TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Fachbereich Mathematik

Linear Algebra over Finitely Generated Fields and Rings

Mannaperuma Herath Mudiyanselage,
Jayantha Suranimalee

1. Gutachter: Prof. Dr. Claus Fieker
2. Gutachter: Prof. Dr. Mohamed Barakat

Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Datum der Disputation: 29. April 2021

D386

Linear Algebra over Finitely Generated Fields and Rings

Mannaperuma Herath Mudiyanselage,
Jayantha Suranimalee

1. Reviewer: Prof. Dr. Claus Fieker
2. Reviewer: Prof. Dr. Mohamed Barakat

Accepted thesis (Dissertation)
of the Department of Mathematics, of the University of Kaiserslautern
for the award of the academic degree
Doctor of the natural sciences

Date of the Disputation: 29. April 2021

Zusammenfassung

Lineare Algebra und Arithmetik von Polynomen bilden die Grundlage von Computeralgebra. Die letzten 20 Jahre hinweg haben sich die Algorithmen stetig weiter verbessert, so dass die aktuellen Algorithmen für Matrixinvertierung, zum Lösen linear Systeme und zur Determinantenberechugn theoretisch nicht mal eine kubische Laufzeit besitzen. In dieser Arbeit werden schnelle und praktische Algorithmen für klassicher Probleme der linearen Algebra über Zahlkörpern und Polynomringe vorgestellt. Als Zahlkörper bezeichnen wir eine endliche Erweiterung des Körpers der raionalen Zahlen.

Eine grundlegende Herausforderung im symbolischen Rechnen ist die Koeffizientenexplosion: Die Bit-Größe von Zwischenergebnissen kann im Verhältnis zur Größe der Ein- und Ausgabe um ein vielfaches anwachsen. Die Standardstrategie, um diesem Effekt beizukommen, ist statt die Zahlen direkt zu berechnen, die Berechnung modulo gewisser andere Zahlen durchzuführen unter Verwendung vom Chinesischen Restsatz oder einer Variante der Newton-Hensel Lifting. Im finalen Schritt dieser Algorithmen werden diese Methoden üblicherweise kombiniert mit Rekonstruktionsverfahren wie beispielsweise der rationalen Rekonstruktion, um ein ganzzahliches Ergebnis in ein rationales Ergebnis umzuwandeln. In dieser Arbeit stellen wir Rekonstruktionsverfahren über Zahlkörper vor mit einem schnellen und simplen Vektor-Rekonstrutkionsalgorithmus.

Die gängige Methode für die Berechnung der Determinante ganzzahliger Matritzen geht zurück auf Storjohann [PS13]. Beim Verallgemeinern diese Methode auf Zahlkörper tat sich das Problem auf, dass der durch die Zeilen der Zahlkörper Matrix generierte Module nicht länger freie ist, und Storjohanns Methode deshalb nicht mehr anwendbar ist. zu betrachten waren wie bislang in Storjohann's Theorie. Um dieser Schwierigkeit Herr zu werden greifen wir auf die Theorie von Pseudo-Matritzen (in [Coh96]) zu. Im Zuge dessen bedurfte es einer Verallgemeinerung von unimodularen Zertifizierungsmethoden für Pseudomatrizen: Ähnlich wie im ganzzahligen Fall überprüfen wir, ob die Determinante von gegebenen Pseudomatrizen eine Einheit ist, indem wir mit Hilfe von höherer Ordnungen lifting die Ganzzahligkeit des zugehörigen dualen Module testen.

Ein zentraler Algorithmus der linearen Algebra ist der Dixon Solver [Dix82] zum Lösen linearer system. Traditionell wird dieser Algorithmus lediglich für quadratische Systeme mit eindeutiger Lösung verwendet. Wir verallgemeinern den Dixon Algorithmus für nicht quadratische lineare Systeme. Da dann das Ergebnis nicht eindeutig ist, benutzen wir eine Basis des Kerns, um die Lösung zu normalisieren. Die Implementierung umfasst einen Algorithmus zur schnellen Berechnung von Kernen, welcher in einer erweiterten Form die reduzierte Zeilenstufenform einer Matrix über Ganzzahlen und Zahlkörpern ausrechnen kann.

Die schnelle Implementierung zur Berechnung charakteristischer Polynome und Minimalpolynome über Zahlkörper verwendet einen modularen, auf dem Chinesischen Restsatz aufbauenden Ansatz. Schließlich präsentieren wir eine Verallgemeinerung Storjohann's Algorithmus zur Determinantenberechnung auf Polynomringe über endlichen Körpern, Teile derer aus Rekonstruktionen und unimodularen Zertifikaten bestehen. Das in diesem Fall auftretende Anwachsen der Grade bekommen wir in den Griff mit Hilfe von Hebungen höherer Ordnungen im unimodularen Zertifizierungsalgorithmus. Erfolgreich verwendeten wir einen halb-ggT Ansatz für die Optimierung der Rekonstruktion rationaler Polynome.

Abstract

Linear algebra, together with polynomial arithmetic, is the foundation of computer algebra. The algorithms have improved over the last 20 years, and the current state of the art algorithms for matrix inverse, solution of a linear system and determinants have a theoretical sub-cubic complexity. This thesis presents fast and practical algorithms for some classical problems in linear algebra over number fields and polynomial rings. Here, a number field is a finite extension of the field of rational numbers, and the polynomial rings we considered in this thesis are over finite fields.

One of the key problems of symbolic computation is intermediate coefficient swell: the bit length of intermediate results can grow during the computation compared to those in the input and output. The standard strategy to overcome this is not to compute the number directly but to compute it modulo some other numbers, using either the Chinese remainder theorem (CRT) or a variation of Newton-Hensel lifting. Often, the final step of these algorithms is combined with reconstruction methods such as rational reconstruction to convert the integral result into the rational solution. Here, we present reconstruction methods over number fields with a fast and simple vector-reconstruction algorithm.

The state of the art method for computing the determinant over integers is due to Storjohann [PS13]. When generalizing his method over number field, we encountered the problem that modules generated by the rows of a matrix over number fields are in general not free, thus Strojohann's method cannot be used directly. Therefore, we have used the theory of pseudo-matrices (in [Coh96]) to overcome this problem. As a sub-problem of this application, we generalized a unimodular certification method for pseudo-matrices: similar to the integer case, we check whether the determinant of the given pseudo matrix is a unit by testing the integrality of the corresponding dual module using higher-order lifting.

One of the main algorithms in linear algebra is the Dixon solver for linear system solving due to Dixon [Dix82]. Traditionally this algorithm is used only for square systems having a unique solution. Here we generalized Dixon algorithm for non-square linear system solving. As the solution is not unique, we have used a basis of the kernel to normalize the solution. The implementation is accompanied by a fast kernel computation algorithm that also extends to compute the reduced-row-echelon form of a matrix over integers and number fields.

The fast implementations for computing the characteristic polynomial and minimal polynomial over number fields use the CRT-based modular approach. Finally, we extended Storjohann's determinant computation algorithm over polynomial ring over finite fields, with its sub-algorithms for reconstructions and unimodular certification. In this case, we face the problem of intermediate degree swell. To avoid this phenomenon, we used higher-order lifting techniques in the unimodular certification algorithm. We have successfully used the half-gcd approach to optimize the rational polynomial reconstruction.

Keywords: determinant, kernel, non square linear system solving, unimodular certification, characteristic polynomial, minimal polynomial, reconstructions, number fields

Acknowlegements

I would like to express my deepest gratitude to my supervisor Prof. Dr. Claus Fieker, for his continuous support, patient guidance, advice and many valuable comments, from which I have learned so much. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. Thank you for offering me the opportunity to work in AGAG research group at the TUK.

I would also like to thank all the members of staff at Technical University Kaiserslauten who helped me for the success of this work. Specially, I would like to thank Dr. Tommy Hofmann, Dr. Daniel Schultz, Dr. William Hart and Dr. Carlo Sircana for their constant support and friendly advices during this research work.

Moreover, I owe special thanks to Prof. Dr Mohamed Barakat for agreeing to act as a referee of my thesis.

I also take this opportunity to thank my friends, Dr. Michelle Gehringer, Prof. Dr. Caroline Lassueur, Dr. Hans Schönemann, Aslam Ali, Sogo Sanon, Yvonne Weber, Petra Bäsel, Jessica Borsche, Ingrid Dietz, Nilusha Gamage, Suresh Nisansala, Yashika Jayathunge, Firoozeh Aga, Vishnupriya Anupindi and Chetana Viswanatha for their help given to me in numerous ways.

I am deeply indebted to my family. To my parents who encourage me in my life, to my sister, Ajantha Samanmalee and husband, Damith Nanayakkara who share all my responsibilities and encourage me in every way.

Completing this work would have been all the more difficult were it not for the support provided by Department of Mathematics, University of Colombo (UoC). I acknowledge with deep gratitude all members in the UoC, who helped me by providing study leave and also the authority of the UoC for allowing me to do this PhD program. My special thank and appreciation go to Dr. Jagath Wijerathna, Prof. Dr. Sanjeewa Perera and Mrs. L. Edussuriya for their constant support and guidance throughout my academic career.

Last but not least, I would like to thank Prof. Dr. Wolfram Decker, Dr. Michael Kunte, Dr. Falk Triebsch, Prof. Dr. Gunter Malle, Dr. habil. Christoph Lossen and the University Grants Commission in Sri Lanka for providing me source of funding as scholarships and job opportunities, which allowed me to undertake my Master's and PhD programs at the TUK.

Contents

List of Algorithms

List of Tables

Abbreviations, Symbols and Nomenclature

In this thesis, we have denoted the integers, rational numbers, the real numbers and the complex numbers respectively by $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$. Most often, we denote number fields by K, finite fields by $\mathbb{F}$ and, all the fields in general by F. The maximal order of the number field K is denoted by $\mathcal{O}$.

For a ring R and two integers $m, n \in \mathbb{Z}_{\geq 1}$, the set of all $n \times m$ matrices over R is denoted by $R^{n\times m}$. A matrix $A \in R^{n\times m}$ can be represented with respect to its entries as $A = [a_{ij}]_{i,j}$ for $1 \leq i \leq n$, $1 \leq j \leq m$, where a_{ij} is the matrix entry of i-th row and j-th column. By $O_{n\times m}$ we denote the zero-matrix in $R^{n\times m}$, and I_n denotes the identity matrix in $R^{n\times n}$. A diagonal matrix $A = [a_{ij}]_{i,j} \in R^{n\times n}$ with $a_{i,j} = 0$ for all i, j with $i \neq j$ is denoted by $\mathrm{diag}(a_{1,1}, a_{2,2}, \ldots, a_{n,n})$. The determinant of any square matrix A is denoted by $\det(A)$. The transpose matrix of A is given by A^t. Given two matrices $A \in R^{n\times m}$ and $b \in R^{n\times r}$ the concatenation of the two matrices is given by $[A\ b]$.

$|a|$ denotes the absolute value of $a \in \mathbb{Z}$ or $a \in \mathbb{C}$. If $A \in \mathbb{Z}^{n\times m}$ or $A \in \mathbb{C}^{n\times m}$, the largest absolute value of matrix entries is denoted by $\|A\|_\infty = \max_{ij} |A_{ij}|$. Let $\|\alpha\|_\varphi$ denote the largest absolute value of coefficients of $\alpha \in K$. If $A \in K^{n\times m}$, the largest absolute value of coefficients of the matrix entries of A is denoted by $\|A\|_\varphi$.

For two integers $a, b \in \mathbb{Z}$, by $a \perp b$ we denotes that two integers are relatively prime.

Let $\mathfrak{a}$ be an ideal of $\mathcal{O}$. A basis of $\mathfrak{a}$ is given as $\langle a_1, \ldots, a_d \rangle$ for some $a_1, \ldots, a_d \in \mathcal{O}$ and d is the degree of K. The norm of $\mathfrak{a}$ is denoted by $\mathrm{norm}(\mathfrak{a})$ or $N(\mathfrak{a})$. For $v \in \mathbb{R}^n$ or $\mathbb{Z}^n$, the Euclidean norm of v is given by $\|v\|$.

To simplify the presentations of complexity results, we use soft-Oh notation $O^\sim$: for functions $f, g : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ we have $f \in O^\sim(g)$ if and only if there exists $k \in \mathbb{Z}_{>0}$ such that $f \in O(g(\log(g))^k)$.

Introduction

Linear algebra is central to almost all areas of mathematics. Group theory, in particular representations of finite groups is based on linear algebra. Moreover, almost all structural computations involving number fields and their rings and ideals are rooted in linear algebra. In this thesis, we present fast and practical algorithms for problems such as determinant computation, linear system solving, characteristic polynomial computation over number fields and polynomial rings. All the computations and experiments are carried out using the Hecke software package in Julia [FHHJ17].

The first chapter starts with an overview of modular algorithms in general. We address the problem of intermediate coefficient swell. Specially, this computational phenomenon occurs with problems in linear algebra and polynomial arithmetic that compute over rational numbers or finitely generated fields. To overcome this, in particular for the rationals and integers, the standard strategy is not to compute the number directly but to compute it modulo some other number M. This can be done using either the Chinese remainder theorem (CRT) ($M = p_1 p_2 \cdots p_m$) or a variation of Newton-Hensel lifting ($M = p^m$). Modular algorithms are frequently combined with rational or vector reconstruction methods to convert an integral result into a rational solution. We present reconstruction methods over number fields with a fast and simple vector-reconstruction algorithm using algebraic-denominator reconstruction method.

We present the major work of this research project in Chapter 2: discovering a fast, practical and deterministic algorithm for computing the determinant of matrices over number fields. We start with Abbott's and Mulders' [ABM99] approach for determinant computation. Currently the fastest algorithm for determinant computation over the integers is due to Storjohann [PS13]. His approach can be interpreted by the modules spanned by the rows of the matrices. Given a square matrix A, the corresponding module is enlarged through the addition of random elements, and each such step gives a factor of the determinant. This is done by computing divisors of the determinant using denominators of the solutions of linear systems $Ax = b$ over the rationals for a randomly chosen vectors b. In the final step, one needs to verify that the module is trivial, i.e. the basis matrix has determinant one. Storjohan calls this "unimodularity certification". The idea is that the module is trivial precisely if the matrix has an integral inverse. When generalizing the method over number rings, we encounter the problem that the denominator is an ideal and it is no longer a unique number which can be used to compute the determinant and number rings are not Euclidean domains. Moreover, modules generated by the rows of a matrix over number fields are in general not free, thus Strojohann's method cannot be used directly. Therefore, we use the theory of pseudo-matrices to overcome this problem. However, we could not use his approach for verifying the determinant, as it was slow in our case. Therefore, we used a different approach and obtained better run times.

Chapter 3 generalizes the problem of unimodular certification to number fields. However, due to the use of pseudo-bases, the approach in [PS12] does not carry over to matrices over number fields directly; here, we need the dual-module to be integral as well. The dual-module, in the pseudo-setting, is given by the inverse of the matrix and the inverses of ideals. In the algorithm we use higher-order lifting methods to obtain better complexity as in the integer case. The chapter ends with an outline of an approach, to obtain competitive run times using representations for algebraic integers that are similar to redundant number systems.

In Chapter 4, we use a modified version of the Dixon algorithm [Dix82] to address two classical problems in linear algebra: non-square linear system solving and matrix kernel computation. We present a deterministic algorithm for solving a non-square linear system over any fields. As the solution is not unique, we use a basis of the kernel to normalize the solution. That is, the implementation is accompanied by a fast algorithm to compute the kernel of a matrix of any size. We can extend this algorithm to compute the reduced row echelon form of a matrix using a lifting technique.

We discuss the problem of computing the characteristic polynomial and minimal polynomial of matrices over number fields in Chapter 5. First, we compute bounds for coefficients of the minimal polynomial and characteristic polynomial using Geršgorin theorem and bounds for the determinants of complex matrices. The existing algorithms can be optimized using a CRT-based modular approach. The existing minimal polynomial implementation in Hecke, which uses the Krylov method is slower than the division-free characteristic polynomial implementation. We present an additional method to compute the minimal polynomial computation using the characteristic polynomial algorithm, in the time of matrix multiplication.

The goal of Chapter 6 is to extend the above mentioned Strojohann's determinant computation algorithm over polynomial ring over finite fields. We have addressed the required sub-algorithms such as solving linear system, reconstructing rational polynomials and certifying unimodularity. To develop an optimized reconstruction method, we have used successfully the half-gcd approach. Interestingly, we were able to use higher-order lifting techniques to avoid intermediate degree explosion in the unimodular certification algorithm.

Chapter 1

Modular Algorithms

1.1 Introduction to Modular Algorithms

Many algorithms in linear algebra and polynomial arithmetic that compute over rational numbers, number fields or other finitely generated fields face the problem called the intermediate coefficient swell. For example, in polynomial greatest common divisor computation, there can be huge intermediate results but small outputs. The other problem is the growth in bit-length of numbers in the output compared to those in the input. In the determinant computation of a matrix $A \in \mathbb{Z}^{n\times n}$, Hadamard's bound guarantee that the bit-length of the determinant can only be up to n times that of entries in the input matrix. However, we have examples with intermediate results that are much large. When we are working with matrices over number fields, these computational phenomenons would occur the same. To overcome this, the standard strategy is to use modular methods.

A modular method is a more indirect way of computing: instead of solving a problem directly in a ring R, we do most of the work in one or more quotient rings $R/\mathfrak{i}_1, R/\mathfrak{i}_2, \ldots$ for some ideals $\mathfrak{i}_1, \mathfrak{i}_2, \ldots$ and then reconstruct the answer back in R. Many algebraic computation problems over a Euclidean domain R can be solved modulo one or several primes. There are three variants:

- Big prime ($m = p$ for a prime p).
- Small primes using CRT ($m = p_1 \cdots p_r$ for pairwise distinct primes $p_1, \ldots, p_r$ and $r \in \mathbb{Z}$).
- Prime power modular algorithms using Hensel/ Newton lifting ($m = p^r$ for a prime p and $r \in \mathbb{Z}$)

The first one is conceptually the simplest. If m is chosen large enough the method will find the correct result. Since working with numbers as large as m is still expensive, this strategy has been refined in other two variants, which are computationally superior.

In each case, three technical problems have to be addressed:

- Proof of correctness or a bound on the solution: i.e. Hadamard's bound.
- How to find a good moduli: CRT or lifting algorithms (i.e Dixon's algorithm.)
- How to reconstruct the solution from modular output: rational reconstruction.

1.2 Modular Algorithms for Matrices over $\mathbb{Z}$ and $\mathbb{C}$

In this section, we discuss some essential information that we need for modular algorithms over integers (or $\mathbb{R}$, $\mathbb{C}$). Our primary goal is to develop methods for solving classical problems coming from linear algebra.

