathcal{O}_K^*$ is μ. The image of $\mathcal{O}_K^*$ is an $r_1 + r_2 - 1$ dimensional lattice. Hence to find fundamental units, we need to find $r_1 + r_2 - 1$ fundamental vectors which form a basis of this lattice. From Lemma 6.3 of [25], we know that if $\mu \times \langle\epsilon_1\rangle \times \cdots \times \langle\epsilon_{r_1+r_2-1}\rangle$ is a subgroup of $\mathcal{O}_K^*$ of finite index, then $\lambda(\epsilon_i)$ are linearly independent. As we saw before, the regulator of the number field is the determinant of $\lambda(\mathcal{O}_K^*)$.

5.2.1 Generating units

If an algorithm outputs many elements of bounded norm, a first step would be to collect the units among those elements. We can group the other elements based on their absolute norm. In each group, we can try to divide any two elements to obtain an element of norm ± 1. However, dividing two elements of the same absolute norm in $\mathcal{O}_K$ can yield an element outside of $\mathcal{O}_K$. Hence after dividing two elements of the same norm, we need to check if the result is in the ring of integers. If it is, we have obtained a new unit, otherwise we try to divide other elements. This way, from our elements of bounded norm, we can construct hopefully many units.

Example 5.2. Define the field $K = \mathbb{Q}(\alpha)$ where α is a root of $X^3 - 5$. Our implementation of Algorithm 3.11 gives in a certain instance the following three elements of absolute norm 4:

$$\beta_1 = -1 + \alpha, \quad \beta_2 = -1 - \alpha + \alpha^2, \quad \beta_3 = 29 + 17\alpha + 10\alpha^2.$$

We get $\beta_1/\beta_2 = 1 + \alpha/2 + \alpha^2/2$, which is not an element of $\mathcal{O}_K$. The same holds for β_2/β_3. On the other hand, $\beta_1/\beta_3 = 1 - 4\alpha + 2$, which is not just an element of $\mathcal{O}_K^*$, it turns out to even be a fundamental unit.

Remark. The method in Cohen, Diaz y Diaz and Olivier uses a method which is similar to this one, but more advanced. When they reduce the matrix of exponents, it comes down to combining and dividing ideals based on their prime ideal decomposition. Even though there is more involved, we do a somewhat similar thing here but with principal ideals.

5.2.2 Constructing a set of independent units

We now have hopefully many units, from which we want to construct a set of independent units. Recall that a set of units $\{\epsilon_1, \ldots, \epsilon_u\}$ is called independent if $\epsilon_1^{m_1} \ldots \epsilon_u^{m_u} = 1$ implies $m_1 = \cdots = m_u = 0$ for every set of integers $\{m_1, \ldots, m_u\}$.
Such a set generates a multiplicative group, which we will try to make as big as possible with the units we have generated in the previous section. If we consider the logarithmic embedding of this group, it is an additive group spanning a lattice. We know many points on this lattice, namely the units generated in the previous section. Hence we need to find a basis spanning this lattice.
We will construct this by maintaining a set of independent units. The set starts empty. We will consider the units generated in the previous subsection one by one. We will add a unit if it is independent of the units already in the set. Denote by $\{\epsilon_1, \ldots, \epsilon_u\}$ a set of independent units, their logarithmic embeddings being $\{\mathbf{v}_1, \ldots, \mathbf{v}_u\}$. A new unit ϵ is independent of this set if its corresponding logarithmic embedding $\mathbf{v}$ is linearly independent of $\{\mathbf{v}_1, \ldots, \mathbf{v}_u\}$. This can be checked by considering the rank of the matrix with columns $\mathbf{v}_1, \ldots, \mathbf{v}_u, \mathbf{v}$. If this rank is greater than the rank of the matrix with columns $\mathbf{v}_1, \ldots, \mathbf{v}_u$, we know that $\mathbf{v}$ is linearly independent of $\mathbf{v}_1, \ldots, \mathbf{v}_u$. In that case, we add ϵ and $\mathbf{v}$ to their respective sets. If, however, the rank does not increase, we know that $\mathbf{v}$ is linearly dependent of $\mathbf{v}_1, \ldots, \mathbf{v}_u$.
In that case, we solve $a_1\mathbf{v}_1 + \cdots + a_u\mathbf{v}_u + a\mathbf{v} = \mathbf{0}$ for $a_1, \ldots, a_u, a \in \mathbb{Z}$. This is just a system of linear equations that can be solved easily. By dividing by their greatest common divisor, we can assume that $\gcd(a_1, \ldots, a_u, a) = 1$. We also want $\gcd(a_1, \ldots, a_u) = 1$. This is not always the case, but we can guarantee this by replacing $\mathbf{v}$ with an other vector $\mathbf{v}'$

constructed as follows. Let $d = \gcd(a_1, \dots, a_u) > 1$. Now, since $\gcd(d, a) = 1$, there exist $l, m \in \mathbb{Z}$ such that $dm - al = 1$. Choose such l, m and define $\mathbf{v}' = m\mathbf{v} + l(a_1\mathbf{v_1} + \cdots + a_u\mathbf{v_u})/d$. Since d is a divisor of all a_i, this is an integer combination of $\mathbf{v}$ and $\mathbf{v}_1, \dots, \mathbf{v}_u$. Now, note that $d\mathbf{v}' = dm\mathbf{v} + l(a_1\mathbf{v}_1 + \cdots + a_u\mathbf{v}_u) = \mathbf{v} + l(a\mathbf{v} + a_1\mathbf{v}_1 + \cdots + a_u\mathbf{v}_u) = \mathbf{v}$. Hence we have the new relation

$$(a_1/d)\mathbf{v}_1 + \cdots + (a_u/d)\mathbf{v}_u + a\mathbf{v}' = \mathbf{0}.$$

Via the above construction, we can guarantee to have a relation of the form

$$a_1\mathbf{v}_1 + \cdots + a_u\mathbf{v}_u + a\mathbf{v} = \mathbf{0}$$

for $a_1, \dots, a_u, a \in \mathbb{Z}$, with $\gcd(a_1, \dots, a_u) = 1$.
Now, there exist $b_1, \dots, b_u \in \mathbb{Z}$ satisfying $a_1b_1 + a_2b_2 + \cdots + a_ub_u = a - 1$. These can be found for example by repeating the Euclidean algorithm. Now, we are able to find a system of vectors of size u that contain $\mathbf{v}_1, \dots, \mathbf{v}_u$ and $\mathbf{v}$.

Lemma 5.3. *Let $a_1, \dots, a_u, a$ and $b_1, \dots, b_u$ be as above. Then,*

$$\{\mathbf{v}_1 + b_1\mathbf{v}, \dots, \mathbf{v}_u + b_u\mathbf{v}\}$$

are a linearly independent system of vectors containing $\mathbf{v}_1, \dots, \mathbf{v}_u$ and $\mathbf{v}$.

Proof. We have

$$\mathbf{v} + a_1(\mathbf{v}_1 + b_1\mathbf{v}) + a_2(\mathbf{v}_2 + b_2\mathbf{v}) + \cdots + a_u(\mathbf{v}_u + b_u\mathbf{v}) = \mathbf{v} - a\mathbf{v} + (a-1)\mathbf{v} = \mathbf{0}.$$

Hence the system contains $\mathbf{v}$. Therefore, it contains $\mathbf{v}_i + b_i\mathbf{v} - b_i\mathbf{v}$ hence $\mathbf{v}_i$ for all i. Since $\mathbf{v}_1, \dots, \mathbf{v}_u$ are linearly independent, so is the system we constructed. □

Hence when considering a unit that is dependent on the other units, we can create a new system by the above discussion. Because of the lemma, this set of independent units $\{\epsilon_1\epsilon^{b_1}, \dots, \epsilon_u\epsilon^{b_u}\}$ generates a bigger group than the set we started with. This way, we will enlarge our set of independent units, until we have considered all the units generated in the previous section.
We now have a subgroup of $\mathcal{O}_K^*$ of the form $U = \mu \times \langle\epsilon_1\rangle \times \cdots \times \langle\epsilon_u\rangle$ containing the units generated in the previous section. If $u = r_1 + r_2 - 1$, this subgroup is of finite index in $\mathcal{O}_K^*$. Then the regulator of U is $\mathrm{Reg}(U) = |\det(c_j \log|\epsilon_i^{(j)}|)_{1 \le i,j \le r}|$. The index of U in $\mathcal{O}_K^*$ is given by $\mathrm{Reg}(U)/\mathrm{Reg}(\mathcal{O}_K)$. Hence in our tests, we can compare our regulator to the regulator that PARI/GP returns (which is correct assuming GRH). If they are equal, then we have indeed found a system of fundamental units. Otherwise, we have a subgroup of which we can calculate the index.

Remark. Note that we are not able to guarantee that the group U we have generated is all of $\mathcal{O}_K^*$ if we do not have a way to check our regulator. Hence for fields of which the generator is not known, we are only guaranteed to compute a subgroup of $\mathcal{O}_K^*$, which is of finite index in $\mathcal{O}_K^*$ if we found $r_1 + r_2 - 1$ independent units. In [1] and [25] the authors compute a lower bound of the regulator and use that together with a more advanced analysis to be able to check if the group generated is all of $\mathcal{O}_K^*$. We will not discuss that here and refer to those articles instead.

Chapter 6

Implementation and results

We implemented the algorithms of Chapter 3 in Mathematica. Algorithm 3.3 was implemented by Rianne Maes [21], Algorithms 3.6 and 3.11 were added later. We chose Mathematica because of the high numerical precision it can guarantee. A disadvantage of Mathematica is that it is not fast, compared to for example C. In the following section, we will describe our code. The complete code can be found in Appendix A.