1.2.1 Chinese Remainder Theorem

In CRT based modular method, one compute result modulo many small prime numbers such that the product of primes is large enough. Then, it combine the smaller results into big one. This method works as long as the results are unique enough to allow combinations, one has enough primes and one either has a bound on solution or a way to verify the result.

Theorem 1.2.1 (Chinese Remainder Theorem)**.** *Let $p_1, p_2, ..., p_k$ be a sequence of pairwise coprime integers, and $b_1, b_2, ..., b_k$ be another sequence of integers. Then the system of simultaneous linear congruence equations*

$$x \equiv b_1 \pmod{p_1}, \quad x \equiv b_2 \pmod{p_2}, \quad \dots x \equiv b_k \pmod{p_k}$$

has an infinite number of solutions $x \in \mathbb{Z}$ which form a unique congruence class $[x]_{p_1 p_2 ... p_k} \in \mathbb{Z}/p_1 p_2 ... p_k$ where $[x]_p = \{y \in \mathbb{Z} : y \equiv x \pmod{p}\}$.

In practice, there are two variants of CRT:

- CRT-iterative: given moduli $p_1, p_2, \dots, p_k$ a plain iteration is performed. First applying CRT to compute the moduli $p = p_1 p_1$, then $p = pp_3$ and so on. This method allows for early stopping of an iteration, considering the correctness of modular results.
- CRT-tree: this strategy is asymptotically faster than the iterative method. First, it computes moduli $p_1 p_2, p_3 p_4, \dots, p_{k-1} p_k$, then it continue as a tree, combining two pairs at once. This method requires modular images of all the primes as input.

1.2.2 P-adic Lifting

For the classical problem of computing the exact rational solution of a linear system, the modular lifting methods can decrease the bit-complexity beyond the Chinese remainder approach. Efficient algorithms for linear system solving are based on p-adic lifting: i.e. Dixon's algorithm.

Dixon's Algorithm

This is a p-adic Hensel lifting method for solving system of linear equations, which can accelerate the bit complexity beyond the CRT approach. Given a square linear system $Ax = b$ with a prime p such that $p \nmid \det(A)$, Dixon's method computes $\bar{x}$, such that $A\bar{x} \equiv b \pmod{p^m}$ for a given precision m. There are two phases of the algorithm. First, the inverse $C = A^{-1} \pmod{p}$ is computed with complexity $O(n^3)$. The second step computes a p-adic approximation $\bar{x}$ with $O(mn^2 \log n)$ integer operations for m iterations. Starting from $b_0 = b$, `DixonSolver` iterates $x_i \equiv Cb_i \pmod{p}$ and $b_{i+1} = p^{-1}(b_i - Ax_i)$ for $i = 0, \dots, m$, and obtains $\bar{x} = \sum_{i=0}^{m-1} x_i p^i$. Dixon's algorithm is efficient in practice with the total complexity of $O^{\sim}(n^3)$ ([Dix82]).

Bounds on Solutions

Hadamard's inequality provides a bound on the determinant of a matrix over complex numbers in terms of the length of its column vectors. Let d be the determinant of the square matrix $A \in \mathbb{C}^{n \times n}$. Hadamard's inequality says:

$$H = \prod_{i=1}^{n} \|A_i\| \geq |d| \tag{1.2.1}$$

where $|d|$ is the absolute value of d and $\|A_i\|$ represents the Euclidean length of the vector A_i, whose coordinates are given by the i-th column (or row) of the matrix A (refer [JH85]). Hadamard's bound is sharp for random matrices, but not for special matrices such as huge unimodular matrices (See [ABM99, Section 3], [AM01]).

By the Cramer's rule, we have that the denominator and numerators of the solution of the system $Ax = b$ is bounded by $\|B\| \prod_{i=1}^{n} \|A_i\|$.

1.2.3 Rational Reconstruction

Often the final step of these modular algorithms is combined with reconstruction methods, such as *rational reconstruction* or *continued fraction* to convert the modular result into the rational solution. As modular reduction discards some of the information, additional information such as the size or nature of the true result is required, and this necessary and complementary information has to be deduced from the original inputs.

Reconstruction of Integers from Modular Results (from $\mathbb{Z}/m$ to $\mathbb{Z}$)

Determine a bound B for the absolute value of the true answer using theoretical results. Then, conduct modular computations sufficient to achieve a precision of at least B.

e.g. Hadamard's bound gives us $\log|\det(A)| \leq n \log n/2 + n \log \|A\|_\infty$, where $\|A\|_\infty = \max_{ij} |A_{ij}|$. Hence, the bit-length of the product of primes required by a modular method to ensure the correctness of the determinant is $O(n(\log n + \log \|A\|_\infty))$.

Reconstruction of Rationals from Modular Results (from $\mathbb{Z}/m$ to $\mathbb{Q}$)

We compute a shortest vector (x, y) of the lattice $\Delta = \langle (m, 0), (r, 1) \rangle \subset \mathbb{Z}^2$ to reconstruct $\frac{a}{b} \in \mathbb{Q}$ from the modular result $r \in \mathbb{Z}/m$. If m is large enough we get $\frac{x}{y} = \frac{a}{b}$, under some conditions on prime divisors of m. An error tolerant algorithm which will reconstruct the correct result even in the presence of bad primes has been introduced in [BDFP15] with the bit-complexity of $O(\log^2 m)$. This strategy is slow in $\mathbb{Q}$, but has successfully been used in algebraic reconstruction: Section 1.3.4.

We can use the lattice basis reduction L^2-algorithm of Nguyen and Stehlé, which is a variant of LLL-algorithm in [LLL82] to obtain a quadratic complexity [NS09]: Given an integer d-dimensional lattice basis with vectors of Euclidean norm less than B in an n-dimensional space, the L^2 algorithm returns a reduced basis in $O(d^3 n(d + \log(B)) \log(B) \cdot \mathcal{M}(d))$ bit operations, where $\mathcal{M}(d)$ denotes the time required to multiply d-bit integers.

1.2.4 Vector Reconstruction

For the final step of Dixon's algorithm, we can extend rational reconstruction to a vector reconstruction algorithm. Many algorithms in linear algebra use vector reconstruction methods which computes the common denominator and numerators of the entries of the solution vectors from their modular images. The cost of modular algorithm depends on the bit length of the modulus m, and the existence of a common denominator. By Cramer's rule we can see that there exists a common denominator in our case. In our applications we use vector reconstruction algorithms which requires about half the size of the bit length of m. In other words, half as many modular image computations as the standard approach which does elementwise rational reconstruction. In this regards, most of the work has been done by A. Storjohann and C. Bright in [BS11].

Vector rational reconstruction methods takes as input a vector $a = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ of the images modulo m, and compute a pair $(\ell, n) \in (\mathbb{Z} \times \mathbb{Z}^n)$ such that $\ell a_i \equiv n_i \pmod{m}$, for $i = 1, \ldots, n$ and $\|[\ell \,|\, n]\| \leq N$. Here N is the given upper bound for the Euclidean norm of $\|[\ell \,|\, n]\|$.

The problem of solving the vector reconstruction problem is equivalent to finding short vectors of the lattice M.

$$M = \begin{pmatrix} 0 & 0 & \cdots & 0 & m \\ 0 & 0 & \cdots & m & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & m & \cdots & 0 & 0 \\ 1 & a_1 & \cdots & a_{n-1} & a_n \end{pmatrix}$$

Storjohann and Bright in [BS11] use lattice reduction to compute a "generating set" as $G = [\ell_j \,|\, n_j]_{1 \leq j \leq c}$ such that every solution $[\ell \,|\, n]$ of the vector reconstruction problem is a $\mathbb{Z}$-linear combination of G. Here, c is the size of the generating set of the solution as explained in [BS11, Section 4] (assuming $c \in O(1)$). Instead of applying the lattice reduction directly to the lattice M, in [BS11] they use an iterative approach to keep the row dimension bounded by $c + 1$ in the working lattice. They add columns of M one by one, and after the reduction they remove rows of the lattice whose norms are larger than N (See [BS11, Algorithm 3]). The basic idea of the algorithm can be explained as follows:

First, the lower-left 2×2 submatrix of M is reduced. $\begin{bmatrix} 0 & m \\ 1 & a_1 \end{bmatrix} \xrightarrow[\text{LLL}]{T_1} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$

Then the basis of the lower-left 2×3 matrix is obtained, using the same unimodular transformation T_1 as: $T_1 \begin{bmatrix} 0 \\ a_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

In the next step, a row is added to have the 3×3 lower-left submatrix of M, and reduced basis is computed.

$$\begin{bmatrix} 0 & 0 & m \\ b_1 & b_2 & B_1 \\ b_3 & b_4 & B_2 \end{bmatrix} \xrightarrow[\text{LLL}]{T_2} \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

Now, if it happens that the final vector $[c_7, c_8, c_9]$ in the Gram-Schmidt orthogonalization of this basis has norm larger than N, then we can discard the last row due to [vHN10, Lemma 2]. We can repeat the same process for all the columns of matrix M to find short vectors.

The cost of the vector reconstruction algorithm in [BS11] is $O(n(\log m)^2)$ bit operations for a vector of size n, using L^2 algorithm with quadratic complexity [NS09]. While elementwise rational reconstruction requires $m > 2N^2$ with $N \geq \beta$ the magnitude of the common denominator and numerators, the vector reconstruction algorithm requires only that $m > 2^{(c+1)/2} N^{1+1/c}$ with $N \geq \sqrt{n+1}\beta$ in order to succeed. However, this method is not fast in practice, due to the number of calls for L^2 algorithm.

1.3 Computing in Algebraic Number Fields

We begin this section with an introduction of algebraic number fields and their basic properties, which can be found in any book containing foundations of algebraic number theory, for example [Neu99] and [Lan94]. Then, we discuss variants of some algebraic reconstruction methods and a modified version of the vector reconstruction to use in our modular approach for linear system solving. Finally, we provide a computational model for number field arithmetic to be used in complexity anlysis of our algorithms.

1.3.1 Basics of Number Fields

A *number field* K is an finite extension of $\mathbb{Q}$. We have $K = \mathbb{Q}[\alpha] \simeq \mathbb{Q}[x]/f(x)$, for some monic and integral polynomial f with the primitive element α. The degree $[K : \mathbb{Q}]$ of the field extension is called *degree* of the number field K. Since $\mathbb{Q}$ is a field of characteristic zero, a number field of degree d admits d embeddings as into $\mathbb{C}$. Also, K can be considered as a $\mathbb{Q}$-vector space with a basis $\omega_1, \ldots, \omega_d$. For all $\beta \in K$ we have a map $\varphi_\beta : K \to K : x \mapsto \beta x$ which is $\mathbb{Q}$-linear, hence we have that $R_\beta \in \mathbb{Q}^{n\times n}$ such that

$$\beta(\omega_1, \ldots, \omega_d) = (\omega_1, \ldots, \omega_d)R_\beta.$$

R_β is called representation matrix (or (right) regular representation) of β. We define the norm and trace of β as $\mathrm{norm}(\beta) = N(\beta) = \det(R_\beta)$ and $\mathrm{trace}(\beta) = \mathrm{trace}(R_\beta)$ respectively. The *maximal order* or *ring of integers* of K is denoted by $\mathcal{O}$, is the integral closure of $\mathbb{Z}$ in K. The *discriminant* Δ_K of the number field K is defined to be $\det\left(((\mathrm{trace}(\omega_i\omega_j))_{i,j}\right)$, for any $\mathbb{Z}$-basis of $\mathcal{O}$

A fractional ideal $\mathfrak{d} \trianglelefteq K$ is finitely generated $\mathcal{O}$-sub-module of K. For some nonzero $r \in \mathcal{O}$ we have $r\mathfrak{d} \trianglelefteq \mathcal{O}$, and the intuition is that r clears the denominators in $\mathfrak{d}$. The minimal positive integer r with this property is defined to be the *denominator* of $\mathfrak{d}$. $\mathcal{O}$ is a Dedekind domain, therefore ideals can be uniquely factorized.

Theorem/ Definition 1.3.1. *The set $\mathcal{F}$ of fractional ideals forms a group with identity $\mathcal{O}$. The product of two ideals $\mathfrak{a}, \mathfrak{b} \trianglelefteq \mathcal{F}$ is the $\mathcal{O}$-module generated by the set $\{\alpha\beta \,|\, \alpha \in \mathfrak{a}, \beta \in \mathfrak{b}\}$, and inverse of $\mathfrak{a}$ is given by $\mathfrak{a}^{-1} = \{\alpha \in K \,|\, \alpha\mathfrak{a} \subseteq \mathcal{O}\}$.*

Given a non-zero ideal $\mathfrak{a} \trianglelefteq \mathcal{O}$, the unique positive integer $m \in \mathbb{Z}_{\geq 0}$ with $\langle m\rangle = \mathbb{Z} \cap \mathfrak{a}$ is called the *minimum* of $\mathfrak{a}$ and denoted by $\min(\mathfrak{a})$. Also, we have that $\min(\mathfrak{a}) = \min\{a \in \mathbb{Z}_{>0} \,|\, a \in \mathfrak{a}\}$.

We can apply modular methods to solve problems in any order of K, with a canonical epimorphism for a suitable ideal $\mathfrak{a}$: $\phi : \mathcal{O} \to \mathcal{O}/\mathfrak{a}$. Instead of looking for a solution $\alpha \in \mathcal{O}$, we look at the simpler problem in $\mathcal{O}/\mathfrak{a}$. Since, $\mathcal{O}$ is residually finite, the quotient $\mathcal{O}/\mathfrak{a}$ is finite for all ideals $\mathfrak{a} \trianglelefteq \mathcal{O}$. The norm of the ideal $\mathfrak{a}$ is $|\mathcal{O}/\mathfrak{a}|$, and denoted it by $N(\mathfrak{a})$. Norm is multiplicative.

1.3.2 Lattices as Tools for Algebraic Reconstruction Methods

In reconstructing solutions from their modular images, we use lattice based techniques. Lattices (denoted by $\Delta_\mathbb{Z}$ and $\Delta_\mathbb{R}$) considered in this section are subsets of $\mathbb{R}^d$ and are equipped with the usual scalar product (Euclidean norm $\|.\|$) as length. We fix the basis $\omega_1, \ldots, \omega_d$ for $\mathcal{O}$ as a $\mathbb{Z}$-module. Given the primitive element α for K, $f(x) = \prod(x - \alpha^{(i)})$. For $i = 1, \ldots, d$ the conjugates $\alpha^{(i)}$ of α are ordered as usual with d embeddings:

$$\begin{aligned} ()^{(i)} : K &\to \mathbb{C} \\ \alpha &\mapsto \alpha^{(i)} \end{aligned} \tag{1.3.1}$$

where $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(r_1)} \in \mathbb{R}$, $\alpha^{(r_1+1)} = \overline{\alpha^{(r_1+r_2+1)}}, \ldots, \alpha^{(r_1+r_2)} = \overline{\alpha^{(r_1+r_2+r_2)}} \in \mathbb{C}\backslash\mathbb{R}$ where (r_1, r_2) is the *signature* of K, and $r_1 + 2r_2 = d$. We define two isomorphisms on $\mathbb{R}^d := \Delta_\mathbb{R}$ and $\mathbb{Z}^d := \Delta_\mathbb{Z}$:

$$\begin{aligned} &\delta_\mathbb{R} : \mathcal{O} \to \mathbb{R}^d \\ &\quad x \mapsto (x^{(1)}, \ldots, x^{(r_1)}, \sqrt{2}\Re(x^{(r_1+1)}), \ldots, \sqrt{2}\Re(x^{(r_1+r_2)}), \sqrt{2}\Im(x^{(r_1+1)}), \ldots, \sqrt{2}\Im(x^{(r_1+r_2)})), \end{aligned}$$

$$\delta_\mathbb{Z} : \mathcal{O} \to \mathbb{Z}^d := \Delta_\mathbb{Z} : x = \sum_{i=1}^{d} x_i\omega_i \mapsto (x_1, \ldots, x_d).$$

Note that the map $\delta_{\mathbb{Z}}$ depends on the basis of $\mathcal{O}$, while $\delta_{\mathbb{R}}$ is canonical. Now, we can define two different measures on the same order $\mathcal{O}$ by applying the scalar products to the image of $\delta_{\mathbb{Z}}(\cdot)$ and $\delta_{\mathbb{R}}(\cdot)$ respectively. We use the Euclidean norm to define $\|x\|^2 := \sum_i^d x_i^2$ which measures the size of the coefficients of x, represented as a linear combination of $\omega_1, \ldots, \omega_d$ as $x = \sum_{i=1}^d x_1\omega_i \in \mathcal{O}$. Also, we define the norm *maximum coefficient* of x as $\|x\|_\varphi = \max\{|x_i|, \ldots, |x_d|\}$ to be used in cost analysis (see Section 1.3.8, Definition 1.3.21). Working with the lattice $\Delta_{\mathbb{Z}}$ has the advantage that it allows operations with integers only. But, the required bounds on solutions can only be obtained using conjugates.

T_2-Norm

We define the T_2-norm of x as $T_2(x) := \|\delta_{\mathbb{R}}(x)\|^2 = \sum_{i=1}^d |x^{(i)}|^2$ (see [FF00, Section 3]). Since we have two different norms $\|\cdot\|$ and $\sqrt{T_2}$ on the same finite dimensional space, they are equivalent to each other up to constants. This allows the T_2-norm to give a bound on the coefficients using bounds on the conjugates. Fieker in [FF00, Section 4] explains how to compute *norm change constants* $(c_1, c_2) \in \mathbb{R}^2_{>0}$ such that for all $x = \sum_{i=1}^d x_i\omega_i \in \mathcal{O}$,

$$T_2(x) \le c_1 \cdot \sum_i^d x_i^2 \quad \text{and} \quad \sum_i^d x_i^2 \le c_2 \cdot T_2(x) \tag{1.3.2}$$

Throughout this paper, we fix the notation (c_1, c_2) for the norm change constants.

Lemma 1.3.2. *T_2- norm is sub-multiplicative.*

Proof. Consider $x, y \in \mathcal{O}$, where $T_2(x) = \sum_{i=1}^d |x^{(i)}|^2$ and $T_2(y) = \sum_{i=1}^d |y^{(i)}|^2$ with their conjugates $x^{(i)}, y^{(i)}$ for $1 \le i \le d$. Then the following holds:

$$T_2(x) \cdot T_2(y) = \Big(\sum_{i=1}^d |x^{(i)}|^2\Big) \cdot \Big(\sum_{i=1}^d |y^{(i)}|^2\Big) = \sum_{i=1}^d |x^{(i)}|^2|y^{(i)}|^2 + \text{other terms} \ge \sum_{i=1}^d |x^{(i)}y^{(i)}|^2 = T_2(xy)$$

□

Moreover we have inequalities,

$$\sqrt{T_2(\alpha)} \le \sqrt{d}\max\{|\alpha^{(i)}| : 1 \le i \le d\}, \quad \text{and} \quad \max\{|\alpha^{(i)}| : 1 \le i \le d\} \le \sqrt{T_2(\alpha)} \tag{1.3.3}$$

Bounds on Lattice Elements

Let us denote the sublattice of $\Delta_{\mathbb{Z}}$ as $\Delta_{\mathbb{Z}}(\mathfrak{a})$, which corresponds to the ideal $\mathfrak{a}$ as a submodule of the ring $\mathcal{O}$ from the isomorphism $\delta_{\mathbb{Z}}$:

$$\begin{aligned}\delta_{\mathbb{Z}} : \mathcal{O} &\to \Delta_{\mathbb{Z}} \\ : \mathfrak{a} &\mapsto \Delta_{\mathbb{Z}}(\mathfrak{a})\end{aligned}$$

Let $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))$ be the square of the length of a shortest non-zero element in the sublattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$ ($\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))$ is called the 1-st successive minimum of $\Delta_{\mathbb{Z}}(\mathfrak{a})$). For applications, we need a way to estimate $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))$ in terms of the norm $N(\mathfrak{a})$ of the ideal $\mathfrak{a}$ ([FF00, Lemma 5]).