6.1 Description of code

The module `Main[f,algo,param]` is the main function, where `f` is an irreducible polynomial defining the field K of which we want to find the unit group. With the parameter `algo` one decides what algorithm to use, where 1 is Algorithm 3.3, 2 means Algorithm 3.6 and 3 stands for Algorithm 3.11. Depending on `algo`, the parameter `param` is a number determining the number of steps for the first two algorithms, or a Boolean determining if the last algorithm should search in random (True) or systematic (False) directions. In `Main`, first some basic properties of the number field (degree, signature, integral basis) are calculated using built-in Mathematica functions. Then, using `QuadraticForm1[f]`, `QuadraticForm2[f]` or `QuadraticForm3[f]`, the right quadratic form matrix `Q` is calculated. For the first two algorithms, then the module `GeodesicLLL[Q,param]` then calculates elements of small norm. For the last algorithm, this is done by `Directions[Q,param]`. In both cases, the result is stored in the list `elements`. After that, the roots of unity of K are calculated using a Mathematica built-in function. The module `Units[elements]` returns a list of units, which is stored in `units`. Finally, `IndependentUnits[units,f]` computes a maximal system of independent units from that.
The module `GeodesicLLL` as implemented by Rianne Maes [21] is left mostly intact. Some things are removed or renamed, since her focus was different from ours. However, since no big changes have been made, we will refer to her thesis for a description of that implementation.
The module `Directions` performs Algorithm 3.11. First, it generates a number of parameter vectors using `Randomparametervectors` or `Systematicparametervectors`, depending on the previously set parameter `param`. `Randomparametervectors[size,power,number]` returns `number` parameter vectors of length `size` filled with random powers of 2 smaller than `power`. The module `Randomparametervectors[size,power,number]` recursively creates all parameter vectors of the form $\{2^{i_1}, \ldots, 2^{i_n}\}$ with $n =$ `size` and $0 \leq i_j <$ `power` for all j. The way these parameters can be chosen is discussed in the next section. Every such a "direction" $t'_1, t'_2, \ldots, t'_n \in \mathbb{R}$ is then passed to the module `ReduceDirection[Q,direction]`. This module then substitutes $t_1, t_2, \ldots, t_n$ in `Q` for this parameter vector and calls `LLLReduce`, which performs LLL reduction on the

obtained matrix. The elements of small norm are collected and in the end returned by `Directions`.
The module `LLLReduce` is very similar to the version of LLL reduction implemented by Rianne Maes. For a discussion, we refer to [21].

Remark. In Chapter 3, we took the first column of the matrix A_k corresponding to the change of variables after performing LLL reduction. For an element corresponding to such a column, we were able to guarantee an upper bound of the norm. In our implementation, we also collect the elements corresponding to the other columns. In practice, these are also of small norm and they can hopefully be divided to obtain units.

6.2 Tests and comparisons

We used the database of the article [18] to obtain many number fields of given determinants and signatures. We performed tests on these number fields, of which we will discuss the results below. We will refer to Algorithm 3.3 as algo1, to Algorithm 3.6 as algo2 and to Algorithm 3.11 as algo3. All tests have been done on a 2013 MacBook Pro with a 2.4 GHz Intel Core i5 processor. The results have been processed with Python and RapidMiner.

6.2.1 Choosing the parameters

We first want to be able to choose optimal parameters for each of the algorithms. We can then use these parameters to compare the algorithms. To that end, we choose 28 number fields of degree 2 to 8 at random, choosing 4 fields for each degree. We pick these from a large set of number fields with relatively small determinants, generated by the database mentioned before. Using PARI/GP, for each of those number fields we calculate the regulator (hence using the algorithm of Cohen, Diaz y Diaz and Olivier). We also look up the signature and calculate $r = r_1 + r_2 - 1$. The results can be found in the table on the next page. We perform tests on these number fields with our implemented algorithms.
With algo1 and algo2, we try to find an optimal number of *steps* to take in the loop. We run the algorithm for $50, 100, \ldots, 400$ steps and collect the results such as running time, number of units found (after division), number of independent units, regulator and max norm. For all number fields with $r_1 > 0$, algo1 was able to find a system of independent units with regulator equal to that calculated by PARI/GP. In other words, the algorithm is able to construct a system of fundamental units for all fields. For all fields, such a system was already found within 50 steps. The number of steps and the number of units found are linearly correlated, but the time seems to increase exponentially as the number of steps increases. We conclude that for both algorithms, a step size of 50 is enough for these number fields. The algorithms also run sufficiently fast, with at most a couple of seconds for 50 steps. We conclude this analysis with two graphs of the results. The first graph is a scatterplot of the running time (in seconds) on various numbers of steps for algo1. The second graph is a scatterplot of the number of units found for various numbers of steps, also for algo1. Different colours correspond to different polynomials. The results can be seen in Figure 6.1.

Polynomial	Reg (PARI)	| Disc |	r_1	r_2	r
x^2-x-1	0.481	5	2	0	1
x^2-x-39	5.361	157	2	0	1
$x^2-x-198$	9.081	793	2	0	1
x^2-13	1.195	13	2	0	1
x^3-x^2-2x+1	0.525	49	3	0	2
x^3+3x-1	1.133	135	1	1	1
x^3-5	4.812	675	1	1	1
x^3-x^2-6	10.029	996	1	1	1
$x^4-x^3+x^2-x+1$	0.962	125	0	2	1
x^4-x^2+1	1.317	144	0	2	1
x^4+1	1.763	256	0	2	1
$x^4-x^3+2x^2-2x-1$	1.479	1963	2	1	2
$x^5-2x^4+2x^3-x^2+1$	0.347	2209	1	2	2
$x^5-x^4+x^3-3x^2+x-1$	1.436	9664	1	2	2
x^5-2x^2-1	1.042	6581	1	2	2
$x^5-x^4+2x^3-2x^2+1$	1.082	9829	1	2	2
$x^6-3x^5+5x^4-5x^3+5x^2-3x+1$	0.237	12167	0	3	2
$x^6-x^5+x^4-x^3+x^2-x+1$	2.102	16807	0	3	2
x^6-x^3+1	3.397	19683	0	3	2
x^6-x-1	0.741	49781	2	2	3
$x^7-x^6-x^5+x^4-x^3-x^2+2x+1$	0.605	357911	1	3	3
x^7+2x^5-x-1	1.266	854575	1	3	3
$x^7-2x^6+3x^5-3x^4+3x^3-2x^2+2x-1$	0.441	227287	1	3	3
$x^7-x^6+x^4-x^3-x^2+x+1$	0.756	451051	1	3	3
$x^8-4x^7+6x^6-8x^4+4x^3+8x^2-8x+2$	5.498	6553600	0	4	3
$x^8-x^7-x^6+x^5+2x^3+x^2-3x+1$	2.126	3379725	0	4	3
$x^8-3x^7+4x^6+x^5-9x^4+13x^3-9x^2+4x-1$	1.207	7288099	2	3	4
$x^8+x^4+x^2+1$	0.763	4227136	0	4	3

For algo3, we first perform tests using random directions. We have two parameters that can be chosen. First, the amount of parameter vectors can be varied. We let this run from $10, 20, \ldots, 50$. Secondly, the highest power of 2 in each parameter vector can be varied. We choose to let this run from 5 to 30 with steps of 5. For each of these 30 configurations and for each field, we collect the results like we did for algo1 and algo2. The performance of the algorithm varies a lot. Often, a highest power of 2^5 or 2^{10} is not enough. For all fields, except x^3-x^2-6, using 20 parameter vectors of maximal size 2^{20} is enough to find a system of fundamental units. For x^3-x^2-6, using 50 parameter vectors of maximum size 2^{50}, algo3 is able to find such a system. The regulator of the corresponding field is big, which means that the fundamental unit is big in a certain sense, too. This suggests why in this case we need to have higher parameters.
The running time of the algorithm varies wildly. For $x^8-x^7-x^6+x^5+2x^3+x^2-3x+1$, when using 40 parameter vectors of maximum size 2^{30}, the algorithm ran for more than 2 minutes, whereas with 50 parameter vectors of maximum size 2^{30} it took 13 seconds. This is caused by the randomness of the parameter vectors. Because of these results, in the next subsection we will run tests with 20 parameter vectors of maximal size 2^{20}. To deal with the randomness, we run every test multiple times and average the results such as running time and number of units found.

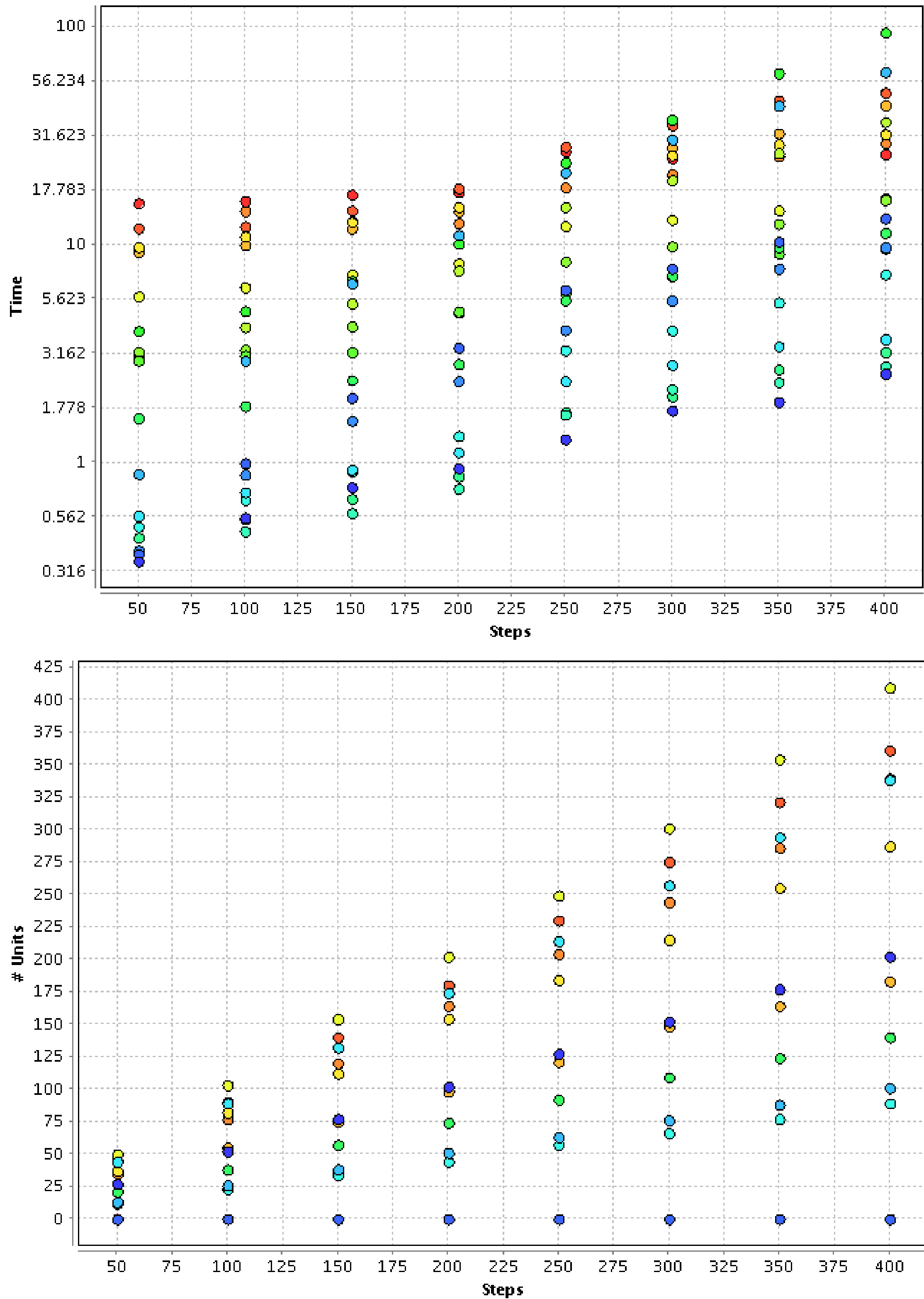

FIGURE 6.1: Scatter plots of running time and number of units found as a function of steps for algo1. Different colours correspond to different fields.

Next, we run tests using systematic parameter vectors. We vary the maximum power of 2 in every parameter vector, hence also the number of parameter vectors considered.

We consider parameter vectors of the form $(2^{5i_1}, 2^{5i_2}, \dots, 2^{5i_{r_1+r_2}})$ with $0 \leq i_j \leq k$ for $k = 2, 3, \dots, 7$ and all j. We chose to multiply every power by 5 to make the elements in the parameter vectors bigger without increasing the number of parameter vectors. For a given k, the number of parameter vectors considered is of order $k^{r_1+r_2}$. Hence the number of reductions to perform increases exponentially, and so does the running time. For $x^7 + 2x^5 - x - 1$ and $k = 7$, the running time is more than 15 minutes, and for the degree 8 fields, the algorithm failed to return an answer within the hour for $k > 4$. However, using $k = 3$, the algorithm is able to find a system of fundamental units for all fields except for $x^3 - x^2 - 6$ within a couple of seconds.

6.2.2 Statistics

Now, we run the algorithms on the chosen parameters on many number fields, to be able to provide statistical results. We run algo1 and algo2, with 50 steps. Algo3 is run with 20 random parameter vectors of maximum size 2^{20} (to which we will refer as algo3 (r)) and with systematical parameter vectors $(2^{5i_1}, \dots, 2^{5i_{r_1+r_2}})$ with $0 \leq i_j \leq 3$ for all j, respectively (algo3 (s)). The results of algo3 (r) are averaged over 5 runs in order to deal with the randomness in the results.
We selected 700 number fields, 100 of each degree from 2 to 8. These number fields were generated by [18] and selected to have small discriminants. We ran the algorithms on each of these number fields and collected the results. The following tables summarise these results.
First we calculate the percentage of number fields for which a system of fundamental units has been found. We break this down on the degrees of the number fields and on $r = r_1 + r_2 - 1$. We also calculate the average running time. The results can be seen in Table 6.1 and 6.2, respectively.
We can see that algo1 and algo3 (r) are the best in terms of percentages and running time. However, algo1 does not work for totally complex fields, whereas algo3 does. What is surprising is that algo3 does not work well for degree 2 fields. In many of these cases, no units were found at all. Even though algo3 works for totally complex fields, it does not work well in those cases. This holds for both the systematic and the random version of algo3. We see that the degree is of big influence on the running time. This is caused by the matrices being larger, hence the calculations will be slower. Moreover, the discriminants are larger on average for higher degree fields. The value of r affects the running time, too, but less significantly so, except for the systematic version of algo3. This is because the value of r is of great influence on the number of parameter vectors to be considered in that algorithm. In the other algorithms, it only influences the search for independent units, which loops over all found units in any case. What is surprising is that algo2 is often much slower than algo1 for the same degree and same r. This may be caused by the fact that the matrix Q contains more information and hence finding its subdeterminants costs more time.
Next, we will look at the norms of the elements obtained by reductions. By Proposition 3.4, 3.7 and 3.12, the norms of certain elements are bounded. However, in our implementation we do not only consider those elements, but rather all elements obtained by reductions. To be able to get a grasp of what kind of elements the algorithms obtain, we plot histograms in which we group elements by their norm. We ran these tests with algo1 and algo2, both with 1000 steps on x^4+3x^2-12 and $x^7-2x^6+3x^5-x^4-x^3+2x^2-1$. For both fields, the results of algo1 and algo2 are very similar. The algorithms find many elements of small absolute norm and few elements of large absolute norm. For both fields, algo2 finds more elements of small norm and less elements of large absolute

Degree	r	algo1	algo2	algo3 (r)	algo3 (s)
2	1	94	95	75	51
3	1	100	100	94	88
	2	100	100	100	100
4	1	-	-	94	86
	2	100	68	82	77
	3	58	42	100	67
5	2	96	96	99	99
	3	90	70	90	90
	4	100	100	100	100
6	2	-	-	99	96
	3	100	90	100	100
7	3	99	94	100	100
	4	100	100	100	100
8	3	-	-	98	88
	4	90	70	100	100
total		96	92	94	87

TABLE 6.1: Percentage of number fields for which a system of fundamental units is found

Degree	r	algo1	algo2	algo3 (r)	algo3 (s)
2	1	0.47	0.54	0.12	0.06
3	1	0.47	0.79	0.09	0.05
	2	0.54	0.85	0.14	0.14
4	1	-	-	0.18	0.08
	2	1.42	2.60	0.27	0.25
	3	1.52	2.89	0.54	2.11
5	2	2.75	5.33	0.45	0.45
	3	3.22	5.51	0.95	3.07
	4	4.14	5.87	0.70	13.14
6	2	-	-	0.73	0.73
	3	5.61	10.39	1.02	3.80
7	3	9.37	17.74	1.88	6.71
	4	9.20	18.28	2.03	26.29
8	3	-	-	2.67	12.15
	4	16.97	28.23	2.23	43.13
total		3.58	6.59	0.90	3.59

TABLE 6.2: Average running time in seconds

Degree	r	fields	50 steps	100 steps	500 steps	1000 steps
9	4	2	1	1	2	2
	5	2	2	2	2	2
	6	2	0	0	1	1
	7	2	0	0	1	1
	8	2	0	0	0	0
10	5	2	2	2	2	2
	6	2	2	2	2	2
	7	2	0	0	0	2
	8	2	0	0	1	1
	9	2	0	0	0	0
11	5	2	1	2	2	2
	6	2	0	0	1	1
	7	2	0	0	0	0

TABLE 6.3: Number of fields for which a system of independent units was found with algorithm 1 and varying numbers of steps

norm than algo1. In the second field, this difference is remarkably big. For x^4+3x^2-12, we also plotted histograms for algo3 (r) and algo3 (s). We considered 80 parameter vectors with maximum size 2^{20} for the random algorithm, and looked at all parameter vectors $(2^{5i_1}, 2^{5i_2}, 2^{5i_3})$ with $0 \leq i_1, i_2, i_3 \leq 5$ for algo3 (s). These algorithms perform worse in terms of the norms of the elements than algo1 and algo2. Although the algorithms find many elements of small norm, also some elements of very large norm are found. We removed these outliers to be able to get a better view of the results. These outliers have norms up to $3 \cdot 10^6$ for both algo3 (r) and algo3 (s). The histograms can be found in 6.2 and 6.3.

6.2.3 High degree number fields

Next, we perform tests on higher degree number fields to see how far we can go. To that end, we collect number fields of degree 10 to 20 and run algo1 and algo3 (r) on these fields. Because of the longer running time and the worse performance for the other two algorithms, we do not run these tests on the other two algorithms. We choose about 6 fields for every degree, of varying signatures.
First, we run algo1 for 50 steps on every field. For 3 out of the 10 fields of degree 9, the algorithm was able to find a system of independent units of maximal size. The same holds for 4 of the 10 fields of degree 10 and 1 field of degree 11. In all those cases, this system of independent units is in fact a system of fundamental units. The fields for which the algorithm is able to find such a system, are fields with many complex embeddings, hence with small $r = r_1 + r_2 - 1$. For higher degrees, the algorithm was not able to find sufficiently many independent units within 50 steps. After increasing the number of steps to 1000, still for no fields of degree 12 or higher the algorithm is able to find r independent units. The results are summarised in Table 6.3. The results suggest that if we go on long enough, we might be able to find a system of fundamental units for number fields of higher degree too. However, for 1000 steps the algorithm already takes 5 minutes for a degree 11 field. We conclude this analysis by an example of the results of a degree 10 field.