Lemma 1.3.3. *For $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))$ of $\Delta(\mathfrak{a})$ the following lower bound holds:*

$$\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) \ge \frac{d}{c_1} N(\mathfrak{a})^{2/d}$$

Proof. $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) = \min_{z\in\mathfrak{a}\setminus\{0\}} \|z\|^2 \geq \min_{z\in\mathfrak{a}\setminus\{0\}} T_2(z)/c_1 = \lambda_1(\Delta_{\mathbb{R}}(\mathfrak{a}))/c_1 \geq dN(\mathfrak{a})^{2/d}/c_1$. The last statement is due to $\lambda_1(\Delta_{\mathbb{R}}(\mathfrak{a})) = \min_{z\in\mathfrak{a}\setminus\{0\}} T_2(z)$ and *arithmetic and geometric mean* of z:

$$\|z\|^2/d \geq (\prod_{i=1}^{d} z^{(i)})^{2/n} = (N(z))^{2/d} \geq (N(\mathfrak{a}))^{2/d}, \quad \text{since } N(\mathfrak{a})|N(z). \tag{1.3.4}$$

□

Fundamental Domain

Definition 1.3.4. *Let Λ be a lattice in $\mathbb{R}^d$ with basis $b_1, \ldots, b_d \in \mathbb{Z}^d$.*

1. *The fundamental domain (centered) associated to the basis is defined by $\mathfrak{F} := \{ \sum_{i=1}^{d} t_i b_i \in \mathbb{R}^d \mid t_i \in [-1/2, 1/2)\}$.*
2. *Let $B(0, r)$ be the open ball of radius r, centered at 0. The radius of the largest ball inscribed in the closure of $\mathfrak{F}$ is $r_{\max} := \sup\{r \in \mathbb{R}^+ | B(0, r) \subset \mathfrak{F}\}$.*
3. *A measure of the orthogonality of the basis is given by the orthogonality defect: $\mathrm{OD}(b_1, \ldots, b_d) = \prod_{j=1}^{d} \|b_i\|/d(\Lambda)$, where $d(\Lambda)$ is the lattice determinant. Let $M = [b_1|\cdots|b_d]$ be the $d \times d$ matrix with the basis as columns. The lattice determinant is defined as $d(\Lambda) := \sqrt{|\det(M^t M)|}$.*

Due to Lenstra [Len82, Lemma 1], we have the following lemma.

Lemma 1.3.5. *The fundamental domain $\mathfrak{F}$ contains a d-dimensional ball $B(0, r)$ with $2r \geq \min_i \|b_i\|/\mathrm{OD}$, and all the non-zero vectors $x \in \Lambda$ have length $\|x\| \geq \min_i \|b_i\|/\mathrm{OD}$.*

We can perform Gram-Schmidt orthogonalisation for the basis of Λ and obtain an orthogonal basis $b_1^\star, \ldots, b_d^\star$. We have $d(\Lambda) = \det(b_i^t b_j) = \det(T)^2 \det((b_i^\star)^t b_j^\star) = \prod_{i=1}^{d} \|b_i^\star\|^2$, for some $T \in \mathrm{GL}(n, \mathbb{Q})$ where $(b_1^\star, \ldots, b_d^\star) = (b_1, \ldots, b_d)T$ and $\det(T) = 1$ (refer [Coh13, Prop 2.5.4]). Hence, we can reinterpret the inequality in Lemma 1.3.5 by replacing OD as:

$$r_{\max} \geq \frac{1}{2} \min_i \|b_i\| \times \prod_{i=1}^{d} \frac{\|b_i^\star\|}{\|b_i\|}. \tag{1.3.5}$$

The size of OD can be bounded from above by a constant that depends on the dimension of the lattice and the *quality ratio* of the basis reduction algorithm. Let $\theta \in (\frac{1}{4}, 1]$ denote the *quality ratio*. It is used to check the Lovàsz condition: $B_i \geq (\theta - \mu_{i,i-1}^2) B_{i-1}$. The original LLL basis reduction algorithm uses $\theta = 3/4$ ([LLL82], [Coh13, Section 2.6]). Take $\gamma(\theta) = (\theta - 1/4)^{-1} \geq 4/3$. Suppose that the basis $b_1, \ldots, b_d$ of the lattice Λ is LLL-reduced with quality ratio θ. From the properties of LLL reduced basis, ([Coh13, Theorem 2.6.2] and [Len82, Theorem 1]), Lenstra obtains the upper bound for OD:

$$1 \leq \mathrm{OD}(b_1, \ldots, b_d) \leq \gamma(\theta)^{d(d-1)/4}, \tag{1.3.6}$$

and a lower bound for $r_{\max}$:

$$r_{\max} \geq \frac{\min_i \|b_i\|}{2\gamma(\theta)^{d(d-1)/4}}. \tag{1.3.7}$$

The given bound for the $r_{\max}$ can be sharpened using Belabas' bound in [Bel04, Prop 3.10]:

$$r_{\max} \geq \frac{\|b_1\|}{2(3\sqrt{\gamma(\theta)}/2)^{d-1}} \tag{1.3.8}$$

When the lattice dimension increases, (1.3.8) provides a better bound than (1.3.7) does. We can reinterpret the Lemma 1.3.5 for all the non-zero vectors $x \in \Lambda$, and get a bound on $\lambda_1(\Delta_{\mathbb{Z}})$ as follows:

Lemma 1.3.6. *Suppose $b_1, \ldots, b_d$ be the* LLL *reduced basis of the lattice $\Delta_{\mathbb{Z}}$ with quality ratio θ. Then it holds that,*

$$\lambda_1(\Delta_{\mathbb{Z}}) = \min_{x \in \Lambda/\{0\}} \|x\|^2 > \frac{\|b_1\|^2}{(9\gamma(\theta)/4)^{d-1}}$$

This is straightforward from (1.3.8), since $\min_{x\in\Lambda/\{0\}} \|x\| > 2r_{\max}$. We extend this idea to a lattice corresponding to an ideal $\mathfrak{a} \triangleleft O$ to get a bound on the norm $N(\mathfrak{a})$.

Lemma 1.3.7. *Suppose $b_1, \ldots, b_d$ be the* LLL *reduced basis of the lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$ with quality ratio θ. Then, following bound holds for $N(\mathfrak{a})$.*

$$\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) > \frac{dN(\mathfrak{a})^{2/d}}{c_1(9\gamma(\theta)/4)^{d-1}}$$

Proof. Apply the arithmetic mean- geometric mean inequality (1.3.4) for $b_1 \in \mathfrak{a}/\{0\}$ in Lemma 1.3.6 with c_1 from (1.3.2): $\|b_1\|^2 \geq T_2(b_1)/c_1$. i.e. $T_2(b_1)/d \geq (\prod_{i=1}^{d} b_1^{(i)})^{2/d} = (N(b_1))^{2/d} \geq (N(\mathfrak{a}))^{2/d}$ (since $N(\mathfrak{a})|N(b_1)$). □

This is a reinterpretation of the bound given in Lemma 1.3.3, for a reduced basis. When we use ideals of the form $\mathfrak{p}^k$ for a fixed prime ideal $\mathfrak{p}$, Lemma 1.3.7 and 1.3.3 allows us to compute some k such that $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{p}^k)) > B$ for a given bound B. Even though this method has the advantage of staying in purely integral computations, we get worse bounds (k is too large in general). We can use the ideas suggested in [FF00, Section 7] to overcome this problem in practice, i.e. computing modular images of $\mathfrak{p}^k$, increasing k until it stabilizes.

1.3.3 Reconstruction of Algebraic Numbers from Modular Images

In order to solve the algebraic reconstruction problem (without denominators), the following observation (Lemma 1.3.8) by Lenstra is used (refer [Len82, Section 2] and [Bel04, Lemma 3.7]).

Lemma 1.3.8. *Let $\Lambda \subset \mathbb{R}^d$ be a lattice with $\mathfrak{F}, r_{\max}$ and matrix M defind as in Section 1.3.2. For $x \in \mathbb{R}^d$ there exists at most one $y \in \mathbb{R}^d$ such that $x \equiv y \pmod{\Lambda}$ and $|y| \leq r_{\max}$. Then, the unique element $y \in \mathfrak{F}$ is given by $y = x - M\left\lfloor M^{-1}x \right\rceil$.*

Problem: Given $\beta \in O$, $c \in \mathbb{R}^+$ and an ideal $\mathfrak{a} \subseteq O$, compute $\alpha \in O$, such that $\alpha - \beta \in \mathfrak{a}$ and $T_2(\alpha) < c$.

We can reconstruct α from $\beta \pmod{\mathfrak{a}}$, provided that the ideal $\mathfrak{a}$ is large enough with respect to the bound on α. This algebraic number reconstruction method is due to Fieker and Friedrichs [FF00, Section 4]. He extends Lemma 1.3.8 over the ring O, using $\mathbb{Z}$-isomorphism $\delta_{\mathbb{Z}}^{-1} : \Delta_{\mathbb{Z}} \to O$ and Lenstra's bound on $r_{\max}$ (1.3.7) with quality ratio $\theta = 3/4$. Refer [FF00, Theorem 1].

We can reconstruct α from $\beta \pmod{\mathfrak{a}}$, provided that the ideal $\mathfrak{a}$ is large enough with respect to the bound on α. This algebraic number reconstruction method is due to Fieker and Friedrichs [FF00, Section 4].

Theorem 1.3.9. *Let $\mathfrak{a}$ be an ideal such that the norm of $\mathfrak{a}$ satisfies*

$$N(\mathfrak{a}) > \left(\frac{c_1 c_2 c\, 2^{d(d-1)/2+1}}{d}\right)^{d/2}$$

for some $c \in \mathbb{R}$, and let $b_1, \ldots, b_d$ be a LLL-reduced basis of the lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$. Let $\beta \in O$ be arbitrary and $\beta = \sum_{i=1}^{d} q_i \delta_{\mathbb{Z}}^{-1}(b_i)$ with some $q_i \in \mathbb{Q}$. If there is an $\alpha \in \beta + \mathfrak{a}$ such that $T_2(\alpha) < c$, then $\alpha = \beta - \eta$, where $\eta := \sum_{i=1}^{d} \lfloor q_i \rceil\, \delta_{\mathbb{Z}}^{-1}(b_i)$.

Proof. Let $N(\mathfrak{a}) > \left((c_1c_2c\,2^{d(d-1)/2+2})/d\right)^{d/2}$. Lemma 1.3.3 yields that $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) > \left(c_2c\,2^{d(d-1)/2+2}\right)$. We have $r_{\max} \geq (\sqrt{\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))})/(2 \times 2^{d(d-1)/2})$, due to $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) = \min_i \|b_i\|^2$ and $\gamma(3/4) = 2$. Hence, $r_{\max}^2 > c_2c$. Now by Lemma 1.3.8 there exists a unique element $\alpha = \beta - \sum_{i=1}^{d} \lfloor q_i \rceil \delta_{\mathbb{Z}}^{-1}(b_i)$, given that $\|\alpha\| < \sqrt{c_2c}$. □

Belabas extens the Lemma 1.3.8 over the ring O using $\mathbb{Z}$-isomorphism $\delta_{\mathbb{Z}}^{-1} : \Delta_{\mathbb{Z}} \to O$ and the bound on $N(\mathfrak{a})$ is sharpened using the bound (1.3.8) on $r_{\max}$ ([Bel04, Lemma 3.12]). Hence, we can reinterpret the Theorem 1.3.9 as follows:

Theorem 1.3.10. *Let $b_1, \ldots, b_d$ be a LLL-reduced basis of the lattice $\Delta_{\mathbb{R}}(\mathfrak{a})$ with respect to T_2-norm and quality ratio θ. Let $\beta \in O$ be arbitrary and $\beta = \sum_{i=1}^{d} q_i \delta_{\mathbb{Z}}^{-1}(b_i)$ with some $q_i \in \mathbb{Q}$. If there is an $\alpha \in \beta + \mathfrak{a}$ such that $T_2(\alpha) < c$, then $\alpha = \beta - \sum_{i=1}^{d} \lfloor q_i \rceil \delta_{\mathbb{Z}}^{-1}(b_i)$, provided that*

$$N(\mathfrak{a}) \geq (2\sqrt{c/d} \cdot (3\sqrt{\gamma(\theta)}/2)^{d-1})^d.$$

Proof. We take the bound (1.3.8) on $r_{\max}$ for the lattice $\Delta_{\mathbb{R}}(\mathfrak{a})$. Then, $\|b_1\| = \sqrt{T_2(b_1)}$ and equation (1.3.4) can be applied directly to obtain the result. □

In his paper Belabas uses a prime ideal $\mathfrak{p}$ and he increases the precision k of the ideal $\mathfrak{a} = \mathfrak{p}^k$ until $r_{\max}$ satisfies $r_{\max}^2 > c_1c$. In practice, both Fieker and Belabas have chosen the lattice $\Delta_{\mathbb{Z}}$ to stay with purely integral computations while reconstructing α. As explained in Lemma 1.3.3, one can find a lower bound for the lengths of vectors in the lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$ (which is not reduced). That is, given c the ideal $\mathfrak{a}$ must satisfy $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) \geq (d/c_1) \cdot N(\mathfrak{a})^{2/d} > 4c$ to reconstruct α such that $\|\alpha\|^2 < c$ holds. Hence, we get $N(\mathfrak{a}) > (2\sqrt{c_1c/d})^d$ and, we can see this is the bound in Theorems 1.3.10 without the $\gamma(\theta)$ term coming from the LLL reduction. Pohst [Poh05, Lemma 2.2] has used the bound $N(\mathfrak{p}^k) \geq (2\sqrt{c/d})^d$ to reconstruct α, given that $T_2(\alpha) < c$. He enumerates short vectors in congruence class modulo p^k, using smaller k. Even though, enumeration uses comparatively smaller bounds, it is exponential in the runtime. The usage of the lattice $\Delta_{\mathbb{R}}$ with conjugates, makes it numericaly unstable, and difficult to analize with errors.

In our implementation we use a reduced basis. Since the reduced basis is as orthogonal as possible, it contains a ball with the maximum radius. Therefore we can obtain the required ideal $\mathfrak{a}$ with in few steps. Enlarging the ideal by multiplying with integers would preserve the maximality of the inscribed ball.

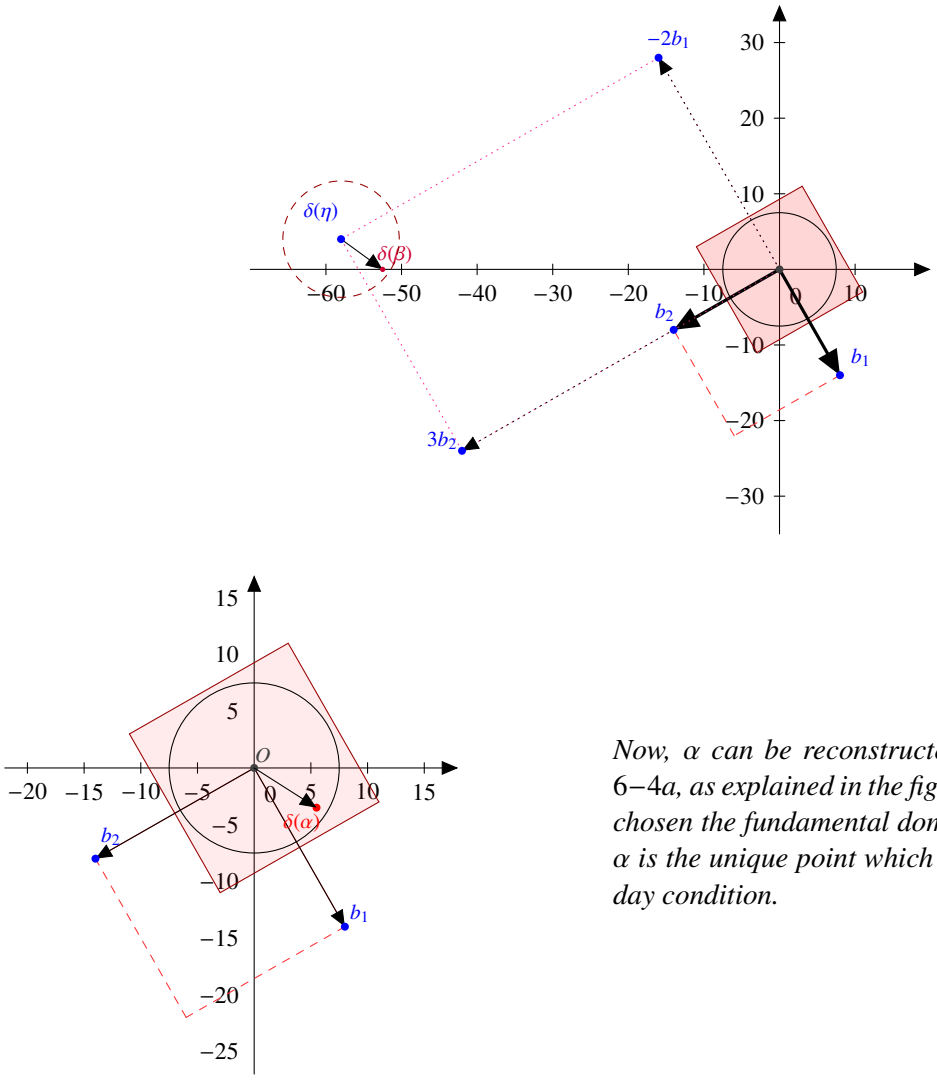
Example 1.3.11. *Let $K = \mathbb{Q}(a) = \mathbb{Q}(x)/x^2 + 1$, then the maximal order of K is: $O = \langle 1, a \rangle$. We define the isomorphism $\delta : O \mapsto \Delta := \mathbb{Z}^2$. Given $\beta = -52$ and ideal $\mathfrak{a} = \langle 130, 8 - 14a \rangle$, we reconstruct α such that $\alpha - \beta \in \mathfrak{a}$ with bound $\|\alpha\|^2 < c = 7.5$ on the coefficients of α. We have the ideal $\mathfrak{a}$ large enough as the Theorem 1.3.9 suggested: $T_2(\alpha) < 7.5c_1$, and $N(\mathfrak{a}) = 130 > 15c_1^2c_2 = 30$ (for $(c_1, c_2) = (2.0, 0.50)$ computed using Hecke).*

LLL-reduced basis matrix of $\mathfrak{a}$ is: $\begin{bmatrix} -14 & -8 \\ 8 & -14 \end{bmatrix}$

Hence, LLL basis for the sub-lattice $\Delta(\mathfrak{a})$ is $\langle b_1 = (8, -14),\ b_2 = (-14, -8) \rangle$, and $\mathfrak{a}$: $\left\langle \delta^{-1}(b_1) = 8 - 14a,\ \delta^{-1}(b_2) = -14 - 8a \right\rangle$.

Then β can be written in terms of b_1 and b_2 as: $\beta = -52 = -\frac{8}{5}\delta^{-1}(b_1) + \frac{14}{5}\delta^{-1}(b_2)$.

by rounding coefficients: $\eta = -2\delta^{-1}(b_1) + 3\delta^{-1}(b_2) = -58 + 4a$.



Now, α can be reconstructed: $\alpha = \beta - \eta = 6-4a$, as explained in the figure. Since we have chosen the fundamental domain large enough, α is the unique point which satisfies the boun-day condition.

1.3.4 Reconstruction with Denominators

Here we will extend the reconstruction method to non-integral algebraic numbers. For example, in linear system solving using modular methods, we require a method for reconstructing solutions which are not contained in the equation order. We fix the basis $\omega_1, \ldots, \omega_d$ for the ring $\mathcal{O}$. The ideal $\mathfrak{a}$ can be represented as a $\mathbb{Z}$-module in the ring $\mathcal{O}$: $(a_1, a_2, \ldots, a_d) = (\omega_1, \ldots, \omega_d)B^t$, where $B \in \mathbb{Z}^{d\times d}$ is the transformation matrix. The columns $B_1, B_2, \ldots, B_d$ form a basis of the sub-lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$. We define the map τ, which extend the lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$:

$$\tau : \mathbb{Z}^d \to \mathbb{Z}^{d+1} : \ (z_1, \ldots, z_d)^t \mapsto (0, z_1, \ldots, z_d)^t.$$

Problem: Given $\beta \in \mathcal{O}$, $c \in \mathbb{R}^+$ and an ideal $\mathfrak{a} \subseteq \mathcal{O}$, find $\alpha \in \mathcal{O}$ and $\ell \in \mathbb{N}$, such that $\ell\beta - \alpha \in \mathfrak{a}$, $\|\alpha\|^2 < c$, $\ell^2 < c$ and $\ell \notin \mathfrak{a}$.

Theorem 1.3.12. *Suppose $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) > 16c^2$. The lattice $\Delta_{\mathbb{Z}}$ is extended to a new lattice $\bar{\Delta}$ and*

$$\bar{\Delta}(\mathfrak{a}) := \left\langle \tau(\Delta_{\mathbb{Z}}(\mathfrak{a}), (1, \beta_1, \beta_2, \ldots, \beta_d)^t \right\rangle,$$

where $\beta = \sum_{i=1}^{d} \beta_i \omega_i$. *Then, a shortest vector of the lattice* $\bar{\Delta}(\mathfrak{a})$ *is given by:*

$$(0, \alpha_1, \ldots, \alpha_d) + \ell(1, 0, \ldots, 0),$$

where $\alpha = \sum_{i=1}^{d} \alpha_i \omega_i$.

This is due to [FF00, Theorem 2], and also it contains purely integral computations as before. In order to create the new lattice $\bar{\Delta}(\mathfrak{a})$, we can use a LLL-reduced basis of $\Delta_{\mathbb{Z}}(\mathfrak{a})$ with the extension map τ. The general case is explained in the next section.

Example 1.3.13. *Consider the same ideal* $\mathfrak{a} = \langle 130, -14a + 8 \rangle$ *in the Example 1.3.11. Given* $\beta = 5 + 7a$, *we reconstruct* $\alpha/\ell = \beta \pmod{\mathfrak{a}}$. *The basis matrix of* $\mathfrak{a}$ *is* $B = \begin{bmatrix} 130 & 0 \\ 36 & 2 \end{bmatrix}$ *Consider the basis matrix corresponding to the lattice* $\bar{\Delta}(\mathfrak{a})$: $M = \left[\begin{array}{c|c} 1 & [\beta_1 \,|\beta_2] \\ \hline O_{d\times 1} & B \end{array}\right] = \left[\begin{array}{c|cc} 1 & 5 & 7 \\ \hline 0 & 130 & 0 \\ 0 & 36 & 2 \end{array}\right]$

The LLL-reduced basis of $\bar{\Delta}(\mathfrak{a})$ *is given by:* $L = \mathrm{LLL}(M) = \begin{bmatrix} 4 & 0 & -2 \\ 2 & -4 & 6 \\ 1 & 5 & 7 \end{bmatrix}$.

Therefore, we have numerator $\alpha = -2a$, *denominator* $\ell = 4$, *and* $\alpha/\ell = -a/2$.

1.3.5 Reconstruction with Algebraic Denominators

For an algebraic number ℓ, consider the representation of $1/\ell$, using the previous reconstruction in Section 1.3.4. There, we get an integer denominator (in general the norm of ℓ), and $1/\ell$ is reconstructed as $\alpha/N(\ell)$, for some $\alpha \in K$. The norm $N(\ell) = \prod_{i=1}^{n} \ell^{(i)}$ and $\alpha = N(\ell)/\ell$ which are approximately n-times as large as ℓ (See Proposition 1.3.26). This implies that the number of primes required for the reconstruction with integer denominators is n-times more than that is required for the reconstruction with algebraic denominators. Given $\beta \in \mathcal{O}$ and an ideal $\mathfrak{a}$, here we reconstruct an algebraic numerator α and algebraic denominator ℓ for $\beta \pmod{\mathfrak{a}}$. The method is similar to the previous method with an extended lattice $\Delta'(\mathfrak{a})$. Let $M_{\mathfrak{a}}$ be the basis matrix, corresponding to the ideal $\mathfrak{a}$ (similar to the column matrix B above). We define the lattice extension ι:

$$\iota : \mathbb{Z}^d \to \mathbb{Z}^{2d} : \ (z_1, \ldots, z_d) \mapsto (0, \ldots, 0, z_{d+1}, \ldots, z_{2d}).$$

Let I_d be the $d \times d$ identity matrix and $O_{n\times m}$ be the $n \times m$ zero matrix over $\mathbb{Z}$. Take R_β to be the representation matrix of β. Then, we can construct the extended lattice:

$$\Delta'(\mathfrak{a}) = \langle (O_{d\times d} \mid M_{\mathfrak{a}}^t), (I_d \mid R_\beta^t) \rangle.$$

Consider the $\mathbb{Z}$-isomorphism: $\delta : \mathcal{O}^2 \to \Delta_{\mathbb{Z}}^2 := \mathbb{Z}^{2d}$, by extending the $\delta_{\mathbb{Z}}$ isomorphism to pairs (For $a = (\alpha, \beta) \in \mathcal{O}^2$, $\delta(a) = (\delta_{\mathbb{Z}}(\alpha), \delta_{\mathbb{Z}}(\beta))$. Then, it holds that $\Delta'(\mathfrak{a}) \subseteq \Delta_{\mathbb{Z}}^2$. Let Q' be the quadratic form associated to $\mathcal{O}^2$, which is achieved by applying the scalar product to the image of δ. If we have $a = (\alpha, \beta) \in \mathcal{O}^2$, the quadratic form gives $Q'(a) = \|\alpha\|^2 + \|\beta\|^2$.

Problem: Find α/ℓ with $\alpha, \ell \in \mathcal{O}$. Given $\beta \in \mathcal{O}$ such that $\ell\beta - \alpha \in \mathfrak{a}$ and bounds: $\|\alpha\|^2 < c$ and $\|\ell\|^2 < c$.

Lemma 1.3.14. *Suppose the first successive minimum* $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a}))$ *is greater than* $4c_2c_1^2B^2$ *for some* $B \in \mathbb{Z}_{>0}$, *then there exists at most one short vector* $v \in \Delta'(\mathfrak{a})$ *such that* $Q'(\delta^{-1}(v)) < B$.

Proof. Suppose $\upsilon_1, \upsilon_2 \in \Delta'(\mathfrak{a})$ with $Q'(\delta^{-1}(\upsilon_1)), Q'(\delta^{-1}(\upsilon_2)) < B$. For $i = 1, 2$, take $f_i, g_i \in \mathcal{O}$, such that $f_i = \sum_{j=1}^{d} f_{i,j}\omega_j$ and $g_i = \sum_{j=1}^{d} g_{i,j}\omega_j$, where $\omega_1, \dots \omega_d$ is the $\mathbb{Z}$-basis of $\mathcal{O}$. (The elements υ_i are of the form $\upsilon_i = [f_{i,1} \dots f_{i,d}][I_d \mid R_\beta^t] + [g_{i,1} \dots g_{i,d}][O_{d\times d} \mid M_\mathfrak{a}^t]$. This can be obtained from the first row of the matrix: $[R_{f_i}][I_d \mid R_\beta^t] + [R_{g_i}][O_{d\times d} \mid M_\mathfrak{a}^t]$, where R_{f_i} and R_{g_i} are representation matrices of f_i, g_i). Then, we have $\delta^{-1}(\upsilon_i) = (f_i, f_i\beta + a_i)$ for some $a_i = g_i\bar{a}_i$, where $a_i, \bar{a}_i \in \mathfrak{a}$. Let $f_i\beta + a_i = \alpha_i$. The bound on $\delta^{-1}(\upsilon_i)$ follows from $\|f_i\|^2 < B$ and $\|\alpha_i\|^2 < B$. Consider the element $\bar{\eta}$: $\bar{\eta} = f_2 \cdot \delta^{-1}(\upsilon_1) - f_1 \cdot \delta^{-1}(\upsilon_2) = (0, f_2a_1 - f_1a_2) = (0, \eta)$. Hence, we have $\eta = f_2a_1 - f_1a_2$: $\delta_\mathbb{Z}(\eta) \in \Delta_\mathbb{Z}(\mathfrak{a})$, where $\delta(\bar{\eta}) = \iota(\delta_\mathbb{Z}(\eta)) \in \iota(\Delta_\mathbb{Z}(\mathfrak{a})) \subseteq \Delta'(\mathfrak{a})$. Now consider the quadratic form for η:

$$\begin{aligned}
\|\eta\|^2 = Q'(\iota(\eta)) &= Q'(f_2 \cdot \delta^{-1}(\upsilon_1) - f_1 \cdot \delta^{-1}(\upsilon_2)) \\
&= Q'(f_2 \cdot (f_1, \alpha_1) - f_1 \cdot (f_2, \alpha_2)) \\
&\leq Q'(f_2f_1, f_2\alpha_1) + Q'(f_1f_2, f_1\alpha_2) \\
&= \|f_2f_1\|^2 + \|f_2\alpha_1\|^2 + \|f_1f_2\|^2 + \|f_1\alpha_2\|^2 \\
&\leq c_2(T_2(f_2f_1) + T_2(f_2\alpha_1) + T_2(f_1f_2) + T_2(f_1\alpha_2)) \\
&\leq c_2(T_2(f_2) \cdot T_2(f_1) + T_2(f_2) \cdot T_2(\alpha_1) + T_2(f_1) \cdot T_2(f_2) + T_2(f_1) \cdot T_2(\alpha_2)) \\
&\leq c_2(2c_1^2 \cdot \|f_2\|^2 \cdot \|f_1\|^2 + c_1^2 \cdot \|f_2\|^2 \cdot \|\alpha_1\|^2 + c_1^2 \cdot \|f_1\|^2 \cdot \|\alpha_2\|^2) \\
&< 4c_2c_1^2B^2
\end{aligned} \tag{1.3.9}$$

In the computation above, we have to shift from $\|.\|^2$ norm to $T_2(.)$, as $\|.\|^2$ is not sub-multiplicative. We apply the Lemma 1.3.2 (Cauchy Schwarz inequality) on quadratic form $T_2(.)$. The (1.3.9) shows that $\|\eta\|^2 < 4c_2c_1^2B^2$. We conclude $\lambda_1(\Delta_\mathbb{Z}(\mathfrak{a})) > 4c_2c_1^2B^2$, such that $\eta = 0$ and $\upsilon_1 = \upsilon_2$.

□

Theorem 1.3.15. *Suppose we have $\lambda_1(\Delta_\mathbb{Z}(\mathfrak{a})) > 16c_2c_1^2c^2$, and there exists $\alpha, \ell \in \mathcal{O}$ such that $\ell\beta - \alpha \in \mathfrak{a}$ and $\|\alpha\|^2, \|\ell\|^2 < c$. If $\ell = \sum_{i=0}^{d} \ell_i\omega_i$ and e_i is the ith canonical basis of $\mathbb{Z}^{2d}$ then the shortest vector of the lattice $\Delta'(\mathfrak{a})$ is $\upsilon = \sum_{i=1}^{d} \ell_ie_i + \iota(\delta(\alpha))$.*

Proof. As $\ell\beta - \alpha \in \mathfrak{a}$ there exists $a \in \mathfrak{a}$ such that $\alpha = \ell\beta + a$. The tuple $(\ell, \ell\beta + a)$ is an element of $\delta^{-1}(\Delta'(\mathfrak{a}))$. Take $\delta^{-1}(\upsilon) = (\ell, \ell\beta + a) = (\ell, \alpha)$. Then, $\upsilon = \sum_{i=1}^{d} \ell_ie_i + \iota(\delta(\alpha))$. Given bounds $\|\alpha\|^2 < c$ and $\|\ell\|^2 < c$, we have $Q'(v) = \|\ell\|^2 + \|\alpha\|^2 < 2c$. If we let $B = 2c$ (in Lemma 1.3.14), υ must be the shortest vector in $\Delta'(\mathfrak{a})$. □

We use Lemma 1.3.3 to interpret Theorem 1.3.15 in terms of norm of the ideal $\mathfrak{a}$.

Theorem 1.3.16. *Let $\mathfrak{a}$ be an ideal such that $N(\mathfrak{a}) > (16c_1^3c_2^3c^2/d)^{d/2}$. Given bounds $T_2(\alpha), T_2(\ell) < c$ and $\beta \in \mathcal{O}$ such that $\ell\beta - \alpha \in \mathfrak{a}$, α and ℓ can be reconstructed using a shortest vector υ of the lattice $\Delta'(\mathfrak{a})$ (as described in Theorem 1.3.15).*

Proof. Assume that $N(\mathfrak{a}) > (16c_1^3c_2^3c^2/d)^{d/2}$. Lemma 1.3.3 yields that $\lambda_1(\Delta_\mathbb{Z}(\mathfrak{a})) > 16c_1^2c_2^3c^2$. Provided that $T_2(\eta) \leq c$ for any $\eta \in \mathcal{O}$, we have $\|\eta\|^2 \leq c_2c$ by (1.3.2). Hence, we can replace the T_2-norm by $\|\cdot\|^2$ norm if we let $\lambda_1(\Delta_\mathbb{Z}(\mathfrak{a})) > 16c_2c_1^2c^2$. The rest of the proof follows from Theorem 1.3.15. □

Now we suppose the basis of $\Delta_\mathbb{Z}(\mathfrak{a})$ is LLL reduced. While we extend above results in to this case, the bound given in Theorem 1.3.15 does not change:

Lemma 1.3.17. *The reconstructed result in Theorem 1.3.15 does not depend on the basis $\Delta_\mathbb{Z}(\mathfrak{a})$ being reduced.*

Proof. We fix a lattice basis reduction algorithm (e.g. LLL). Let $M_{\mathfrak{a}}$ be the basis matrix of $\Delta_{\mathbb{Z}}(\mathfrak{a})$ and $L_{\mathfrak{a}}$ be the basis matrix corresponding to the reduced basis of $\Delta_{\mathbb{Z}}(\mathfrak{a})$. Consider the two extended lattices for the two cases: $\Delta'(\mathfrak{a}) = \langle (O_{d\times d} \mid M_{\mathfrak{a}}^t), (I_d \mid R_\beta^t)\rangle$ and $\Delta'_L(\mathfrak{a}) = \langle (O_{d\times d} \mid L_{\mathfrak{a}}^t), (I_d \mid R_\beta^t)\rangle$. Since, the reduced basis matrix of $(O_{d\times d} \mid M_{\mathfrak{a}}^t)$ is $(O_{d\times d} \mid L_{\mathfrak{a}}^t)$, we have that the two lattices $\Delta'_L(\mathfrak{a})$ and $\Delta'(\mathfrak{a})$ are same. Therefore, we have same properties for reduced bases $\Delta'_L(\mathfrak{a})$ and $\Delta'(\mathfrak{a})$, and we obtain same bounds. □

We apply the bound improved by Belabas in Lemma 1.3.7 to interpret Theorem 1.3.15 in terms of the norm of the ideal $\mathfrak{a}$.

Theorem 1.3.18. *Let $\mathfrak{a}$ be an ideal such that the basis of $\Delta_{\mathbb{Z}}(\mathfrak{a})$ is* LLL *reduced with quality ratio θ and,*

$$N(\mathfrak{a}) > \left((4c\sqrt{c_1^3 c_2/d}) \cdot (3\sqrt{\gamma(\theta)}/2)^{d-1}\right)^d.$$

Given bounds $\|\alpha\|^2, \|\ell\|^2 < c$ and $\beta \in \mathcal{O}$ such that $\ell\beta - \alpha \in \mathfrak{a}$, α and ℓ can be reconstructed using a shortest vector v of the lattice $\Delta'(\mathfrak{a})$ (as described in Theorem 1.3.15).

Proof. Simplifying $(dN(\mathfrak{a})^{2/d})/(c_1(9\gamma(\theta)/4)^{d-1}) > 16c_2c_1^2c^2$. Then Lemma 1.3.7 yields that $\lambda_1(\Delta_{\mathbb{Z}}(\mathfrak{a})) > 16c_2c_1^2c^2$ and Theorem 1.3.15 holds due to Lemma 1.3.17. □

The bound suggested in Theorem 1.3.18 is larger than the one in Theorem 1.3.16, but it allows us to use a LLL-reduced basis. Since this basis is as orthogonal as possible we can efficiently enlarge the lattice $\Delta_{\mathbb{Z}}(\mathfrak{a})$ to obtain the required bound on $\mathfrak{a}$.