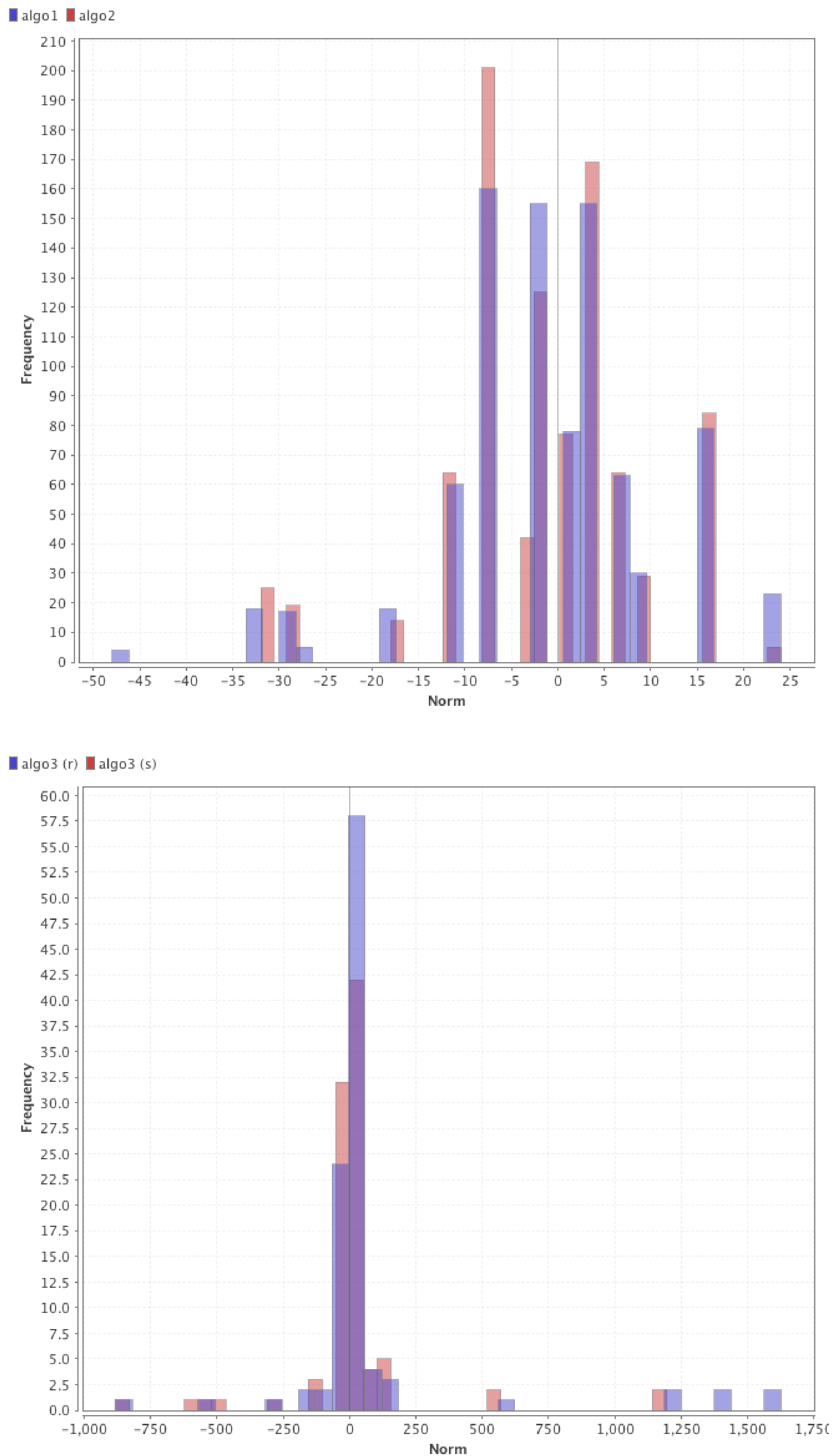

FIGURE 6.2: Histograms of norms of elements obtained by reduction on $x^4 + 3x^2 - 12$

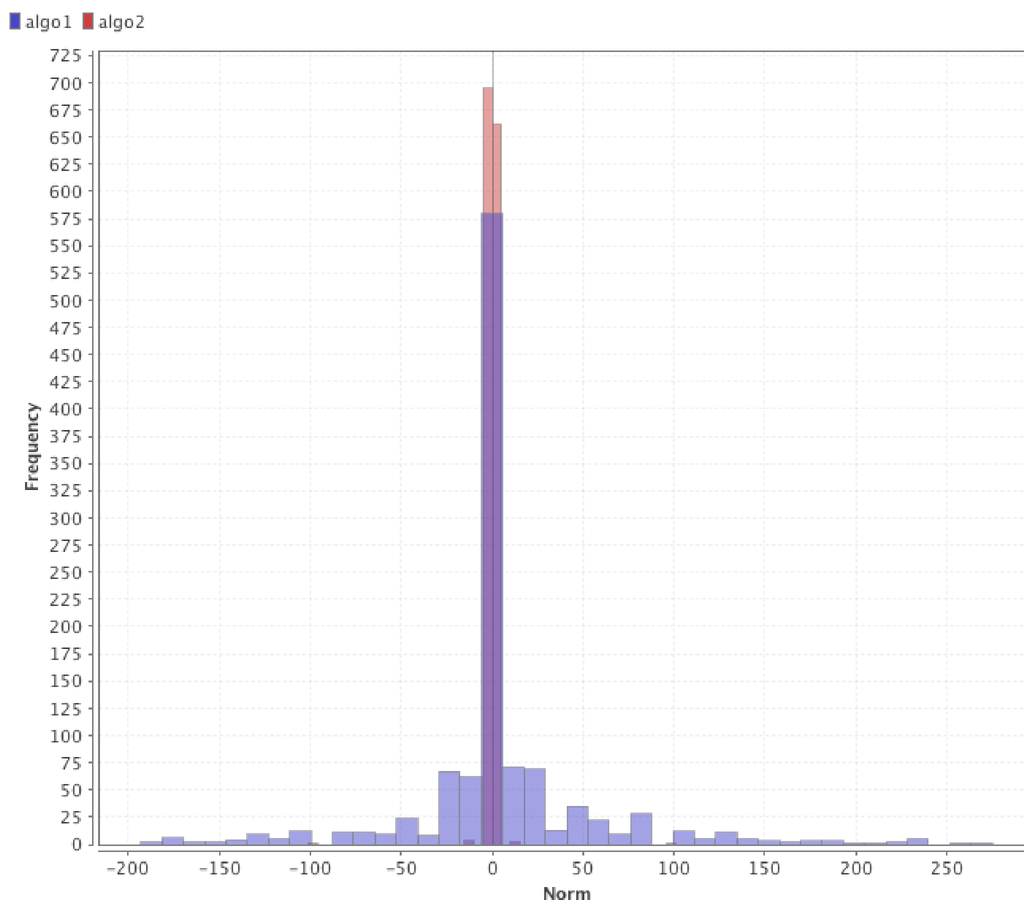

FIGURE 6.3: Histograms of norms of elements obtained by reduction on $x^7 - 2x^6 + 3x^5 - x^4 - x^3 + 2x^2 - 1$

Example 6.1. Consider the field defined by a root of the polynomial

$$x^{10} - 2x^9 + 5x^7 - x^6 + 4x^5 - 12x^4 - 28x^3 + 9x + 1.$$

Its discriminant is approximately $3 \cdot 10^{11}$ and we have $(r_1, r_2) = (6, 2)$, so $r_1 + r_2 - 1 = 7$. PARI/GP computes a regulator of 97.835. If we run algorithm 1 on this field with 50 steps, it finds 2 units that are also independent. Increasing the number of steps to 100, the algorithm also finds 2 independent units. With 500 steps, 9 units are found among which there are 7 independent units. The calculated regulator is 195.670, hence we have found a subgroup of the group of fundamental units of index $195.670/97.835 = 2$. If we run the algorithm with 1000 steps, the algorithm finds 23 units. It finds a system of independent units with regulator 97.835, hence corresponding to a system of fundamental units. With 1000 steps, this computation took about 2 minutes.

We also run algorithm 3 on these high degree number fields with 50 random parameter vectors $(2^i_1, \ldots, 2^i_{r_1+r_2})$ with $i_j \leq 5$ for all j. We do this, because with our previous parameters (20 of such parameter vectors with $i_j \leq 20$), on a degree 14 number field the algorithm takes very long if it ends at all. Therefore, we choose to decrease the size but increase the number of parameter vectors. The algorithm is able to find a system of fundamental units in 42 of the 53 cases of fields of degree 9 up to degree 15. The surprising part is that it performs almost equally well for all degrees and values

of r. The algorithm is also relatively fast, with an average running time of 3 and a maximum of 7.5 minutes. These running times are displayed in Figure 6.4. For number fields of degree 16 and higher, the algorithm has not been able to give a result within a reasonable amount of time.

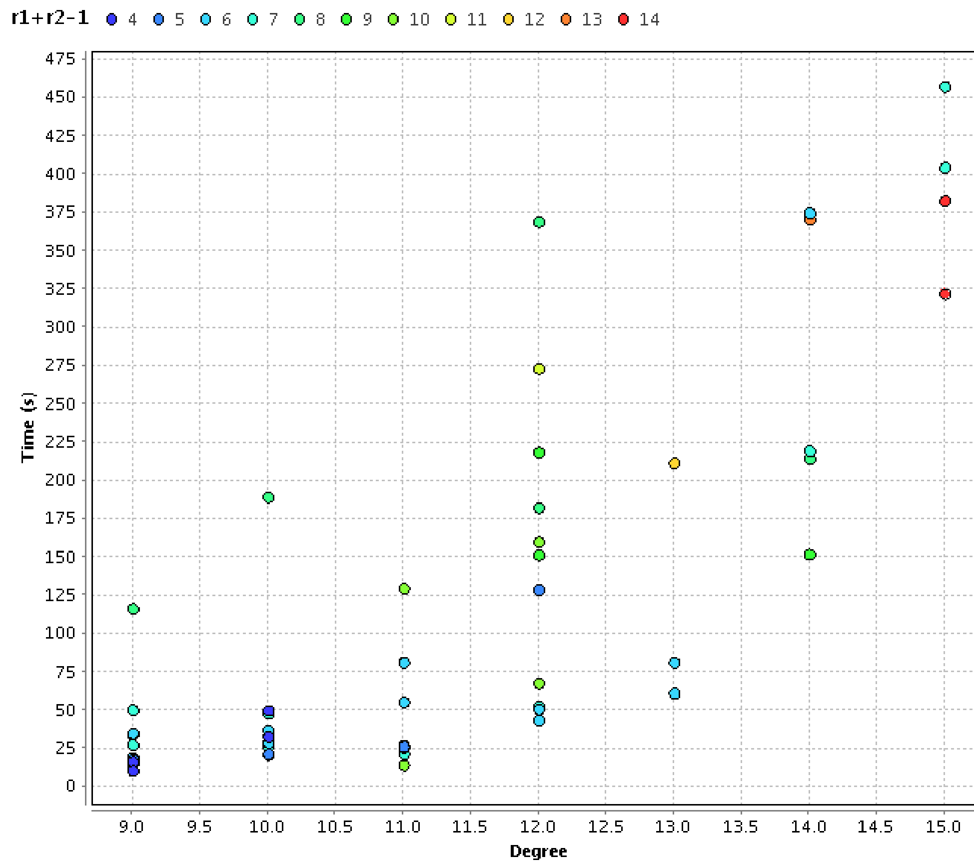

FIGURE 6.4: Running time of algo3 (r) on high degree number fields.