Example 1.3.19. *For $\mathfrak{a} = \langle 130, -14a + 8\rangle$, we reconstruct $5 + 7a \pmod{\mathfrak{a}}$ with algebraic denominator ℓ and numerator α. The basis matrix corresponding to the lattice $\Delta'(\mathfrak{a})$ is given by*

$$M = \left[\begin{array}{c|c} I_d & R_\beta \\ \hline O_{d\times d} & B \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 0 & 5 & 7 \\ 0 & 1 & -7 & 5 \\ \hline 0 & 0 & 130 & 0 \\ 0 & 0 & 36 & 2 \end{array}\right], \text{ whre } R_\beta \text{ is the representation matrix of } \beta.$$

Then the LLL-reduced basis of $\Delta'(\mathfrak{a})$ is $L = \mathrm{LLL}(M) =$

$$\begin{bmatrix} 2 & -1 & 3 & 1 \\ 1 & 2 & -1 & 3 \\ 0 & 4 & 2 & 0 \\ -4 & 0 & 0 & 2 \end{bmatrix}$$

Therefore, we have numerator $\alpha = a + 3$ denominator $\ell = -a + 2$ and $\ell\beta - \alpha \in \mathfrak{a}$.

Given $a \in \mathcal{O}$ and $p \in \mathbb{Z}$, $a \pmod{p}$ reduces the coefficient vector of a modulo p and returns the corresponding element. The coefficient vector of the result will have entries x with $0 \le x \le p$.

Algorithm 1 reconstructs an algebraic denominator and numerator of $\beta \pmod{p}$, based on LLL lattice basis reduction algorithm. The algorithm would produce the principal ideal generated by p. Therefore, the basis matrix would be $p \cdot I_n$ and it is already reduced. The same algorithm can be used for modular reconstruction with respect to an ideal $\mathfrak{a}$. Then, the basis matrix $p \cdot I_n$ is replaced by some basis matrix $M_{\mathfrak{a}}$ of ideal $\mathfrak{a}$.

Algorithm 1 Algebraic Reconstruction (`AlgRecon`)

Input: $a \in O$, $p \in \mathbb{Z}$ and basis $\{\omega_1, \ldots, \omega_d\}$ of O.
Output: α and d such that $\frac{\alpha}{\ell} = a \pmod{p}$

1: R_a = representation matrix of a

$$M = \left[\begin{array}{c|c} O_{d\times d} & p \cdot I_d \\ \hline I_d & R_a \end{array}\right]$$

2: Compute the LLL reduced basis L of M.
3: $\alpha = \sum_{i=1}^{d} L_{1,i}\omega_i$
4: $\ell = \sum_{i=1}^{d} L_{1,d+i}\omega_i$
5: **return** α, ℓ

1.3.6 Vector Reconstruction Methods

Here, we present two vector reconstruction algorithms, which can be used in classical linear algebra problems such as linear system solving and kernel computation over K. First, we extend the vector rational reconstruction algorithm in [BS11] as a vector algebraic reconstruction algorithm. Then we present a simple and fast vector reconstruction method which uses the algebraic denominator reconstruction optimally.

Extension to Bright's and Storjohann's method over K

Here, we extend the vector rational reconstruction algorithm of Storjohann and Bright in [BS11] over number fields. Given a vector $A = (a_1, \ldots, a_n) \in O^n$ and an ideal $\mathfrak{p}$ (or a integer), vector reconstruction problem asks for the common denominator $\ell \in O$ and numerators $n \in O^n$ such that $\ell a_i \equiv n_i \pmod{\mathfrak{p}}$ for $i = 1, \ldots, n$. Let P be the basis matrix of the ideal $\mathfrak{p}$ (or if it is a number p, $P = pI_d$) and R_{a_i} be the representation matrix of a_i. Then any short vector of the lattice M gives a solution for the vector rational reconstruction problem as explained in the Section 1.2.4. The only difference is that, the removal of rows and addition of columns happence with the size of d instead of one row (or column) at a time.

$$M = \begin{pmatrix} 0 & 0 & \cdots & 0 & P \\ 0 & 0 & \cdots & P & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & P & \cdots & 0 & 0 \\ I_d & R_{a_1} & \cdots & R_{a_{n-1}} & R_{a_n} \end{pmatrix}$$

We have tested [BS11, Algorithm 3] over number fields. There, we have to apply the expensive lattice reduction n-times for matrices of size $d\times$ with the total cost of $O(nd^2\beta^2)$ operations in $\mathbb{Q}$ using L^2 algorithm. Here, β is the size of the coefficients of the input. The gain here is that, we require about the half the size of p, that is required for elementwise rational reconstruction. The experimental results shows that this method is slower than the new approach which is presented in the next section.

Vector Reconstruction Using Common Denominator

The algebraic vector reconstruction algorithm in Algorithm 2, computes the common denominator while reconstructing the solution. The initial assumption is that all the denominators in the solution vector are same. Given $a \in O^n$, $p \in \mathbb{Z}_{>0}$ and a bound B, if exist, we reconstruct the vector S such that $S \equiv a \mod p$ (that is $S_i \equiv a_i \pmod{p}$ for each $i = 1, \ldots, n$) and the coefficients of numerators and denominator of S are bounded by B.

Algorithm 2 Vector Algebraic Reconstruction (`VecAlgRecon`)

Input: Vector $a \in O^n$, an integer p and bound B.
Output: false or true with a reconstructed matrix S such that $S \equiv a \pmod{p}$.

```
1:  D = K(1) and S = O_{n×1}.
2:  for i = 1, ..., n do
3:      b = a_i D
4:      b = b (mod p)
5:      if all the coefficient of b ≤ B then
6:          S_i = b/D
7:      else
8:          α, d_1 = AlgRecon(b, p)
9:          D = D d_1
10:         if any coefficient of D > B then
11:             return false, S
12:         end if
13:         S_i = α/D
14:     end if
15: end for
16: return true, S
```

Theorem 1.3.20. *The Algorithm 2 is correct and has the complexity $O^{\sim}(\beta n d^5)$ for a vector of size n with entries of bitsize β.*

Proof. Consider the vector $a = (a_1, \ldots, a_n)$. We start algebraic reconstruction from first entry a_1. Let $\frac{\alpha_1}{d_1} \equiv a_1 \pmod{p}$. Let α_2' and d_2' be the reconstructed numerator and denominator of $d_1 a_2$. If $D = d_1$ is a common denominator then d_2' is a unit and $\alpha_2' \equiv d_1 a_2 \pmod{p}$. If all the coefficients of $\alpha_2' \leq B$ we have obtained the correct numerator $\alpha_2 = \alpha_2'$ without applying the reconstruction algorithm. Otherwise, we do the algebraic reconstruction for $d_1 a_2$ to find α_2' and d_2' such that $\alpha_2'/d_2' \equiv d_1 a_2 \pmod{p}$. Hence, we get the numerator $\alpha_2 = \alpha_2'$ and denominator $d_2 = d_1 d_2'$ such that $\alpha_2/d_2 \equiv a_2 \pmod{p}$ and new denominator is $D = d_1 d_2$. Other entries of the vector a can be constructed continuing this method. We can set a early abort condition for incorrect results, if any coefficient of $D > B$. Therefore, the correctness of the algorithm follows from the construction.

If the L^2-algorithm is used in the algebraic reconstruction, it runs in (at most) $O^{\sim}(\beta^2 d^5)$ operations on number of bitsize $O(\beta)$. Hence, the worst case complexity of the algorithm with respect to the bit size of the input is $O^{\sim}(n\beta^2 d^5)$. When the denominators of a are all equal, the bit complexity would be $O^{\sim}(\beta^2 d^5)$. □

This is a much faster approach as we accumulate the denominator, and in practice, we observed that typically after the second iteration, we do not have to use lattice reduction.

1.3.7 Linear System Solving Using Lifting Method

One of the main tool that we use in this thesis is the `DixonSolver` Algorithm 3. Here we have extended Dixon's algorithm to number fields.

Algorithm 3 Dixon Solver (`DixonSolver`)

Input: Matrix $A \in K^{n\times n}$, $b \in K^{n\times 1}$, a prime number p ($p \perp \det(A)$) and a bound B on solutions.
Output: Solution matrix S of the linear system $Ax = b$.

$C = A^{-1} \pmod p$
$D = b$, $s = 0$ and $p_1 = p$.
while true **do**
  $x \equiv Cb \pmod p$
  $s = s + xp_1$
  $p_1 = p_1 p$
  if $p_1 > B$ **then**
    $f, S = \texttt{VecAlgRecon}(s, p_1)$
    if f **and** $AS = b$ **then**
      return S
    end if
  end if
  $b = p^{-1}(b - Ax)$
end while

First we use Cramer's rule to compute a bound on the coefficients of numerators and denominators of the solution S of $Ax = b$. Since, the solution S can be written as quotients of determinants of some matrices, we need a method to estimate sizes of determinants (in the next section, we explain how to get bounds on determinants). Then we can use Theorem 1.3.18 to decide the prime power that is required to guarantee the solution.

1.3.8 Bound and Norm for Matrices over Number Fields

Let K be a number field of degree d and let $A \in K^{n\times n}$. We can get a bound on the coefficients of the determinant of a matrix A using Hadamard's bound for conjugate matrices of A (conjugate matrices are the corresponding complex matrices given by the map $c^{(i)}$ as defined below). Consider the map $(\cdot)^{(i)}$ as described in the Section 1.3.2:

$$(\cdot)^{(i)} : K \to \mathbb{C} : \ \alpha \mapsto \alpha^{(i)}.$$

Assuming the conjugates of any element in K are ordered in the usual way, we can extend the map $(\cdot)^{(i)}$ for matrices over $K^{n\times n}$:

$$\begin{aligned} c^{(i)} &: K^{n\times n} \to \mathbb{C}^{n\times n} \\ &: A \mapsto A^{(i)} \end{aligned}$$

The map $c^{(i)}$ gives the matrix $A^{(i)}$, where all the entries are the i-th conjugate of the entries of matrix A. The j,k-th entry a_{jk} of A is mapped to $a_{jk}^{(i)}$ of $A^{(i)}$. Now we can compute Hadamard's bound for each conjugate matrices $A^{(i)}$ as they are complex matrices. Let $B^{(i)}$ be Hadamard's bound for the matrix $A^{(i)}$: $|\det(A^{(i)})| \le B^{(i)}$. Conjugation is an isomorphism, which implies that $\det(A^{(i)}) = \det(A)^{(i)}$. Using these facts, we can apply the T_2-norm, to get a bound on the determinant with respect to $B = \sum_{i=1}^{d} {B^{(i)}}^2$ as:

$$T_2(\det(A)) = \sum_{i=1}^{d} |\det(A)^{(i)}|^2$$
$$\leq \sum_{i=1}^{d} B^{(i)^2} = B \tag{1.3.10}$$

When a bound on coefficients of the determinant is needed, we may apply the norm change constant in (1.3.2) and obtain that $\|\det(A)\| \leq \sqrt{c_2 B}$. In next sections, we relate the the size of the coefficients of $\det(A)$, with the size of the input matrix A using the measure $\|\cdot\|_\varphi$.

Norm on the Size of Coefficients

For $A^{(i)} = [a_{j,k}^{(i)}] \in \mathbb{C}^{n\times n}$, let $C^{(i)} \in \mathbb{R}$ such that $C^{(i)} \geq \max_{j,k}\{|a_{j,k}^{(i)}|\}$. Using the Hadamard bound on $A^{(i)}$, we have that $|\det(A^{(i)})| \leq n^{n/2} C^{(i)^n}$ from [GG13, Theorem 16.6]. We define the following norm for future applications.

Definition 1.3.21. *Let K be a number field of degree d with ring of integers $\mathcal{O}$, by $\Omega = \{\omega_1, \omega_2, \ldots, \omega_d\}$ we denote the $\mathbb{Z}$-basis of $\mathcal{O}$ with $\omega_1 = 1$. Given $\alpha \in \mathcal{O}$ with respect to the integral basis Ω by its coefficient vector $(\alpha_1, \ldots, \alpha_d) \in \mathbb{Z}^d$ satisfying $\alpha = \sum_{i=1}^{d} \alpha_i \omega_i$. For a matrix $A \in \mathcal{O}^{n\times m}$ we extend $\|A\|_\varphi = \max\{\|A_{ij}\|_\varphi \mid 1 \leq i \leq n, 1 \leq j \leq m\}$.*

The Lemma 1.3.22 provides a multiplicative property of the norm $\|\cdot\|_\varphi$.

Lemma 1.3.22. *For $\alpha, \beta \in \mathcal{O}$ the norm $\|\cdot\|_\varphi$ satisfies $\|\alpha\beta\|_\varphi \leq d c_1 \sqrt{c_2}\|\alpha\|_\varphi \|\beta\|_\varphi$, where c_1 and c_2 are the norm change constants of $\mathcal{O}$ as described in* (1.3.2)*.*

Proof. First we consider the $\|\cdot\|^2$ norm on $\alpha\beta$. Suppose α_i and β_i for $i = 1, \cdots, d$ are the coefficients of α and β respectively. Using Lemma 1.3.2, we have: $\|\alpha\beta\|^2 \leq c_2 T_2(\alpha\beta) \leq c_2 T_2(\alpha) T_2(\beta) \leq c_2 c_1 \|\alpha\|^2 c_1 \|\beta\|^2 = c_2 c_1^2 \sum_{i=1}^{d} \alpha_i^2 \sum_{i=1}^{d} \beta_i^2$. From this, we can get the required relation as $\|\alpha\beta\|_\varphi \leq \sqrt{\|\alpha\beta\|^2} \leq \sqrt{c_2 c_1^2 d \|\alpha\|_\varphi^2 d \|\beta\|_\varphi^2} \leq d c_1 \sqrt{c_2}\|\alpha\|_\varphi \|\beta\|_\varphi$. □

Bound on the Size of the Coefficients of $\det(A)$ in terms of $\|A\|_\varphi$

Now, Lemma 1.3.23 computes bounds on $T_2(\det(A))$ and the maximum coefficient of $\det(A)$, in terms of the maximum coefficient of matrix entries of A.

Lemma 1.3.23. *For a matrix $A \in \mathcal{O}^{n\times n}$, it holds that $T_2(\det(A)) \leq n^n (c_1 d \|A\|_\varphi^2)^n$ and $\|\det(A)\|_\varphi \leq (c_2^{1/2}(c_1 n d)^{n/2})\|A\|_\varphi^n$.*

Proof. Consider the corresponding conjugate matrices of A as $A^{(i)}$ for $i = 1, \ldots, d$. Let e be the matrix entry of A with the maximum T_2-norm. For all $i = 1, \ldots, d$, let $e^{(i)}$ be the corresponding ith conjugates of e and e_i be the ith coefficient of e. From the above result, we have that $|\det(A^{(i)})| \leq n^{n/2} |e^{(i)}|^n$. By (1.3.10) it satisfied: $T_2(\det(A)) \leq \sum_{i=1}^{d} n^n |e^{(i)}|^{2n} \leq n^n \Big(\sum_{i=1}^{d} |e^{(i)}|^2 \Big)^n$. Now we can change the norm and obtain a bound on coefficients using the fact that $|e_i| \leq \|A\|_\varphi$: $\sum_{i=1}^{d} |e^{(i)}|^2 \leq c_1 \sum_{i=1}^{d} |e_i|^2 \leq c_1 \sum_{i=1}^{d} \|A\|_\varphi^2 = c_1 d \|A\|_\varphi^2$. Therefore, we get: $T_2(\det(A)) \leq n^n (c_1 d \|A\|_\varphi^2)^n$. Combining this with changing norms on $\det(A)$, we obtain a bound on $\|\det(A)\|_\varphi$ as follows: $\|\det(A)\|_\varphi \leq \sqrt{c_2 T_2(\det(A))} \leq (c_2^{1/2}(c_1 n d)^{n/2})\|A\|_\varphi^n$. □

1.3.9 Computational Model and Cost Functions

When we develop a computational model for number field arithmetic, we need a notion of size that bounds the size required to represent ideals and field elements.

Presentation of Ideals

There are mainly two ways to present an ideal $\mathfrak{a}$ of $\mathcal{O}$ as explained in [PZ97, Section 6.3] and [BP91, Section 3].

> **$\mathbb{Z}$-Basis Presentations**: For every integral $\mathcal{O}$-ideal $\mathfrak{a}$ has a $\mathbb{Z}$-basis representation. Since $\mathfrak{a}$ is a free $\mathbb{Z}$-module of rank d, it can be represented as $\mathfrak{a} = \mathbb{Z}\eta_1 + \mathbb{Z}\eta_2 + \cdots + \mathbb{Z}\eta_d$, with $\eta_i \in \mathcal{O}$.
>
> **Two Element Presentation**: Every integral and fractional ideal $\mathfrak{a}$ can be generated by two elements, one of which can be chosen randomly in $\mathfrak{a}/\{0\}$. That is: $\mathfrak{a} = a\mathcal{O} + \alpha\mathcal{O}$, where $a \in \mathfrak{a}$ is chosen to be a positive integer. For example, the least positive integer in $\mathfrak{a}$, or the norm if $\mathfrak{a}$, are convenient. The construction of this representation is explained in [FHHJ17, Section 6]. For fractional ideals, we can find a presentation of the form $\mathfrak{a} = a\mathcal{O} + \alpha/k\mathcal{O}$ with $a, k \in \mathbb{Z}_{>0}$ and with $\alpha \in \mathcal{O}$. In this thesis, two generators of an ideal $\mathfrak{a}$ are called: `gen_one`($\mathfrak{a}$) and `gen_two`($\mathfrak{a}$).

Conceptually, 2-element presentation is less storage consuming and some operations are much more efficient in this setting ([FHHJ17, Section 6]). When the storage requirements for $\mathfrak{a}$ is considered, it takes $(d+1)$ integers to store generators using two element representation. If we have picked an integral basis for K, then it takes d^2 integers to store the basis itself.

Let P be a finite set of prime numbers, and P_K denote the (finite set of) prime ideals of $\mathcal{O}_K$ that lie over some prime in P.

Definition 1.3.24. *Let $\mathfrak{a}$ be a fractional $\mathcal{O}$-ideal. We call $a\mathcal{O} + \alpha\mathcal{O}$, with $a \in \mathbb{Z}_{\geq 0}$ and $\alpha \in K$ a P-normal presentation for $\mathfrak{a}$, if $\mathfrak{a} = a\mathcal{O} + \alpha\mathcal{O}$, and for the factorization into prime ideals in $\mathcal{O}$ it holds that: both $\mathfrak{a}$ and $a\mathcal{O}$ are divisible only by primes in P_K, and the ideal $\alpha\mathfrak{a}^{-1}$ is not divisible by any prime in P_K.*

Some important properties of normal presentations, from [BP91, Section 3]:

1. Let $\mathfrak{a} = a\mathcal{O} + \alpha/k\mathcal{O}$ be a presentation for the fractional ideal $\mathfrak{a}$, with $a, k \in \mathbb{Z}_{>0}$ and $\alpha \in \mathcal{O}$. Then $\mathfrak{a}$ admits a P-normal presentation with P consisting of the prime numbers dividing ak.

2. If $\mathfrak{a} = a\mathcal{O} + \alpha\mathcal{O}$ and $\mathfrak{b} = b\mathcal{O} + \beta\mathcal{O}$ are P-normal presentations, then $\mathfrak{a}\mathfrak{b} = ab\mathcal{O} + \alpha\beta\mathcal{O}$, and moreover this is a P-normal presentation of $\mathfrak{a}\mathfrak{b}$.

3. If $\mathfrak{a} = a\mathcal{O} + \alpha\mathcal{O}$ is a P-normal presentation for the integral ideal $\mathfrak{a}$, then there exists an integer k coprime to all primes in P (and hence coprime to a), such that $\mathfrak{a}^{-1} = \mathcal{O} + \alpha^{-1}k\mathcal{O}$. For instance, for any positive integer r in $\alpha\mathcal{O}$ the largest divisor k of r that is coprime to a will do.

Notion of Size

For a number field K of degree d, we fix the $\mathbb{Z}$-basis $\Omega = \{\omega_1, \omega_2, \ldots, \omega_d\}$ for the maximal-order $\mathcal{O}$ with $\omega_1 = 1$. We can represent an integral element $\alpha \in \mathcal{O}$ with respect to the integral basis Ω as explained in Definition 1.3.21. Now, we define the size of elements and ideals as follows.

Definition 1.3.25 (Size of Elements). *We define the size of an integral element $\alpha \in \mathcal{O}$ with respect to the chosen integral basis Ω as* $\mathsf{size}(\alpha) = d(\log(\|\alpha\|_\varphi))$. *For an $\alpha \in K$, $\alpha = \tilde{\alpha}/k$ with $k \in \mathbb{Z}_{>0}$ we define* $\mathsf{size}(\alpha) = \mathsf{size}(\tilde{\alpha}) + \log(k)$, *and $\alpha = \tilde{\alpha}/k$ with algebraic denominator $k \in \mathcal{O}$, we define* $\mathsf{size}(\alpha) = \mathsf{size}(\tilde{\alpha}) + \mathsf{size}(k)$. *When $\alpha \in \mathcal{F}$ is represented as a tuple $(\tilde{\alpha}, \ell) \in \mathcal{O}^2$ such that $\alpha = \tilde{\alpha}/\ell$ we define* $\mathsf{size}(\alpha) = \mathsf{size}(\tilde{\alpha}) + \mathsf{size}(k)$.

Considering the multiplicative structure of $\mathcal{O}$, we can give the size bounds with respect to multiplicative operations as follows.

Proposition 1.3.26. *For $\alpha, \beta \in \mathcal{O}$, the following holds.*

(i) $\mathsf{size}(\alpha\beta) \leq \mathsf{size}(\alpha) + \mathsf{size}(\beta) + d\log(dc_1c_2)$.

(ii) $\mathsf{size}(\alpha^{-1}) \leq d\mathsf{size}(\alpha) + d^2\log(dc_1) + d\log(c_2)$

Proof. (i) : By Lemma 1.3.22 the norm $\|\cdot\|_\varphi$ satisfies $\|\alpha\beta\|_\varphi \leq dc_1\sqrt{c_2}\|\alpha\|_\varphi\|\beta\|_\varphi$. Therefore it holds: $d\log(\|\alpha\beta\|_\varphi) \leq d\log(\|\alpha\|_\varphi) + d\log(\|\beta\|_\varphi) + d\log(dc_1\sqrt{c_2})$.

(ii) : For $\alpha \in \mathcal{O}$, let $\alpha^{-1} = \beta/k$ with $k \in Z_{>0}$ the denominator of α^{-1} and numerator $\beta \in \mathcal{O}$. We have that $|k| = |\operatorname{den}(\alpha^{-1})| \leq |N(\alpha)|$. Using (1.3.4) $\|\alpha\|^2/d \geq (N(\alpha))^{2/d}$, we obtain $\log(k) \leq d\log(\|\alpha\|_\varphi)$.

Considering the embeddings for $i = 1, \ldots, d$ and combining with Inequality (1.3.3) we have:

$$|\beta^{(i)}| = \frac{|k^{(i)}|}{|\alpha^{(i)}|} = \frac{|k|}{|\alpha^{(i)}|} \leq \frac{|N(\alpha)|}{|\alpha^{(i)}|)} = \prod_{j\in\{1,\ldots,d\}/i} |\alpha^{(j)}| \leq T_2(\alpha)^{(d-1)/2}$$

That is $T_2(\beta) = \sum_{i=1}^d |\beta^{(i)}|^2 \leq dT_2(\alpha)^{d-1}$:

$$\begin{aligned} T_2(\beta) &\leq d(T_2(\alpha))^{d-1} \\ \frac{1}{c_2}\sum_{i=1}^d |\beta_i|^2 &\leq d\left(c_1\sum_{i=1}^d |\alpha_i|^2\right)^{d-1} \\ \sum_{i=1}^d |\beta_i|^2 &\leq dc_2\left(c_1\sum_{i=1}^d |\alpha_i|^2\right)^{d-1} \\ \|\beta\|_\varphi &\leq \sqrt{c_2 d}\left(\sqrt{c_1 d}\|\alpha\|_\varphi\right)^{(d-1)} \end{aligned}$$

Therefore the size of α^{-1} is

$$\begin{aligned} d\log(\|\beta\|_\varphi) + \log(k) &\leq d\Big((d-1)(\log(\|\alpha\|_\varphi) + \log(c_1d)/2) + \log(c_2d)/2\Big) + d\log(\|\alpha\|_\varphi) \\ &\leq d^2\log(\|\alpha\|_\varphi) + d^2\log(dc_1) + d\log(c_2) \end{aligned}$$

□

Concerning the increase of the size of elements with multiplicative operations we define the constant C_Ω which depends on the basis Ω:

$$C_\Omega = \max\{d\log(dc_1c_2), d^2\log(dc_1) + d\log(c_2)\}.$$

Now we extend these relations for the multiplicative structure over K (for proofs and more details, see [BFH17, Chapter 3]).

Proposition 1.3.27. *For all $\alpha, \beta \in K$ and $m \in \mathbb{Z}$ the following holds:*

(i) $\mathsf{size}(m\alpha) = \mathsf{size}(\alpha) + d\log(|m|)$.

(ii) $\mathsf{size}(\alpha\beta) \leq \mathsf{size}(\alpha) + \mathsf{size}(\beta) + C_\Omega$.

(iii) $\mathsf{size}(\alpha^{-1}) \leq d\mathsf{size}(\alpha) + C_\Omega$.

(iv) $\mathsf{size}(\alpha + \beta) \leq 2(\mathsf{size}(\alpha) + \mathsf{size}(\beta))$.

An integral ideal $\mathfrak{a}$ of K is a free $\mathbb{Z}$-submodule of $\mathcal{O}$ with rank d, and it can be represented by a *basis matrix* $M_\mathfrak{a} \in \mathbb{Z}^{n\times n}$ with respect to the basis Ω. Therefore the size required to store the ideal with the $\mathbb{Z}$-basis presentation is bounded by $d^2\log(\|M_\mathfrak{a}\|_\infty)$. Note that, $\|M_\mathfrak{a}\|_\infty = \min\{a \in \mathbb{Z}_{>0} \mid a \in \mathfrak{a}\}$, and denote it by $\min(\mathfrak{a})$. With the two-element presentation of ideals, we can save the storage as explained in the following definition.

Definition 1.3.28 (Size of Ideals). *For an integral ideal $\mathfrak{a}$ in $\mathbb{Z}$-basis presentation, the size of $\mathfrak{a}$ is defined as* $\mathsf{size}(\mathfrak{a}) = d^2\log(\min(\mathfrak{a}))$. *If $\mathfrak{a} = \tilde{\mathfrak{a}}/k \trianglelefteq \mathcal{F}$ is a fractional ideal where $\tilde{\mathfrak{a}}$ is integral and $k \in \mathbb{Z}_{>0}$ is the denominator of $\mathfrak{a}$, we define the size of $\mathfrak{a}$ by* $\mathsf{size}(\mathfrak{a}) = \mathsf{size}(\tilde{\mathfrak{a}}) + d^2\log(k)$. *With the two-elements presentation of the ideal, the size of $\mathfrak{a} \trianglelefteq \mathcal{O}$ is* $\mathsf{size}(\mathfrak{a}) = (d+1)\log(\min(\mathfrak{a}))$, *and when $\mathfrak{a} \trianglelefteq \mathcal{F}$ the size is* $\mathsf{size}(\mathfrak{a}) = (d+1)\log(\min(\tilde{\mathfrak{a}})) + \log(k)$.

Considering the properties of two-elements presentation, we can provide the size bounds with respect to ideal operations as follows:

Proposition 1.3.29. *Let $\mathfrak{a}, \mathfrak{b}$ be fractional ideals of K.*

(i) $\mathsf{size}(\mathfrak{ab}) \leq \mathsf{size}(\mathfrak{a}) + \mathsf{size}(\mathfrak{b})$.

(ii) $\mathsf{size}(\mathfrak{a}^{-1}) \leq (d+2)\log(\min(\mathfrak{a}))$.

Proof. We take $\mathfrak{a} = \tilde{\mathfrak{a}}/k$ and $\mathfrak{b} = \tilde{\mathfrak{b}}/l$ with k and l the denominators of $\mathfrak{a}$ and $\mathfrak{b}$ respectively. Let $\tilde{\mathfrak{a}} = \tilde{a}\mathcal{O} + \tilde{\alpha}\mathcal{O}$ and $\tilde{\mathfrak{b}} = \tilde{b}\mathcal{O} + \tilde{\beta}\mathcal{O}$ be in P-normal presentation,

(i) We have that $\tilde{\mathfrak{a}}\tilde{\mathfrak{b}} = \tilde{a}\tilde{b}\mathcal{O} + \tilde{\alpha}\tilde{\beta}\mathcal{O}$, therefor $\mathsf{size}(\tilde{\mathfrak{a}}\tilde{\mathfrak{b}}) = \mathsf{size}(\tilde{\alpha}\tilde{\beta}) + \mathsf{size}(\tilde{a}\tilde{b})$. Since $\min(\tilde{\mathfrak{a}}\tilde{\mathfrak{b}})$ divides $\min(\tilde{\mathfrak{a}})\min(\tilde{\mathfrak{b}})$, we have $\mathsf{size}(\tilde{\mathfrak{a}}\tilde{\mathfrak{b}}) \leq (d+1)\log(\min(\tilde{\mathfrak{a}})\min(\tilde{\mathfrak{b}})) = \mathsf{size}(\tilde{\mathfrak{a}}) + \mathsf{size}(\tilde{\mathfrak{b}})$.

Let $\mathfrak{a}, \mathfrak{b} \trianglelefteq \mathcal{F}$, then $\mathsf{size}(\mathfrak{a}) = \mathsf{size}(\tilde{\mathfrak{a}}) + \log(k)$ and $\mathsf{size}(\mathfrak{b}) = \mathsf{size}(\tilde{\mathfrak{b}}) + \log(l)$. In P-normal presentation let $\mathfrak{a} = a\mathcal{O} + \alpha/k\mathcal{O}$ and $\mathfrak{b} = b\mathcal{O} + \beta/l\mathcal{O}$. Then $\mathfrak{ab} = ab\mathcal{O} + \alpha\beta/kl\mathcal{O}$ and it holds $\mathsf{size}(\mathfrak{ab}) \leq \mathsf{size}(\mathfrak{a}) + \mathsf{size}(\mathfrak{b})$, similar to the integral ideal case.

(ii) We have that $\min(\tilde{\mathfrak{a}}) \in \tilde{\mathfrak{a}}$. Thus the principal ideal $\langle\min(\tilde{a})\rangle$ can be divided by $\tilde{\mathfrak{a}}$ and there exists an integral ideal $\mathfrak{c}$ with $\langle\min(\tilde{a})\rangle = \tilde{\mathfrak{a}}\mathfrak{c}$, that is $\tilde{\mathfrak{a}}^{-1} = \mathfrak{c}/\min(\tilde{\mathfrak{a}})$. Since $\min(\tilde{\mathfrak{a}}) \in \mathfrak{c}$, it holds $\min(\mathfrak{c}) \leq \min(\tilde{a})$. As $\min(\tilde{\mathfrak{a}})$ is the denominator of $\tilde{\mathfrak{a}}^{-1}$, we have $\mathsf{size}(\tilde{\mathfrak{a}}^{-1}) = \mathsf{size}(\mathfrak{c}) + \log(\min(\tilde{\mathfrak{a}}))$. That is $\mathsf{size}(\tilde{\mathfrak{a}}^{-1}) \leq (d+2)\log(\min(\tilde{a}))$.

When $\mathfrak{a} \trianglelefteq \mathcal{F}$, we have that $\mathfrak{a}^{-1} = k\tilde{\mathfrak{a}}^{-1}$. Since the denominator of $\mathfrak{a}^{-1}$ is equal to $\min(\mathfrak{a})$ it holds that $\mathsf{size}(\mathfrak{a}^{-1}) = \mathsf{size}(\tilde{\mathfrak{a}}^{-1}) + \log(\min(\mathfrak{a}))$. Thus $\mathsf{size}(\mathfrak{a}^{-1}) \leq (d+2)\log(\min(\mathfrak{a}))$.

□

For $A \in \mathcal{O}^{n\times n}$ consider the largest absolute value of coefficients of A as $S_A = \log(\|A\|_\varphi)$ and let B be an upper bound on $\log(\|\det(A)\|_\varphi)$. By Lemma 1.3.23, we have that $\|\det(A)\|_\varphi \leq (c_2^{1/2}(c_1 nd)^{n/2})\|A\|_\varphi^n$. Therefore we have that $B \in n(C\log(nd) + S_A)$ for some $C \in \mathbb{R}$ and $B = O^\sim(nS_A)$. That is $\mathsf{size}(\det(A)) \in O^\sim(ndS_A)$.

Consider the storage requirements for $\mathcal{O}$-module M (see Section 2.4.2, for a matrix $A \in \mathcal{O}^{n\times m}$ and vector of fractional ideals $(\mathfrak{a}_i)_{1\leq i\leq n} \subseteq \mathcal{F}^n$, $M = \sum_{i=1}^{n} \mathfrak{a}_i A_i$, where A_i are the rows of A). It takes nmd integers to store the basis A: $A_1, \ldots, A_n$ and $n(d+2)$ integers to store generators for the ideals $\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_n$ using two element representation. If we have picked an integral basis for $\mathcal{O}$, then it takes $2nd^2$ integers to store the basis itself. Hence, the total storage space required to store M is $nmd + 2nd^2$ integers.

Cost Function

For a number field of degree d, we define the non-decreasing function $s \mapsto M_d(s)$ and an algorithm which multiplies two number field elements in K of size at most s using at most $M_d(s)$ bit operations. For multiplication of two elements $\alpha, \beta \in K$, we take $s = \max\{\mathsf{size}(\alpha), \mathsf{size}(\beta)\}$. We assume that $M_d(s) = 1$ for $s \leq 1$, so that $M_d(s) \geq 1$ for all values of s. We will also have the following assumptions on M_d:

1. $M_d(s) \in O^{\sim}(s)$.
2. $M_d(s_1 s_2) \leq M_d(s_1) M_d(s_2)$ for all $s_1, s_2 \in \mathbb{R}$.

The classical method has $O(d^2)$ number of bit operations, for multiplication. In our applications, we use asymptotically fast algorithms such as FFT-based methods which has the bit complexity as $O(d \log(d) \log(\log(d)))$. Since we use $O^{\sim}$ asymptotic notation in complexity analysis, we assume that for Euclidean algorithm related computations has the same complexity as $M_d(s)$ for multiplication.

Consider the complexity for inversion of number field elements. Let $\alpha \in \mathcal{O}$ and f be the defining polynomial of the number field K. Then, we can obtain α^{-1} using the Bezout coefficients β, γ such that $1 = \beta\alpha + \gamma f$. The output is larger by a factor of d, therefore the total complexity for α^{-1} computation is $M_d(\beta) \approx d M_d(s)$, where $s = \mathsf{size}(\alpha)$.

We use two-elements representation of ideals in our computations. Therefore, the complexity for ideal arithmetic follows from the properties of the normal presentation.

For linear algebra such as matrix multiplication, inverse computation, normal form computations we use classical method with complexity $O^{\sim}(n^3 M_d(s_A))$ for $s_A = d \log(\|A\|_{\varphi})$. Note that, for multiplication of two $n \times n$ matrices over $\mathbb{Z}$, the algorithm of Strassen in [SS71] has the complexity of $O(42 n^{\log_2 7})$, and asymptotically fastest known method allows $O(n^{2.376})$.

Chapter 2

Algorithms for Computing the Determinant of a Matrix over Number Fields.

We present a fast and practical deterministic algorithm for computing the determinant of a non-singular $n \times n$ matrix A over a number field K of degree d. Based on Storjohann's [PS13] determinant computation algorithm for integer matrices, we first solve $Ax = b$ for a randomly chosen vector b, using Dixon's algorithm [Dix82]. It can be seen from Cramer's rule that the denominators inherent in the solution are the determinant or a large divisor of the determinant. When generalizing this to number rings, we encounter the problem that the denominator is an ideal and it is no longer a number which can be used to compute the determinant. To tackle this problem, we use the approach of Fieker and Friedrichs [FF00]. We rigorously assess the complexity of this new algorithm as $O^{\sim}(n^4d + n^3d + nd^5)$ operations, whereas the usual Gaussian method takes $O^{\sim}(n^5d^2)$ operations in $\mathbb{Q}$.

2.1 Introduction

Computation of the determinant of a non-singular $n \times n$ integer matrix is a well-studied classical problem. The classical Gaussian elimination method takes a cubic number of operations. Here, we face the computational phenomenon of *intermediate coefficient swell*, and the standard strategy to avoid this is to use modular methods. Using a Chinese Remainder Theorem (CRT) based modular method, one can compute the determinant modulo many small prime numbers such that the product of the primes is large enough. This computation takes $O^{\sim}(n^3)$ number of operations in $\mathbb{Q}$, with the size of determinant $O^{\sim}(n)$ the total complexity grows to $O^{\sim}(n^4)$.

Over time, the complexity of the determinant computation has been reduced. The first breakthrough was the division-free algorithm by Kaltofen [Kal92]. The state-of-the-art method is due to Storjohann [PS13], extending his technique in [Sto03] using high-order lifting and integrality certification. The run-time of this heuristic algorithm grows to $O^{\sim}(n^3)$ approximately. His algorithm is based on the determinant algorithm of Abbott, Bronstein and Mulders in [ABM99, Section 2]. To compute the determinant of $A \in \mathbb{Z}^{n\times n}$, first they compute a large divisor of the determinant. This is done by solving linear systems $Ax = b$ over the rationals for a randomly chosen vectors b. Then, a divisor of the determinant (more precisely the largest elementary divisor of the matrix) can be found from the minimal $d \in \mathbb{Z}_{>0}$, such that $dA^{-1}b$ is integral. This method can be understood by the modules (or latices) spanned by the rows of the matrices. The module corresponding to the rows of A is enlarged through the addition of random elements, and in each such step gives a factor of the determinant. In the final step, one

needs to verify that the module is trivial, i.e. the basis matrix has determinant one, which is called unimodularity certification. The idea is that the module is trivial if the matrix has an integral inverse. Otherwise he repeats the same steps for a new matrix b. In Section 2.2, we discuss these determinant computation algorithms for integer matrices.