Bibliography

[1] B. Arenz. Computing fundamental units from independent units. *Computational Number Theory*, pages 163–171, 1991.

[2] E. Bach. Explicit bounds for primality testing and related problems. *Math. Comp.*, 1990.

[3] F. Beukers. Geodesic continued fractions and LLL. *Indagationes Mathematicae*, 25:632–645, 2014.

[4] J. Buchmann. A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. *Séminaire de Théorie des Nombres*, 91:27–41, 1990.

[5] J. Buchmann and A. Pethő. Computation of independent units in number fields by Dirichlet's method. *Mathematics of Computation*, 52:149–159, 1989.

[6] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer.

[7] H. Cohen, F. Diaz y Diaz, and M. Olivier. Subexponential algorithms for class group and unit computations. *Journal of Symbolic Computation*, 24:433–441, 1997.

[8] K. Conrad. Factoring after Dedekind.

[9] J. Dirichlet. Zur theorie der complexen einheiten. *Ber. Verhandl. Kgl. Preuss. Akad. Wiss.*, pages 103–107, 1846.

[10] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44:463–471, 1985.

[11] U. Fincke and M. Pohst. A new method of computing fundamental units in algebraic number fields. *EUROCAL '85*, 204:470–478, 1985.

[12] J. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, 2003.

[13] G. Hanrot and D. Stehleé. Improved analysis of kannan's shortest lattice vector algorithm. *Thème SYM*, 2007.

[14] B. Helfrich. Algorithms to construct minkowski reduced and hermite reduced lattice bases. *Theoretical Computer Science*, 41, 1985.

[15] B. Helfrich. Algorithms to construct minkowski reduced and hermite reduced lattice bases. *Theoretical Computer Science*, 41, 1985.

[16] D. Hensley. *Continued Fractions*. World Scientific Publishing Company, 2006.

[17] C. Hermite. Deuxième lette à Jacobi. *Oeuvres de Hermite I*, 1905.

[18] J. Jones and D. Roberts. A database of number fields. *LMS J. Comput. Math.*, 17:595–618, 2004.

[19] A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Mathematische Annalen*, 1873.

[20] H.W. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coeficients. *Mathematische Annalen*, 4, 1982.

[21] R. Maes. An application of LLL in geodesic continued fractions. 2015.

[22] H. Minkowski. Über die positiven quadratischen Formen und über kettenbruchähnliche Algorithmen. *J. Reine Angew. Math.*, 105, 1891.

[23] W. Narkiewicz. *Elementary and Analytic Theory of Algebraic Numbers*. Springer.

[24] M. Pohst and H. Zassenhaus. An effective number geometric method of computing the fundamental units of an algebraic number field. *Mathematics of Computation*, 31(139):754–770, 1977.

[25] M. Pohst and H. Zassenhaus. *Algorithmic Algebraic Number Theory*. Cambridge University Press, 1989.

[26] P. Stevenhagen. *Number Rings*. Leiden University, 2012.

Appendix A

Mathematica Code

Computes units for the field defined by f, where algo=1 means the geodesic algorithm, algo=2 stands for the geodesic algorithm with another form, algo=3 stands for reduction in random directions and algo=4 stands for reduction in systematic directions.

```
In[1]:= Main[f_,algo_,steps_,size_]:=Module[{Q, elements, units},
    n = Exponent[f,x]; (* the degree of the field *)
    {r1,r2}=NumberFieldSignature[Root[f,1]]; (* number of real (r1) and
    complex (r2) embeddings *)
    b=NumberFieldIntegralBasis[Root[f,1]];(* basis of the ring of
    integers of the number field *)
    np=1000; (* the numerical precision  *)
    If[algo>2,
      Q=QuadraticForm3[f]; (* create the required quadratic form *)
      elements = Directions[Q,steps,size]; (* collect elements found by
      algorithm *)
    ,
    If[r1 == 0, Print["No real embeddings."]; Return[]];
    If[algo==1,
      Q=QuadraticForm1[f],(* create the required quadratic form *)
      Q=QuadraticForm2[f]];(* create the required quadratic form *)
    elements = GeodesicLLL[Q,steps]; (* collect elements found by
    algorithm *)
    ];
    elements = DeleteDuplicates[elements];
    units = Units[elements];
    Print["Number of units: ", Length[units]];
    If[Length[units]>0, IndependentUnits[units,f]];
    Print["Roots of unity: ", NumberFieldRootsOfUnity[Root[f,1]]]
  ]
```

QuadraticForm1 creates the square $(n+1)\times(n+1)$ matrix Q for $x_1^2+...+x_n^2+t\alpha^2$, where for each entry the constant and the coefficient of t are separated.

```
In[2]:= QuadraticForm1[f_]:=Module[{Q,a,i,j},
    Q=ConstantArray[0,{n,n,2}];
    a=N[b, np];
    For[i=1,i≤n,i++, (* loop over variables xjxk *)
      For[j=1,j≤n,j++,
        Q[[i,j,1]]=a[[i]]*a[[j]]];
      Q[[i,i,2]]=1];
    Q[[1,1,2]]=0;

    Return[Q]
  ]
```

QuadraticForm2 creates the square $n \times n$ matrix Q for $|\alpha^{(2)}|^2 + \cdots + |\alpha^{(n)}|^2 + t(\alpha^{(1)})^2$ where for each entry the constant and the coefficient of t are separated.

```
In[3]:= QuadraticForm2[f_]:=Module[{Q,a,i,j,k},
   Q=ConstantArray[0,{n,n,2}];
   For[i=1,i≤n,i++, (* loop over embeddings *)
     a=N[NumberFieldIntegralBasis[Root[f,i]],np];
     For[j=1,j≤n,j++, (* loop over variables  xjxk *)
       For[k=1,k≤n,k++,
         If[i==1,
           Q[[j,k,1]]+=Re[a[[j]]*Conjugate[a[[k]]]], (* first
           embedding is multiplied with t *)
           Q[[j,k,2]]+=Re[a[[j]]*Conjugate[a[[k]]]]]
   ]]];
   Return[Q]
  ]
```

QuadraticForm3 creates the square $n \times n$ matrix Q for $t_1|\alpha^{(1)}|^2 + \cdots + t_n|\alpha^{(n)}|^2$ where for each entry the constant and the coefficients of t_i are separated.

```
In[4]:= QuadraticForm3[f_]:=Module[{Q,a,i,j,k},
   Q=ConstantArray[0,{n,n,n}];
   For[i=1,i≤n,i++, (* loop over embeddings *)
     a=N[NumberFieldIntegralBasis[Root[f,i]],np];
       For[j=1,j≤n,j++, (* loop over variables  xjxk *)
         For[k=1,k≤n,k++,
           Q[[j,k,i]]+=Re[a[[j]]*Conjugate[a[[k]]]] (* every
           embedding is multiplied with ti *)
   ]]];
   Return[Q]
  ]
```

Extracts units from small norm elements by dividing elements of the same norm.

```
In[5]:= Units[els_]:=Module[{start,elements,units,subsets,newelem,i,j},
   elements = GatherBy[els,Abs[AlgebraicNumberNorm[#]]&]; (* now
   'elements' contains 'buckets' of elements with the same norm *)
   If[Length[elements] == 0, Return[]];
   units = {};
   (* collect elements of absolute norm 1 *)
   If[Abs[AlgebraicNumberNorm[elements[[1,1]]]]==1, (* does the first
   bucket have absolute norm 1 elements? *)
   (* yes *)
     For[i=1,i≤Length[elements[[1]]],i++,
       If[elements[[1,i]]≠1,
         units=Union[units,{elements[[1,i]]}]] (* the units *)
     ];
     start=2 (* we can start at the second bucket by dividing *)
     ,
     (* no *)
     start=1 (* we should divide elements of the first bucket too *)
   ];

   (* divide two other elements to find units *)
   For[i=start,i≤Length[elements],i++,
```

```
      If[Length[elements[[i]]]>1,
        subsets = Subsets[elements[[i]],{2}];
        (* pick pairs of elements of the same norm *)
        For[j=1,j≤Length[subsets],j++,
          newelem = subsets[[j,1]]/subsets[[j,2]];
          (* divide the elements and check if they
          are algebraic integers *)
          If[AlgebraicIntegerQ[newelem]&&Abs[newelem]≠1,
          units=Union[units,{newelem}]]
    ]]];
    Return[units]
  ]
```