Section 2.3 provides a fast algorithm for computing determinants over number fields. Here, we encounter the problem that there is no well-defined algebraic denominator. In fact, the denominator we obtain from the solution matrix is an ideal containing all algebraic numbers that can act as a denominator. Since we have ideals as divisors, there are no unique representations for fractions. We define the denominator ideal with respect to a solution $S \in K^n$ of the linear system $Ax = b$ as $\langle\{\mu \in O \mid \mu S \in O^n\}\rangle$. While the denominator ideal can be computed, it cannot immediately serve as an approximation to the determinant. We combine this ideal with some CRT-based computation to obtain the determinate using the LLL algorithm. This new deterministic algorithm uses $O^{\sim}(n^4d + n^3d + nd^5)$ operations in the worst case, whereas the Gaussian method uses $O^{\sim}(n^5d^2)$ operations in $\mathbb{Q}$, to compute the determinant of a non-singular $n \times n$ matrix over the number field K of degree d.

Finally, in Section 2.4, we generalizing Storjohann's idea to the number ring situation. The problem here is that number rings are not Euclidean domains. Moreover, module generated by the rows of a matrix over number fields cannot be used as free-modules to apply the same theory as Storjohann does. Therefore, we use the theory of pseudo-matrices to overcome this problem. However, we could not use his the unimodular certification approach in [PS12], as it was slow in our case. Therefore, we used a different approach to verify the determinant and obtain better runtimes.

2.2 Determinant computation of an Integer Matrix

2.2.1 Modular Determinant Algorithm by Abbott et el.

The determinant Algorithm 4 of Abbott, Bronstein and Mulders [ABM99, Section 2] uses two variants of modular methods. In the first step, it solves linear systems using Dixon's algorithm [Dix82], which is based on Hensel lifting. Then, it uses a CRT-based modular technique to find the determinant of a matrix A. The algorithm involves two reconstruction steps: Step-1 requires a reconstruction method to find a rational solution from the lifting output, and Step-4 reconstruct the integer determinant from the CRT output. There are different methods of doing the reconstruction in each case. The Algorithm 4 uses Hadamard's bound H: if the prime product $m \geq 2H$, it returns a unique result $d = \det(A)$.

Algorithm 4 Determinant (`Determinant`)

Input: $A \in \mathbb{Z}^{n\times n}$.
Output: $\det(A)$.

1: Using Dixon lifting and rational reconstruction, compute solutions to $Ax = b$ for several matrices $b \in \mathbb{Z}^{n\times 1}$ where the entries are chosen uniformly randomly from a suitable interval $I = [m, m + 1, \ldots, m + r] \subset \mathbb{Z}$.
2: Let D be the LCD (least common denominator) of all the solutions.
3: Compute H: a bound for the absolute value of $\det(A)$ using Hadamard's bound.
4: Compute $\det(A)/D$ using CRT: start with values and primes used in Step 1. Since $|\det(A)/D| \leq H/D$, use $2H/D$ as the bound for the CRT modulus.
5: Compute $D \times \det(A)/D$.
6: **return** $\det(A)$

It is clear from Algorithm 4 when the factor D, computed in Step 2, is small; we have to perform CRT in Step 4, for many primes (i.e. when $A = \mathrm{diag}(a, a, \ldots, a)$ for some $a \in \mathbb{Z}$). In that case, the complexity of the algorithm is $O^{\sim}(n^4)$ operations in $\mathbb{Q}$. However, in general, the expected complexity for a randomly chosen input matrix A is $O^{\sim}(n^3)$, as we can obtain the complete determinant, either without or with only few CRT steps (See [ABM99, Section 4]).

2.2.2 Storjohann's Determinant Algorithm

Based on the Algorithm 4, Storjohann developed a new algorithm in [PS13], with the runtime approximately being $O^{\sim}(n^3)$ operations in $\mathbb{Q}$. In order to discuss his algorithm , we need the following two definitions regarding matrix cannonical forms.

Matrix Normal Forms

Definition 2.2.1 (Hermite normal form)**.** *A nonsingular matrix $H = [h_{ij}] \in \mathbb{Z}^{n\times n}$ is said to be in Hermite normal form if the following conditions are satisfied.*

- *H is upper triangular*
- *$h_{ii} > 0$, for $1 \le i \le n$*
- *$0 \le h_{ij} \le h_{ii}$ for all $1 \le i \le n$ and $i < j \le n$.*

Definition 2.2.2 (Elementary Divisors)**.** *For a matrix $A \in \mathbb{Z}^{n\times n}$ and $1 \le i \le n$, the i-th determinantal divisor α_i is the greatest common divisor of all $i \times i$ minors of A. The ratios of successive determinantal divisors are the elementary divisor (invariant factors) $s_i = \alpha_i/\alpha_{i-1}$, with $\alpha_0 = 1$ for convenience.*

The first determinantal divisor α_1 is the greatest common divisor of the entries of A and the n-th divisor α_n is the determinant $|\det(A)|$ itself. The diagonal matrix $S = \mathrm{diag}(s_1, s_2, \ldots, s_n)$ is the *Smith normal form* of A.

The Algorithm

The steps of determinant algorithms in general:

1. Solve the linear system $Ax = b$ for a randomly chosen vector $b \in \mathbb{Z}^{n\times 1}$. Then, the LCD is computed as before.
2. - Either proving the denominator is the determinant:
     Construct a matrix C from the solution such that, $\det(C) =$ denominator and $B = AC^{-1}$ is integral. If the new matrix B is unimodular, the determinant is complete. (This does not work immediately for non-random diagonal matrices (i.e. $A = \mathrm{diag}(a, a, \ldots, a)$) as the denominator is always a proper divisor of the determinant. Then we have to continue with CRT method.)
   - or, compute the determinant modulo more primes using CRT until the prime product achieve enough precision with respect to Hadamard's bound.

Definition 2.2.3. *Given $v \in \mathbb{Q}^{n\times 1}$, a minimal triangular denominator of v is a nonsingular upper triangular matrix $T \in \mathbb{Z}^{n\times n}$ with minimal magnitude determinant such that Tv integral.*

In Storjohann's approach, he repeats Step 1 until the new matrix B becomes unimodular. First he computes a *minimal triangular denominator* T, such that $TA^{-1}b$ is integral. The `hcol` algorithm in [PS13, Figure 1]) constructs the matrix T based on Lemma 2.2.4.

Lemma 2.2.4. *[PS13, Lemma 1] Let $s \in \mathbb{Q}^{n\times 1}$ and $d \in \mathbb{Z}_{>0}$ be such that $w := ds$ is integral. Write the Hermite form of $\left[\begin{array}{c|c} d & \\ \hline w & I_n \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}$ as $\left[\begin{array}{c|c} * & * \\ \hline & H \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}$, where $I_n \in \mathbb{Z}^{n\times n}$ is the identity matrix. Then $H \in \mathbb{Z}^{n\times n}$ is a minimal triangular denominator of s.*

Let s be the solution of the linear system $Ax = b$, and T be the minimal triangular denominator of s. Then, one can obtain AT^{-1} as an integral matrix. Here the idea (which will be used over number fields) is that AT^{-1} is also a basis for the module generated by the rows of the matrix $[A \mid b]^t$. This way a matrix C is constructed from the product of matrices T_i for the solutions of different b_i matrices for $i = 1, 2, \ldots, k$, until the new matrix $B = AT_1^{-1}T_2^{-1}\cdots T_k^{-1}$ is unimodular. Roughly speaking, each projection corresponds to a single elemetary divisor of the matrix A and the largest remaining elementary divisor of the matrix B. If B is unimodular, then the determinant is complete, and the sign of the determinant can be recovered by computing $\det(A) \mod p$ for a small odd prime p. Unimodular certification is due to `UniCert` algorithm in [PS12, Section 4].

2.3 Algorithm for Computing the Determinant of a Matrix over Number Field

Similar to the integer case, we start with a solution S of a linear system $Ax = b$ using `DixonSolver` in Section 1.3.7. Then we compute the denominator of the solution as an ideal: we take the intersection of principal ideals generated by each coordinate of S as $\mathfrak{C}$. There is a unique factorization for ideals as $\mathcal{O}$ is a unique factorization domain. Hence, $\mathfrak{C}$ can be represented as a quotient of two integral ideals $\mathfrak{C} = \mathfrak{n}/\mathfrak{d}$: denominator ideal $\mathfrak{d}$ and numerator ideal $\mathfrak{n}$. Since we can obtain the required denominator ideal after a few intersections without computing the intersection of all the ideals, we use a modified algorithm in the implementation.

Algorithm 5 CRT of ideals (`CRT_Ideal`)

Input: Two tuple of elements in coprime ideals: (r_1, i_1) and (r_2, i_2).
Output: x such that $x \equiv r_1 \mod i_1$ and $x \equiv r_2 \mod i_2$
1: Find elements $y_1 \in i_1$ and $y_2 \in i_2$ such that $y_1 + y_2 = 1$ using idempotents.
2: **return** $r_1y_2 + r_2y_1$

Algorithm 6 Determinant Computation (`ModularDeterminant`)

Input: $A \in K^{n\times n}$ and a starting prime $p \in \mathbb{Z}$.
Output: Determinant of matrix A.

1: $\mathcal{O}$ = maximal order of K and d = degree(K).
2: B = random($n \times 1$) matrix over K, for a suitable range of coefficients.
3: S = `DixonSolver`(A, b)
4: $\mathfrak{d}$ = Denominator ideal of S.
5: D_M = LLL reduced basis matrix of $\mathfrak{d}$.
6: $D_I = {D_M}^{-t}$
7: F = LLL reduced basis of $\mathfrak{d}$
8: $d_0 = d_1 = \mathcal{O}(0)$
9: $p_1 = 1$
10: **while** true **do**
11: d_p = det(A mod p)
12: $\mathfrak{p}$ = principal ideal of p.
13: d_1 = `CRT_Ideal`$((d_1, \mathfrak{d}), (d_p, \mathfrak{p}))$
14: $\mathfrak{d} = \mathfrak{d}\mathfrak{p}$
15: $p_1 = p_1 p$
16: T = the column vector of the cordinates of $\mathfrak{d}$.
17: $W = D_I\, T$
18: $k = \sum_{i=1}^{d} F_i \left\lfloor \frac{W_{i1}}{p_1} \right\rceil$
19: $k = p_1\, k$
20: $d_1 = d_1 - k$
21: **if** $d_0 = d_1$ **then**
22: **return** d_1
23: **end if**
24: $d_0 = d_1$
25: p = a new prime number.
26: **end while**

Considering the denominator ideal, we have that $\det(A) \equiv 0 \pmod{\mathfrak{d}}$. Next, we compute the determinant modulo some prime ideals $\mathfrak{p}_i$ for $i = 1, 2, \ldots$ until we get the correct determinant using CRT and algebraic reconstruction from a solution modulo $(\mathfrak{d} \prod \mathfrak{p}_i)$.

In Algorithm 6, we use the constructive version of the CRT (Algorithm 5) introduced by Cohen in [Coh96]. In order to use CRT for ideals, the construction of idempotents has to be made explicit as in [Coh96, Prop 1.1].

The recovered determinant from the modular result can be guaranteed by Theorem 1.3.10, if the lattice corresponding to the ideal $(\mathfrak{d} \prod \mathfrak{p}_i)$ is large enough. In order to estimate the size of the ideal, we compute a bound B for the determinant of the matrix A, using Hadamard's bound for conjugate matrices of A (As described in Section 1.3.8). Let $\mathfrak{I} = \mathfrak{d} \prod \mathfrak{p}_i$ be the ideal computed in Step 14 of the Algorithm 6.

Theorem 2.3.1. *Given bound for the determinant $T_2(\det(A)) \leq B$, suppose the norm of the ideal $\mathfrak{I}$ satisfies*

$$N(\mathfrak{I}) \geq (2\sqrt{B/d} \cdot (3\sqrt{\gamma(\theta)}/2)^{d-1})^d.$$

Then, Algorithm 6 computes the correct determinant of the matrix A.

Proof. Suppose the norm of the ideal $\Im$ computed in Step 14 satisfies Belabas' condition as above. Let d_1 be the CRT output in step 13, such that $\det(A) \equiv d_1 \pmod{\Im}$. Given $T_2(\det(A)) \leq B$, Theorem 1.3.10 guarantees that the reconstructed solution $d_1 - k$ is the correct determinant of A.

Reconstruction is clear from the Algorithm 6: Let $\mathfrak{d}$ be the denominator ideal, then we can compute the LLL reduced basis F of $\mathfrak{d}$ (Step 4 and 7). Let p_1 be the product of prime numbers used in the computation and take $\mathfrak{p}_1 = \prod \mathfrak{p}_i$ for the corresponding prime ideal. Using the fact that $\Im = \mathfrak{d} \prod \mathfrak{p}_i$, a LLL reduced basis of the ideal $\Im$ is $p_1 \cdot F$. We compute the coordinate matrix W/p_1 of d_1 with respect to the basis $p_1 \cdot F$, using the basis matrix D_M of $\mathfrak{d}$ (in Step 17). k is computed in Step 18 by rounding coordinates in the matrix W/p_1. In the algorithm, we do multiplication by p_1 at the end to change the basis from F to $p_1 \cdot F$. Finally, $d_1 - k$ gives $\det(A)$. □

Theorem 2.3.2. *Overall complexity of the Algorithm 6 is* $O^{\sim}(n^4 d + n^3 d + n d^5) \log(\|A\|_\varphi)$ *number of bit operations.*

Proof. The algorithm has two main steps: linear system solving and reconstructing the determinant using denominator of the solution. For the complexity analysis, we have used asymptotically fast operations over K, but classical complexities for linear algebra. Let $s_A = d \log(\|A\|_\varphi)$ and B be an upper bound on $d \log(\|\det(A)\|_\varphi)$, then we have that $\mathsf{size}(\det(A)) \leq B \in O^{\sim}(n s_A)$ from Section 1.3.9.

Linear system solving at Step 3:

i. Computing $A^{-1} \pmod{p}$ takes $O^{\sim}(n^3 M_d(d \log(p)))$ bit operations:

   Because, the addition and multiplication operations for number field elements modulo p costs $O^{\sim}(M_d(d \log(p)))$ and, row column operations for the matrix $A \pmod{p}$ costs $O^{\sim}(n^3)$ over K. We can choose p such that $M_d(d \log(p)) \leq M_d(s_A)$ to make it easier for complexity analysis.

ii. Dixon solver computes the p-adic expansion of the solution with $O^{\sim}(n^3 M_d(s_A))$ bit complexity:

   One lifting iteration contains matrix-vector multiplications which cost $O^{\sim}(n^2 M_d(s_A))$. The number of iteration required to obtain the solution is $O^{\sim}(n)$ (See [Dix82]).

iii. Vector reconstruction to recover the solution takes $O^{\sim}(d^5 (B/d)^2)$ bit operations for random input matrices (as mentioned in the Theorem 1.3.20, using L^2-algorithm). Worst case the bit complexity would be $O^{\sim}(n d^5 (B/d)^2)$, which is equalent to $O^{\sim}(n^2 d^5 (s_A/d)^2)$.

Recovering the determinant:

i. The step 5 is using L^2 algorithm to find the reduced basis of the determinant ideal. It has the bit complexity $O^{\sim}(d^5 (B/d)^2)$. That is equal to $O^{\sim}(n d^5 (s_A/d)^2)$.

ii. The last step, uses CRT which costs $O^{\sim}(n^3 M_d(P))$ bit operations, to achieve the remaining number of digits P (as the bound for the determinant): that is we need primes such that the size of the prime product p_1 satisfies $P = d \log(p_1) \geq (B/d \log(\|\min(\mathfrak{d})\|_\varphi))$. For random matrices the size of P is negligible. But worst case $P = B$, and the complexity grows as $O^{\sim}(n^3 M_d(B))$ (that is $O^{\sim}(n^4 M_d(s_A))$), in the presence of non-random (diagonal) matrices.

The total bit complexity of the algorithm in the worst case is $O^{\sim}((n^4 + n^3) M_d(s_A) + n^2 d^5 (s_A/d)^2)$. That is $O^{\sim}(n^4 d + n^3 d \log(\|A\|_\varphi) + n^2 d^5 \log^2(\|A\|_\varphi))$. But, for generic random matrices the bit complexity would be $O^{\sim}(n^3 d \log(\|A\|_\varphi) + n d^5 \log^2(\|A\|_\varphi))$. □

While the new implementation has complexity as above, the direct Gaussian method has $O^{\sim}(n^5 d^2 s_A)$ bit operations, because the row and column operations takes $O^{\sim}(n^3 d^2)$ operations, and inputs are as large as the size of the matrix $O^{\sim}(n^2)$ (see [GG13, Sec 5.5, p.111]). The key point of the `ModularDeterminant`

algorithm is that the computations are not carried out over K, except the second part of the algorithm. The complexity $O^{\sim}(n^4d + n^3d + n^2d^5)$ is given with bit operations. The expensive lattice reduction is used only once for a basis matrix of the denominator ideal corresponding to the solution vector. Since we already have a maximum possible divisor of the determinant, the CRT step does not cost much. In the next section, we make the performance-wise comparison with [BFH17, Algorithm 3], which has bit complexity of $O^{\sim}(n^4d^4)$.

d	r / n	$-100:100$			$-1000:1000$			$-1000^2:1000^2$		
		MDet	Hecke	Quot	MDet	Hecke	Quot	MDet	Hecke	Quot
2	10	0.007	0.001	0.17	0.005	0.001	0.12	0.015	0.001	0.07
	30	0.027	0.022	0.81	0.035	0.030	0.85	0.112	0.050	0.45
	50	0.117	0.16	1.34	0.11	0.21	1.83	0.14	0.34	2.32
	100	0.61	2.32	3.81	0.80	3.09	3.88	0.84	6.11	7.30
	150	1.82	12.29	6.74	1.93	17.17	8.89	2.88	33.29	11.57
	300	14.65	4 m	16.10	15.74	5.6 m	21.19	19.69	11.6 m	35.26
	500	1.1 m	38 m	32.82	1.2 m	54 m	43.94	1.6 m	2 h	73.18
	1000	9.5 m	13.6 h	85.25	10.3 m	20.6 h	120.14	13. 2 m	-	∞
3	50	0.15	0.35	2.30	0.17	0.41	2.42	0.32	0.66	2.06
	100	1.06	4.91	4.66	1.24	6.02	4.88	1.89	11.48	6.07
	150	3.19	24.14	7.57	3.69	32.61	8.84	6.09	1 m	10.56
	300	22.61	7.5 m	20.02	27.68	10.6 m	23.05	37.65	22.3 m	35.52
	500	1.7 m	1.2 h	42.47	2 m	1.7 h	51.97	2.9 m	3.9 h	78.19
10	50	2.91	3.23	1.11	3.86	4.35	1.13	15.74	8.37	0.53
	100	15.47	55.15	3.56	24.88	75.58	3.04	1 m	2.5 m	2.33
	150	26.56	5.2 m	11.64	1 m	7.3 m	7.02	1.8 m	14.5 m	8.18
	300	2.6 m	1.8 h	40.66	3.5 m	2.6 h	43.96	4.6 m	5.2 h	67.57
20	10	9.32	0.03	0.00	8.97	0.04	0.00	9.84	0.07	0.01
	50	42.68	13.48	0.32	58.95	18.27	0.31	3.9 m	35.15	0.15
	100	3.3 m	4.3 m	1.29	4.8 m	5.8 m	1.20	10.3 m	10.7 m	1.03
	150	6.3 m	23.9 m	3.79	8.9 m	32 m	3.60	36.2 m	1 h	1.78
30	10	48.62	0.2	0.00	32.50	0.13	0.00	1.4 m	0.42	0.01
	50	5.7 m	52.32	0.15	5.8 m	56.13	0.16	25.9 m	1.8 m	0.07
	100	21.3 m	13 m	0.58	57 m	30 m	0.52	1.2 h	47.4 m	0.65
	150	38.2 m	1 h	1.67	52 m	1.4 h	1.55	1.7 h	2.4 h	1.41
50	10	4.8 m	0.62	0.00	5.6 m	0.48	0.00	48.1 m	1.13	0.00
	50	56.9 m	3.5 m	0.06	1.1 h	6.6 m	0.10	4.6 h	21.2 m	0.08

Table 2.1: Timing for Determinant Computation over Number Fields.