Find a system of independent units from units_.

```
In[6]:= IndependentUnits[us_,f_]:=Module[{units,logunits,independentunits,
        logindependentunits,i,maxrank,relation,x,b,j,d,l,m,newunit,reg,
        lognewunit,newrelation},
          units = Complement[us, NumberFieldRootsOfUnity[Root[f, 1]]];
          (* get rid of roots of unity *)
          units = SortBy[units, Total[Abs[#[[2]]]] &]; (* prioritize elements
          with small coefficients *)
          If[Length[units] == 0, Return[]];
          logunits = LogEmbed[#, f] & /@ units; (* logarithmic embedding *)
          maxrank = MatrixRank[logunits];
          logindependentunits = {logunits[[1]]};
          independentunits = {units[[1]]};
          (* consider new unit *)
          For[i=2,i≤Length[units],i++,
            newunit = units[[i]];
            lognewunit = logunits[[i]];
            (* check if rank increases *)
            If[MatrixRank[logindependentunits]<MatrixRank[Append[
              logindependentunits,logunits[[i]]]],
            (* independent *)
            (* add it to the independent units *)
            AppendTo[independentunits,units[[i]]];
            AppendTo[logindependentunits,logunits[[i]]]
            ,
            (* dependent *)
            (* find relation a1v1+...+auvu+av=0 *)
            relation = NullSpace[Append[logindependentunits,logunits[[i]]]^T]
                  [[1]];
            relation /= Select[relation,Chop[#]≠0&,1][[1]]; (* divide everything
            by a nonzero number *)
            relation = Rationalize[relation,10^-40];
            relation = relation/Apply[GCD,relation]; (* relation is now
            integral *)
            (* now relation = {a1,...,au,a} *)
            If[Or[Abs[relation[[-1]]] == 1, Or @@ (# > 2^50 & /@ relation)],
              Continue[]];
            (* the span won't get bigger if v is a combination of v_1,...,v_u *)
            d = Apply[GCD, Drop[relation, -1]];
            (* d = gcd(a1,...,au) *)
```

```
      If[d > 1, (* replace v by v′ *)
        l, m = l1, m1 /. FindInstance[d*m1 - relation[[-1]]*l1 == 1,
          {l1, m1}, Integers][[1]];
        newrelation = 1/d*Drop[relation, -1];
        relation = Append[newrelation, relation[[-1]]];
        lognewunit = m*lognewunit +
          l(Drop[relation, -1].logindependentunits);
        newunit = newunit^m *
          Product[independentunits[[j]]^(relation[[j]]*l),
            {j, 1, Length[independentunits]}]
      ];
      (* solve a₁b₁+...+aᵤbᵤ=a-1 *)
      x = Table[Unique["x"],{Length[relation]-1}];
      Clear[x];
      vars=Array[x,Length[relation]];
      x[Length[vars]]=-1;
      b = FindInstance[relation.vars==1||relation.vars==-1,
      Drop[vars,-1],Integers];
      b=vars/.b[[1]];
      (* now b = {b₁,...,bᵤ,-1} *)
      (* change basis *)
      For[j=1,j≤Length[logindependentunits],j++,
        logindependentunits[[j]]+=logunits[[i]]*b[[j]];
        independentunits[[j]]*=units[[i]]^b[[j]];
      ]]];
    reg = Null;
    If[MatrixRank[logindependentunits]==r1+r2-1,
     reg = N[Abs[Det[logindependentunits]],6]];
    Print["Number of independent units: ", Length[independentunits],
     "/", r1+r2-1];
    Print["Independent units: ", independentunits];
    Print["Regulator: ", reg]
  ]
```

Returns the logarithmic embedding of x.

```
In[7]:= LogEmbed[x_,f_]:=Module[{lambda,i},
    lambda = ConstantArray[0, r1 + r2 - 1];
    For[i = 1, i ≤ r1 + r2 - 1, i++,
     If[i ≤ r1,
        (* need only one complex conjugate *)
      lambda[[i]] = Log[Abs[x[[2]].N[Root[f, i], np]^
       Array[# &, r1 + 2 r2, 0]]],
      lambda[[i]] = Log[Abs[x[[2]].N[Root[f, r1 + 2*(i - r1) - 1], np]^
       Array[# &, r1 + 2 r2, 0]]]
     ];
     If[i > r1, lambda[[i]] *= 2];
    ];
    Return[lambda]
  ]
```

GeodesicLLL performs the Geodesic algorithm on quadratic form Q. The number of steps it should do is determined by the variable steps_.

```
In[8]:= GeodesicLLL[Q_,steps_]:=Module[{stepcounter,norms,units,i},
          stepcounter = 0;
          A = Initialize[N[Q,np]];
          InitialReduction[];
          elements = {};
          While[!complete,
           stepcounter++;
           ApproximationStep[];
           For[i=1, i≤Length[transform[[1,All]]],i++,
            AppendTo[elements, transform[[All,i]].b] (* collect elements from
             transform matrix *)
           ];
          If[Accuracy[t]≤0||Accuracy[A]≤0||stepcounter>steps, complete=True];
          ];
          Return[elements]
        ]
```

In Initialize we compute the initial values of B_{ij} and C_i. You can recall the value B_{ij} with B[[i,j,1]]*t+B[[i,j,2]] and the value of C_i with C[[i,1]]*t+C[[i,2]]. This Module returns {B,C}. Also we define some global variables which are used in several Modules.

```
In[9]:= Initialize[Q_]:=Module[{n,B,C},
          n=Length[Q];
          t=1;
          transform=IdentityMatrix[n];(*this matrix keeps hold of the
          transformations *)
          stepcounter=0;(*counts the number of approximation steps*)
          complete=False;(*becomes true when the process has to stop.*)
          Return[Determinants[Q]]
        ]
```

Creates matrix B and array C consisting of determinants.

```
In[10]:= Determinants[Q_]:=Module[{B,C,i,j},
          B=ConstantArray[0,{n,n,2}];
          C=ConstantArray[0,{n-1,2}];
          For[i=1,i<n,i++,
           For[j=i,j≤n,j++,
            B[[i,j]]=MakeBij[Q,i,j];
           ];
           C[[i]]=MakeCi[Q,i];
          ];
          B[[n,n]]=MakeBij[Q,n,n];
          Return[{B,C}]
        ]
```

Finds the determinant B_{ij} to put in matrix B.

```
In[11]:= MakeBij[Q_,i_,j_]:=Module[{B,k,l},
          Clear[p];
          B=ConstantArray[0,{i,i}];
          For[k=1,k≤i,k++,
```

```
     For[l=1,l≤i-1,l++,
      B[[l,k]]=Q[[l,k]][[2]]+p*Q[[l,k]][[1]];
     ];
     B[[i,k]]=Q[[j,k]][[2]]+p*Q[[j,k]][[1]];
    ];
    Return[{Coefficient[Det[B],p,1],Coefficient[Det[B],p,0]}]
  ]
```

Finds the determinant C_i to put in matrix C.

```
In[12]:= MakeCi[Q_,ii_]:=Module[{i,j,C,k,l},
    Clear[p];
    i=ii+1;
    j=ii+1;
    C=ConstantArray[0,{i,i}];
    For[k=1,k≤i,k++,
     For[l=1,l≤i-1,l++,
      C[[k,l]]=Q[[k,l]][[2]]+p*Q[[k,l]][[1]];
     ];
     C[[k,i]]=Q[[k,j]][[2]]+p*Q[[k,j]][[1]];
    ];
    C=Drop[C,{ii}]; (* remove i'th row and column *)
    C=Drop[C,{}, {ii}];
    Return[{Coefficient[Det[C],p,1],Coefficient[Det[C],p,0]}]
  ]
```

We perform the first shifts to make sure that all μ_{ij} lie in the interval [-0.5;0.5], thus we check all the inequalities $2|B_{1j}| \leq B_{11}$ for $t = 1$.

```
In[13]:= InitialReduction[]:=Module[{B11,B1j,mid,a,j},
    B11=A[[1,1,1]][[1]]*t+A[[1,1,1]][[2]]; (*Compute value of B_11*)
    For[j=2,j≤n,j++,
     B1j=A[[1,1,j]][[1]]*t+A[[1,1,j]][[2]];
     If[2*Abs[B1j]>B11, (*If the inequality is not met, we need to
      perform a shift*)
      mid= B1j/B11;(*We use mid to calculate with which
       value we have to shift*)
      a=Floor[0.5-mid];
      Shift[1,j,a](*We perform the shift x_1→x_1+a x_j*)
    ]]
  ]
```

This Module performs one approximation step and calls for the corresponding critical shift or swap. It calls for Reductionstep2 and Reductionstep3 to make the new form LLL-reduced.

```
In[14]:= ApproximationStep[]:=Module[{plan,a,i,j,mij},
    plan=MakePlan[];
    If[!complete, (*While making the plan, complete can become True in
     the ComputeInt Module, then we skip the next part and the process
     stops*)
     If[plan[[1]], (*This means we have to shift*)
      a=-1; (*a decides whether we shift with -1 or 1*)
      i=plan[[2]];
      j=plan[[3]];
      mij=(A[[1,i,j,1]]*t+A[[1,i,j,2]])/(A[[1,i,i,1]]*t+A[[1,i,i,2]]);
```

```
      If[mij<0,a=1];
      Shift[i,j,a];
      If[j==i+1,ReductionStep2[]];
      ReductionStep3[];
      , (* we will swap *)
      Swap[plan[[2]]];
      ReductionStep2[];
      ReductionStep3[] (*For partial reduction we can skip this step*)
    ]]
  ]
```

This Module uses the list of intervals which is the output of ComputeInterval[]. We calculate the next value of t and return a plan. This plan is a list of the form {shift,i,j}. The variable shift is a boolean, if it is true we need to perform a shift, otherwise a swap. The i and j tell us which shift or swap (if $j = -1$) we have to perform.

```
In[15]:= MakePlan[]:=Module[{int,shift,i,j,upperbound,k},
    int=ComputeInterval[];
    shift=True;
    upperbound=Max[int[[1,3]]];
    i=int[[1,1]];
    j=int[[1,2]];
    For[k=1,k≤Length[int],k++,
     If[Max[int[[k,3]]]<upperbound,
      upperbound=Max[int[[k,3]]];
      i=int[[k,1]];
      j=int[[k,2]]]];
    If[j==-1,shift=False];
    t=upperbound; (*assigns the new value for t*)
    Return[{shift,i,j}]
  ]
```

This Module computes for each inequality the intervals for t for which the inequality is met. It returns an array with elements of the form {int, i,j} where int is the interval and i and j denote the corresponding inequality. If $j = -1$ the interval belongs to the inequality $\omega B_{ii} \leq C_i$ (so we have to swap), otherwise the interval belongs to the inequality $2|B_{ij}| < B_{ii}$ (so we have to shift).

```
In[16]:= ComputeInterval[]:=Module[{Int,Bij,Bii,int1, int2, int,Ci,i,j},
    Int={};
    (*In the following for-loop we check the inequalities 2|Bij|<Bii*)
    For[i=1,i≤ n,i++,
     For[j=i+1,j≤n,j++,
      Bij=2A[[1,i,j]];
      Bii=A[[1,i,i]];
      If[Bij[[1]]-Bii[[1]]==0||-Bii[[1]]-Bij[[1]]==0,
       complete=True];
      If[!complete, (*We have to check both 2Bij<Bii and -Bii<2Bij.
       Then we compute their intersection.*)
       If[Bij[[1]]-Bii[[1]]>0,
        int1 = Interval[{0,(Bii[[2]]-Bij[[2]])/(Bij[[1]]-Bii[[1]])}],
        int1 = Interval[{(Bii[[2]]-Bij[[2]])/(Bij[[1]]-Bii[[1]]),
         Infinity}]];
       If[-Bii[[1]]-Bij[[1]]>0,
        int2 = Interval[{0,(Bij[[2]]+Bii[[2]])/(-Bii[[1]]-Bij[[1]])}],
```

```
         int2 = Interval[{(Bij[[2]]+Bii[[2]])/(-Bii[[1]]-Bij[[1]]),
          Infinity}]];
        int = IntervalIntersection[int1,int2];
        AppendTo[Int,{i,j,int}]
     ]];
     (*In the following for-loop we check the inequalities 3/4Bii ≤ Ci*)
      For[i=1,i≤n-1,i++,
       Bii=3/4A[[1,i,i]];
       Ci=A[[2,i]];
       If[Bii[[1]]-Ci[[1]]>0,int=Interval[{0,(Ci[[2]]-Bii[[2]])/
        (Bii[[1]]-Ci[[1]])}],
        int = Interval[{(Ci[[2]]-Bii[[2]])/(Bii[[1]]-Ci[[1]]),
         Infinity}]];
       AppendTo[Int,{i,-1,int}]]];
        (*The -1 indicates that we have to perform a swap*)
     Return[Int]
   ]
```

Performs the shift $x_r \to x_r + a x_s$.

```
In[17]:= Shift[rr_,ss_,aa_]:=Module[{r,s,a,B,C,i},
     r=rr;
     s=ss;
     a=aa;
     (*Update transformmatrix*)
     For[i=1,i≤n, i++,
      transform[[i,s]]+=a transform[[i,r]]];
      (*Update determinants*)
     If[r==s-1,
      A[[2,r]]+=2 a A[[1,r,s]]+a^2 A[[1,r,r]]];
     For[i=1,i≤r,i++,
      A[[1,i,s]]+=a*A[[1,i,r]]]
   ]
```

Performs the swap $x_r \leftrightarrow x_{r+1}$.

```
In[18]:= Swap[rr_]:=Module[{r,old,Br1r1,Br2r2,Brijb,Bjj1,Bjj,mjj1,shift,
   oldtrans,i,j},
     r=rr;
     (*Update the transformmatrix*)
     For[i=1,i≤n, i++,
      oldtrans=transform[[i,r]];
      transform[[i,r]]=transform[[i,r+1]];
      transform[[i,r+1]]=oldtrans;
     ];
     (*Update determinants*)

     (*update rule 4 and 5*)
     For[j=r+2,j≤n,j++,
      old=A[[1,r,j]];
      If[r==1,Br1r1={0,1},Br1r1=A[[1,r-1,r-1]]];
      If[A[[1,r,r,2]]==0,A[[1,r,j,2]]=(A[[1,r,r+1,2]]*A[[1,r,j,1]]+
       A[[1,r,r+1,1]]*A[[1,r,j,2]]+Br1r1[[2]]*A[[1,r+1,j,1]]+Br1r1[[1]]*
       A[[1,r+1,j,2]])/A[[1,r,r,1]];
```

```
   A[[1,r,j,1]]=(A[[1,r,r+1,1]]*A[[1,r,j,1]]+Br1r1[[1]]*
    A[[1,r+1,j,1]])/(A[[1,r,r,1]]);
   A[[1,r+1,j,2]]=(A[[1,r+1,r+1,2]]*old[[1]]+A[[1,r+1,r+1,1]]*old[[2]]-
    A[[1,r,r+1,2]]*A[[1,r+1,j,1]]-A[[1,r,r+1,1]]*A[[1,r+1,j,2]])/
     (A[[1,r,r,1]]);
   A[[1,r+1,j,1]]=(A[[1,r+1,r+1,1]]*old[[1]]-A[[1,r,r+1,1]]*
    A[[1,r+1,j,1]])/(A[[1,r,r,1]]);,
   (*else*)
   A[[1,r,j,2]]=(A[[1,r,r+1,2]]*A[[1,r,j,2]]+Br1r1[[2]]*
    A[[1,r+1,j,2]])/A[[1,r,r,2]];
   A[[1,r,j,1]]=(A[[1,r,r+1,2]]*A[[1,r,j,1]]+A[[1,r,r+1,1]]*old[[2]]+
    Br1r1[[2]]*A[[1,r+1,j,1]]+Br1r1[[1]]*
    A[[1,r+1,j,2]]-A[[1,r,j,2]]*A[[1,r,r,1]])/A[[1,r,r,2]];
   Brijb=A[[1,r+1,j,2]];
   A[[1,r+1,j,2]]=(A[[1,r+1,r+1,2]]*old[[2]]-A[[1,r,r+1,2]]*
    A[[1,r+1,j,2]])/A[[1,r,r,2]];
   A[[1,r+1,j,1]]=(A[[1,r+1,r+1,2]]*old[[1]]+A[[1,r+1,r+1,1]]*
    old[[2]]-A[[1,r,r+1,2]]*A[[1,r+1,j,1]]-A[[1,r,r+1,1]]*Brijb-
     A[[1,r+1,j,2]]*A[[1,r,r,1]])/A[[1,r,r,2]]]]];
  (*update rule 8*)
  If[r>1,
   If[r==2,Br2r2={0,1},Br2r2=A[[1,r-2,r-2]]];
   If[A[[1,r-1,r-1,2]]==0,
    A[[2,r-1,2]]=(Br2r2[[2]]*A[[2,r,1]]+Br2r2[[1]]*A[[2,r,2]]+
     2*A[[1,r-1,r+1,2]]*A[[1,r-1,r+1,1]])/A[[1,r-1,r-1,1]];
    A[[2,r-1,1]]=(Br2r2[[1]]*A[[2,r,1]]+A[[1,r-1,r+1,1]]*
     A[[1,r-1,r+1,1]])/A[[1,r-1,r-1,1]];,(*else*)
    A[[2,r-1,2]]=(Br2r2[[2]]*A[[2,r,2]]+A[[1,r-1,r+1,2]]*
     A[[1,r-1,r+1,2]])/A[[1,r-1,r-1,2]];

    A[[2,r-1,1]]=(Br2r2[[2]]*A[[2,r,1]]+Br2r2[[1]]*
     A[[2,r,2]]+2*A[[1,r-1,r+1,2]]*A[[1,r-1,r+1,1]]-A[[2,r-1,2]]*
      A[[1,r-1,r-1,1]])/A[[1,r-1,r-1,2]];
  ]];
  (*update rule 2 and 3*)
  For[i=1,i<r,i++,
   old=A[[1,i,r]];
   A[[1,i,r]]=A[[1,i,r+1]];
   A[[1,i,r+1]]=old];
  (*update rule 9*)
  If[r<n-1,
   If[A[[1,r+1,r+1,2]]==0,A[[2,r+1,2]]=(A[[1,r+2,r+2,2]]*
    A[[2,r,1]]+A[[1,r+2,r+2,1]]*A[[2,r,2]]+2*A[[1,r+1,r+2,2]]*
     A[[1,r+1,r+2,1]])/A[[1,r+1,r+1,1]];
    A[[2,r+1,1]]=(A[[1,r+2,r+2,1]]*A[[2,r,1]]+A[[1,r+1,r+2,1]]*
     A[[1,r+1,r+2,1]])/(A[[1,r+1,r+1,1]]);
   ,(*else*)
   A[[2,r+1,2]]=(A[[1,r+2,r+2,2]]*A[[2,r,2]]+A[[1,r+1,r+2,2]]*
    A[[1,r+1,r+2,2]])/A[[1,r+1,r+1,2]];
   A[[2,r+1,1]]=(A[[1,r+2,r+2,2]]*A[[2,r,1]]+A[[1,r+2,r+2,1]]*
    A[[2,r,2]]+2*A[[1,r+1,r+2,2]]*A[[1,r+1,r+2,1]]-A[[2,r+1,2]]*
     A[[1,r+1,r+1,1]])/A[[1,r+1,r+1,2]]];
```

```
    ];
    (*update rule 1 and 7*)
    old=A[[1,r,r]];
    A[[1,r,r]]=A[[2,r]];
    A[[2,r]]=old;

    (*After a swap,we always need to check for possible shifts,
     hence it is included in the Swap method.*)
    For[j=(r-1),j≤(r+1),j++,
     If[(j<1||j>n-1),Continue[]];
    Bjj1=A[[1,j,j+1,1]]*t+A[[1,j,j+1,2]];
    Bjj=A[[1,j,j,1]]*t+A[[1,j,j,2]];
    If[2*Abs[Bjj1]>Bjj,
     mjj1=Bjj1/Bjj;
     shift=Floor[0.5-mjj1];
     Shift[j,j+1,shift];
    ]]
  ]
```

After computing the new t and performing the corresponding critical shift or swap, we sometimes have to perform more shifts or swaps to make the new form LLL-reduced. Reductionstep2 executes the required swaps, Reductionstep3 executes the required shifts.

```
In[19]:= ReductionStep2[]:=Module[{Bii,Ci,i},
    For[i=1,i<n,i++,
     Bii=A[[1,i,i,1]]*t+A[[1,i,i,2]];
     Ci=A[[2,i,1]]*t+A[[2,i,2]];
     If[3/4Bii>Ci,
      Swap[i];
      i=0;]]
  ]
```

```
In[20]:= ReductionStep3[]:=Module[{Bij,Bii,mij,shift,i,j},
    For[j=n,j>1,j--,
     For[i=j-1,i>0,i--,
      Bij=A[[1,i,j,1]]*t+A[[1,i,j,2]];
      Bii=A[[1,i,i,1]]*t+A[[1,i,i,2]];
      If[2*Abs[Bij]>Bii,
       mij=Bij/Bii;
       shift=Floor[0.5-mij];
       Shift[i,j,shift]
    ]]]
  ]
```

Directions performs the 'reduction in directions' algorithm on the form Q, where steps_ is the number of steps and size_ is the size of each vector.

```
In[21]:= Directions[Q_,steps_,size_]:=Module[{direction,directions,j,i},
    elements={}; (* global 'association', the found elements grouped
    together by their norms *)
    (* generate directions to reduce in *)
    directions={};
    If[steps>0,
     directions=RandomVectors[r1+r2,steps,size], (* generate random
      directions with powers of two *)
```

```
    directions=SystematicVectors[r1+r2,size] (* systematically
     generate directions with powers of two *)
    ];
    (* reduce in generated directions *)
    For[i=1,i≤Length[directions],i++,
     (* we want direction[i]=direction[i+r₂] for i>r₁
     (the complex conjugate direction gets the same power) *)
     direction=Array[1&,n];
     For[j=1,j≤r1,j++, (* real *)
      direction[[j]]=directions[[i,j]]];
     For[j=1,j≤r2,j++, (* complex *)
      direction[[r1+2*j-1]]=directions[[i,r1+j]];
      direction[[r1+2*j]]=directions[[i,r1+j]]];
     ReduceDirection[Q,direction] (* reduce and collect elements *)
    ];
    (* now 'elements' contains all elements found by reducing *)
    Return[elements];
  ]
```

Create all vectors of the form $\{2^{i_1}, 2^{i_2}, \ldots, 2^{i_n}\}$ with n = size_ and $i_j <$ power_.

```
In[22]:= SystematicVectors[size_,power_]:=Module[{res2,i,j},
    (* get the digits of a number converted to base power_, padded
     with zeros so it has the right size *)
    Return[2^(2*
     Table[If[MemberQ[IntegerDigits[i, power, size], 0],
      IntegerDigits[i, power, size], ## &[]], i, 0, power^size - 1])];
  ]
```

Returns number random vectors of length size with powers of 2 smaller than power.

```
In[23]:= RandomVectors[size_,power_,number_]:=Module[{res,arr,a,i},
    res=Array[1,number]; (* array of directions *)
    For[i=1,i≤number,i++,
     a=RandomInteger[{1,size}]; (* one position gets a 1 *)
     arr=RandomInteger[power,size]; (* direction *)
     arr[[a]]=0;
     res[[i]]=2^arr;
    ];
    Return[res];
  ]
```

LLL-reduce the quadratic form Q in "direction" ts.

```
In[24]:= ReduceDirection[Q_,ts_]:=Module[{done,Q1,i}, (* LLL reduce in
    "direction" ts *)
    transform=IdentityMatrix[n]; (* keep track of the
    transformation *)
    Q1=FillTs[Q,ts]; (* numerical matrix with ts filled in for t,
    obtaining a not necessarily reduced matrix *)
    A=Determinants2[Q1] ; (* matrix containing Bᵢⱼ and matrix
    containing Cᵢⱼ *)
    (* we reduce this matrix and keep
    track of the shifts and swaps in transform *)
    done=False; (* check if the result is reduced *)
```

```
        While[done==False,
         done=LLLReduce[];
          (* perform a reduction step *)
        ];
        (* collect the units *)
        For[i=1,i≤Length[transform[[1,All]]],i++,
         AppendTo[elements,transform[[All,i]].b]]
      ]
```

Creates matrix B and array C consisting of determinants.

```
In[25]:= Determinants2[QQ_]:=Module[{Q,B,C,i,j},
           Q=QQ;
           B=ConstantArray[0,{n,n}];
           C=ConstantArray[0,{n-1}];
           For[i=1,i<n,i++,
            For[j=i,j≤n,j++,
             B[[i,j]]=MakeBij2[Q,i,j];
            ];
            C[[i]]=MakeCi2[Q,i];
           ];
           B[[n,n]]=MakeBij2[Q,n,n];
           Return[{B,C}]
         ]
```

Creates B_{ij} for matrix B.

```
In[26]:= MakeBij2[QQ_,ii_,jj_]:=Module[{Q,i,j,B,k,l},
           Q=QQ;
           i=ii;
           j=jj;
           B=ConstantArray[0,{i,i}];
           For[k=1,k≤i,k++,
            For[l=1,l≤i-1,l++,
             B[[l,k]]=Q[[l,k]];
            ];
            B[[i,k]]=Q[[j,k]];
           ];
           Return[Det[B]]
         ]
```

Creates C_i for matrix C.

```
In[27]:= MakeCi2[QQ_,ii_]:=Module[{Q,i,j,C,k,l},
           Q=QQ;
           i=ii+1;
           j=ii+1;
           C=ConstantArray[0,{i,i}];
           For[k=1,k≤i,k++,
            For[l=1,l≤i-1,l++,
             C[[k,l]]=Q[[k,l]];
            ];
            C[[k,i]]=Q[[k,j]];
           ];
           C=Drop[C,{ii}]; (* remove i'th row and column *)
```

```
    C=Drop[C,{}, {ii}];
    Return[Det[C]];
  ]
```

Numerical matrix with ts_i filled in for t_i, obtaining a not necessarily reduced matrix A.

```
In[28]:= FillTs[B_,ts_]:=Module[{A,i,j},
    A=ConstantArray[0,{n,n}];
    For[i=1,i≤n,i++,
     For[j=1,j≤n,j++,
      A[[i,j]]=B[[i,j]].ts;
    ]];
    Return[A];
  ]
```

LLLReduce matrix Q.

```
In[29]:= LLLReduce[]:=Module[{i,j,mid,a,done},
    done=True;
    (* check if 2|Bij|≤Bij is satisfied *)
    For[i=1,i<n,i++,
     For[j=i+1,j≤n,j++,
      If[2*Abs[A[[1,i,j]]]>A[[1,i,i]],
       mid=A[[1,i,j]]/A[[1,i,i]];
       a=Floor[0.5-mid];
       Shift2[i,j,a]; (* if not, we shift *)
       done=False;
    ]]];
    (* check if 2Bii≤Ci is satisfied *)
    For[i=1,i<n,i++,
     If[(3/4)*A[[1,i,i]]>A[[2,i]],
      Swap2[i];
      done=False;
     ]
    ];
    Return[done]
  ]
```

Performs the shift $x_r \to x_r + a x_s$.

```
In[30]:= Shift2[rr_,ss_,aa_]:=Module[{r,s,i,j,a},
    r=rr;
    s=ss;
    a=aa;
    (*Update transformmatrix*)
    For[i=1,i≤n, i++,
     transform[[i,s]]+=a transform[[i,r]]];
    (*Update determinants*)
    If[r==s-1,
     A[[2,r]]+=2 a A[[1,r,s]]+a^2 A[[1,r,r]]];
    For[i=1,i≤r,i++,
     A[[1,i,s]]+=a*A[[1,i,r]]]
  ]
```

Performs the swap $x_r \leftrightarrow x_{r+1}$.

```
In[31]:= Swap2[rr_]:=Module[{r,old,Br1r1,Br2r2,Brijb,Bjj1,Bjj,mjj1,shift,
  oldtrans,i,j},
  r=rr;
  (*Update the transformmatrix*)
  For[i=1,i≤n, i++,
   oldtrans=transform[[i,r]];
   transform[[i,r]]=transform[[i,r+1]];
   transform[[i,r+1]]=oldtrans;
  ];
  (*Update determinants*)
  (*update rule 4 and 5*)
  For[j=r+2,j≤n,j++,
   old=A[[1,r,j]];
   A[[1,r,j]]=(A[[1,r,r+1]]*A[[1,r,j]]+A[[1,r-1,r-1]]*A[[1,r+1,j]])/
    (A[[1,r,r]]);
   A[[1,r+1,j]]=(A[[1,r+1,r+1]]*old-A[[1,r,r+1]]*A[[1,r+1,j]])/
    (A[[1,r,r]]);
  ];
  (*update rule 8*)
  If[r>1,
   Br2r2=A[[1,r-2,r-2]];
   A[[2,r-1]]=(Br2r2*A[[2,r]]+A[[1,r-1,r+1]]*A[[1,r-1,r+1]])/
    A[[1,r-1,r-1]]
  ];
  (*update rule 2 and 3*)
  For[i=1,i<r,i++,
   old=A[[1,i,r]];
   A[[1,i,r]]=A[[1,i,r+1]];
   A[[1,i,r+1]]=old
  ];
  (*update rule 9*)
  If[r<n-1,
   A[[2,r+1]]=(A[[1,r+2,r+2]]*A[[2,r]]+A[[1,r+1,r+2]]*A[[1,r+1,r+2]])/
    A[[1,r+1,r+1]];
  ];
  (*update rule 1 and 7*)
  old=A[[1,r,r]];
  A[[1,r,r]]=A[[2,r]];
  A[[2,r]]=old;

  (*After a swap, we always need to check for possible shifts, hence
   it is included in the Swap method.*)
  For[j=(r-1),j≤(r+1),j++,
   If[(j<1||j>n-1),Continue[]];
   Bjj1=A[[1,j,j+1]];
   Bjj=A[[1,j,j]];
   If[2*Abs[Bjj1]>Bjj,
    mjj1=Bjj1/Bjj;
    shift=Floor[0.5-mjj1];
    Shift2[j,j+1,shift];
  ]]
]
```