2.3.1 Performance Analysis Based on Timing

We have implemented Algorithm 6 (see [Sur21]) using Hecke [FHHJ17] in the Julia language [BEKS17]. To illustrate the efficiency of the new method, we have computed timings using number

fields $\mathbb{Q}[x]/(x^2+7x+1)$, $\mathbb{Q}[x]/(x^3+7x+1)$ and $\mathbb{Q}[x]/(f)$ for $f = x^{10}+\sum_{i=0}^{9} 2(-x)^i$, $f = x^{20}+\sum_{i=0}^{19} 2(-x)^i$, $f = x^{30}+\sum_{i=0}^{29} 2(-x)^i$ and $f = x^{50}+\sum_{i=0}^{49} 2(-x)^i$. Table 2.1 shows timing in seconds (m -for minutes and h- for hours) for the new algorithm `ModularDeterminant` (`MDet`) for different choices of parameters: d- degree of the number field, n- size of the matrix and r- range of the coefficients of input matrix. Comparison have been made with the existing implementation in Hecke based on the deterministic algorithm [BFH17, Algorithm 3], which uses modular methods and decomposition of rings. Experimental results (with quotient times) shows that the new algorithm works much better for big matrices in small degree fields: For 1000×1000 matrix the new implementation computes the determinant, nearly 120 times faster than the existing one. Moreover, over a field of degree 30, a 300×300 matrix with the size of coefficients $-100 : 100$ takes 1.8-hours to compute the determinant, whereas Hecke implementation does not terminate. The timings prove the complexity results: over fields with larger degrees, our implementation becomes slow due to the expensive basis reduction algorithm. Over fields of fixed degrees, the modular approach performs well for large matrices.

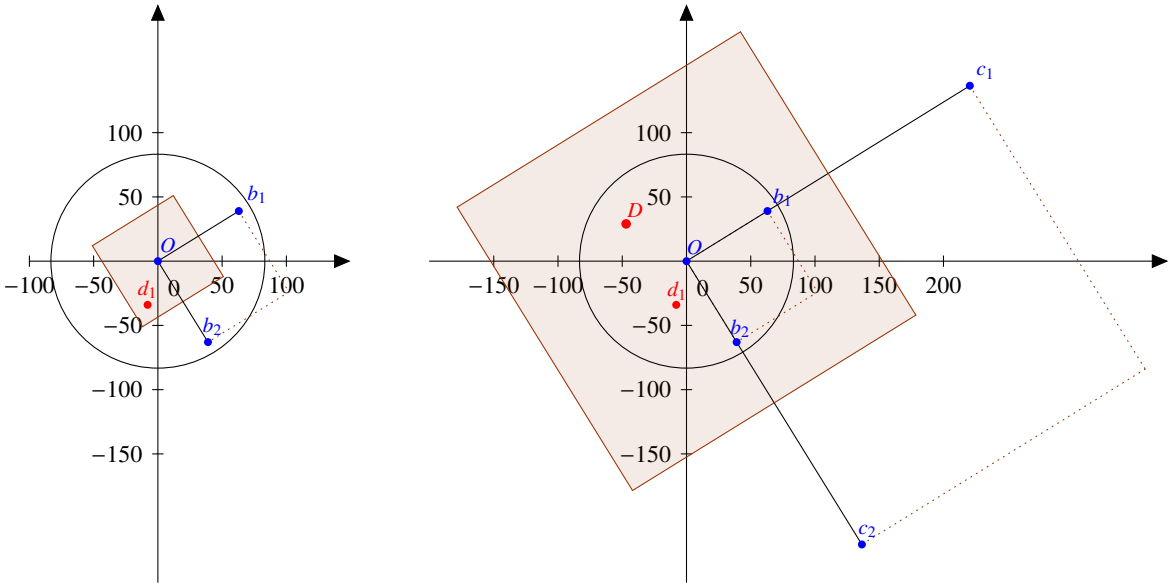
2.3.2 Example

Consider $K = \mathbb{Q}[a] = \mathbb{Q}[x]/x^2 + 1$ and maximal order $O = \langle 1, a\rangle$.

$$A = \begin{bmatrix} 7a+3 & 4a+4 \\ 2a+2 & 8a+3 \end{bmatrix} \quad b = \begin{bmatrix} 4a+3 \\ 7a+9 \end{bmatrix}, \quad \text{Solution} \quad S = \begin{bmatrix} \frac{443}{610}a + \frac{129}{610} \\ \frac{-271}{305}a + \frac{297}{305} \end{bmatrix}$$

We compute the determinant of the matrix A as follows: The denominator ideal of S is computed using Hecke as: $\mathfrak{B} = \langle 610, 135641a + 253507\rangle$.

Basis matrix of $\mathfrak{B}$ is: $\begin{bmatrix} 610 & 0 \\ 377 & 1 \end{bmatrix} \xrightarrow{\text{LLL}} \begin{bmatrix} 13 & -21 \\ 21 & 13 \end{bmatrix}$.



Therefore, we have that $\mathfrak{B}$: $\langle 21 + 13a, 13 - 21a\rangle$
Take $p_1 = 3$: Ideal $\mathfrak{p}_1 = \langle 3, 3a\rangle \triangleleft O$.
LLL basis for the ideal $\mathfrak{B}\mathfrak{p}_1$: $\langle B_1 = 63 + 39a, B_2 = 39 - 63a\rangle$.
Lattice basis of $\Delta(\mathfrak{B}\mathfrak{p}_1) = \langle b_1 = (63, 39), b_2 = (39, -63)\rangle$
Reconstruct determinant from CRT output:

$-1220 - 2440a \pmod{\mathfrak{B}\mathfrak{p}_1}\ d_1 = -1220 - 2440a - (-31B_1 + 19B_2) = -8 - 34a$

Take $p_2 = 7$: Ideal $\mathfrak{p}_2 = \langle 7, 7a\rangle \triangleleft O$.
LLL basis for the ideal $\mathfrak{B}\mathfrak{p}_1\mathfrak{p}_2$: $\langle C_1 = 441 + 273a, C_2 = 273 - 441a\rangle$
Lattice basis of $\Delta(\mathfrak{B}\mathfrak{p}_1\mathfrak{p}_2) = \langle c_1 = (441, 273), c_2 = (273, -441)\rangle$
CRT output: $-36608 - 128134a \pmod{\mathfrak{B}\mathfrak{p}_1\mathfrak{p}_2}$
Determinant: $D = -36608 - 128134a - (-190C_1 + 173C_2) = -47 + 29a$

2.4 Extension to Storjohann's Determinant Algorithm

When generalizing Storjohann's determinant computation algorithm (Section 2.2) to the number ring O, we encounter the problem that finitely generated torsion-free O-modules are not free in general. For this reason, matrix normal forms are more subtle than in the principal ideal domain case. Hence, we cannot directly apply Lemma 2.2.4 to compute a minimal triangular denominator of a solution matrix.

2.4.1 Finitely Generated Modules over Dedekind Rings

In this section, we give an overview of modules over Dedekind rings in general. The results given at the end of this section will be used in next sections with pseudo-normal forms, to compute the denominator ideal of a solution matrix.

Theorem/ Definition 2.4.1. *[Coh12, Chapter 1] Let R be a Dedekind domain, and $\mathcal{K}$ its quotient field (fractions). Let M denote a finitely generated R-module. We define $V = \mathcal{K}M = \mathcal{K}\otimes_R M$ as a $\mathcal{K}$-module. Since M is finitely generated, $\mathcal{K}M$ is a finite-dimensional $\mathcal{K}$ vector space, whose dimension is called the rank of the R-module M.*

1. *The torsion submodule of M is defined by*

$$M_{\text{tors}} = \{x \in M \mid \exists a \in R\ /\{0\}, ax = 0\}$$

2. *We say that M is torsion-free if $M_{\text{tors}} = \{0\}$ and M is a torsion module if $M = M_{\text{tors}}$.*

3. *The R-module M is torsion-free if and only if M is a projective module.*

4. *There exists a torsion-free submodule N of M such that*

$$M = M_{\text{tors}} \oplus N \quad \text{and} \quad N \simeq M/M_{\text{tors}}.$$

There exist fractional ideals $\mathfrak{a}_i$ and elements $\alpha_i \in V$ such that $N = \mathfrak{a}_1\alpha_1 \oplus +\mathfrak{a}_2\alpha_2 \oplus \cdots \oplus \mathfrak{a}_r\alpha_r$, and there exist unique nonzero integral ideals $\mathfrak{d}_i$ of R and (non-unique) elements $\omega_i \in M_{\text{tors}}$ such that $M_{\text{tors}} = (R/\mathfrak{d}_1)\omega_1 \oplus \cdots \oplus (R/\mathfrak{d}_m)\omega_m$ and $\mathfrak{d}_{i-1} \subset \mathfrak{d}_i$ for $2 \le i \le m$.

Similar to the case of integer matrices, the definition of an elementary divisor can be generalize to arbitrary unitary commutative rings R as follows.

Definition 2.4.2. *The R-module $M = R/\mathfrak{d}_1 \oplus \cdots \oplus R/\mathfrak{d}_n$, where $\mathfrak{d}_1, \ldots, \mathfrak{d}_n$ are ideals of R subject to $0 \subseteq \mathfrak{d}_1 \subseteq \mathfrak{d}_2 \subseteq \cdots \subseteq \mathfrak{d}_n \subset R$, is said to be elementary divisor form presentation, and the ideals $\mathfrak{d}_1, \ldots, \mathfrak{d}_n$ are said to be the invariant factors or the elementary divisor ideals of M*

Now using the elementary divisor form presentation of a torsion-module, we can address the following definitions and theorem which will be used in the preceding section for modules over O.

Theorem 2.4.3. *[Coh12, Theorem 1.2.30] Every finitely generated torsion module over a Dedekind ring R has an elementary divisor form presentation.*

Definition 2.4.4. *1. The order-ideal of a finitely generated torsion module M over a Dedekind ring R is defined as $\mathfrak{E}_0(M/R)$, which is the product of elementary divisor ideals.*

2. Let M and N be two finitely generated torsion-free R-modules having the same rank and such that $N \subset M$. The order-ideal of the torsion module M/N is called the index-ideal of N into M and denoted $[M : N]$.

3. The exponent ideal $\mathfrak{A}(M/R)$ of a module M over a Dedekind ring R is defined as the ideal of R consisting of all the elements a of R which annihilate M as $aM = 0$.

Lemma 2.4.5. *[PZ97, Section 4.5, Lemma 5.31] For a finitely generated R-module M it holds that $\mathfrak{E}_0(M/R) \subseteq \mathfrak{A}(M/R)$. The equality holds when M is cyclic.*

O'Meara in [O'M00, Section 81C, 81D] gives the two lemmas below.

Theorem 2.4.6. *Let M and N be two torsion-free R-modules of rank n in the same vector space V as $M = \mathfrak{a}_1\alpha_1 \oplus \cdots \oplus \mathfrak{a}_n\alpha_n$ and $N = \mathfrak{b}_1\beta_1 \oplus \cdots \oplus \mathfrak{b}_n\beta_n$. Let $T = (t_{i,j})$ be the $n \times n$ matrix giving β_j in terms of α_i such that $(\beta_1, \ldots, \beta_n) = T(\alpha_1, \ldots, \alpha_n)$. Then $N \subseteq M$ if and only if $t_{ij}\mathfrak{b}_j \subseteq \mathfrak{a}_i$ for all i, j. Suppose $N \subseteq M$, then $M = N$ if and only if $\mathfrak{a}_1 \cdots \mathfrak{a}_n = \mathfrak{b}_1 \cdots \mathfrak{b}_n \cdot \det(T)$.*

Theorem 2.4.7. *Let $N \subseteq M$ be two torsion-free R-modules of rank n. Then there exist a basis $(m_1, \ldots, m_n)$ of V, fractional ideals $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ and integral ideals $\mathfrak{d}_1, \ldots, \mathfrak{d}_n$ such that $M = \mathfrak{a}_1 m_1 \oplus \cdots \oplus \mathfrak{a}_n m_n$, $N = \mathfrak{d}_1\mathfrak{a}_1 m_1 \oplus \cdots \oplus \mathfrak{d}_n\mathfrak{a}_n m_n$ and $\mathfrak{d}_{i-1} \subset \mathfrak{d}_i$ for $2 \leq i \leq n$. The ideals $\mathfrak{d}_i$ are uniquely determined by M and N.*

It can be easily proven that the ideals $\mathfrak{d}_i$ for $1 \leq i \leq n$ are the elementary divisors of M/N, and $\mathfrak{d}_n$ is the exponent ideal of M/N over R. Using the above two theorems, we summarize this section into a simple lemma which will be used in triangular denominator computation.

Lemma 2.4.8. *Let $N \subseteq M$ be finitely generated, torsion-free R-modules having the same rank n such that $M = \mathfrak{a}_1\alpha_1 \oplus \cdots \oplus \mathfrak{a}_n\alpha_n$, $N = \mathfrak{b}_1\beta_1 \oplus \cdots \oplus \mathfrak{b}_n\beta_n$. Suppose $T \in R^{n\times n}$ be the matrix such that $(\beta_1, \ldots, \beta_n) = T(\alpha_1, \ldots, \alpha_n)$. Then, the order-ideal of M/N (the index-ideal of N into M) is $\mathfrak{b}\mathfrak{a}^{-1}\det(T)$ where $\mathfrak{a} = \mathfrak{a}_1 \cdots \mathfrak{a}_n$ and $\mathfrak{b} = \mathfrak{b}_1 \cdots \mathfrak{b}_n$.*

Proof. Since N is a submodule of M, by Theorem 2.4.7 we can represent M and N with respect to a single basis of V as $M = \mathfrak{g}_1\gamma_1 \oplus \cdots \oplus \mathfrak{g}_n\gamma_n$ and $N = \mathfrak{d}_1\mathfrak{g}_1\gamma_1 \oplus \cdots \oplus \mathfrak{d}_n\mathfrak{g}_n\gamma_n$. Let $X = (x_{i,j})$ and $Y = (y_{i,j})$ be the transformations which change basis as $(\gamma_1, \ldots, \gamma_n) = X(\alpha_1, \ldots, \alpha_n)$ and $(\gamma_1, \ldots, \gamma_n) = Y(\beta_1, \ldots, \beta_n)$. Let $A = (\alpha_1, \ldots, \alpha_n)$, $B = (\beta_1, \ldots, \beta_n)$ and $G = (\gamma_1, \ldots, \gamma_n)$. Then we have that $X = GA^{-1}$ and $Y = GB^{-1}$. From the Theorem 2.4.6 we have that $\mathfrak{a} = \mathfrak{g}\det(X)$ and $\mathfrak{b} = \mathfrak{d}\mathfrak{g}\det(Y)$, where $\mathfrak{g} = \mathfrak{g}_1 \cdots \mathfrak{g}_n$ and $\mathfrak{d} = \mathfrak{d}_1 \cdots \mathfrak{d}_n$. Since $\mathfrak{g}_i/\mathfrak{d}_i\mathfrak{g}_i \simeq R/\mathfrak{d}_i$, we have the elementary divisor representation by Theorem 2.4.1 and 2.4.3 as $M/N \simeq R/\mathfrak{d}_1 \oplus \cdots \oplus R/\mathfrak{d}_n$. Hence $[M : N] = \mathfrak{d} = \mathfrak{b}\det(Y)^{-1}\mathfrak{a}^{-1}\det(X) = \mathfrak{b}\mathfrak{a}^{-1}\det(XY^{-1}) = \mathfrak{b}\mathfrak{a}^{-1}\det(BA^{-1}) = \mathfrak{b}\mathfrak{a}^{-1}\det(T)$. □

2.4.2 The Theory of Pseudo-matrices

In our applications, we are interested in the particular case where $R = O$ is the maximal order of the number field K. In order to work with matrix-normal forms over the Dedekind ring O, Cohen [Coh96] has introduced the notion of pseudo-objects as follows.

Definition 2.4.9. *Let $M \subseteq K^m$ be a finitely generated torsion-free $\mathcal{O}$-module, and set $V = KM = K \otimes_{\mathcal{O}} M$. We say that $(\mathfrak{a}_i, A_i)_{1\le i\le m}$ form a pseudo-generating set of M if $\sum_{i=1}^{m} \mathfrak{a}_i A_i = M$ where $(\mathfrak{a}_i)_{1\le i\le m}$ are fractional ideals of K and $A_i \in V$. (V is the vector space spanned by the rows of matrix $A \in K^{m\times n}$). We say that it is a pseudo-basis of M if the sum is direct.*

Definition 2.4.10. *Let $A \in K^{m\times n}$. A pseudo-matrix is a pair $\mathcal{A} = ((\mathfrak{a}_i), A)$ where the family $(\mathfrak{a}_i)_{1\le i\le m}$ of fractional ideals of K are called the coefficient ideals. If $\mathcal{A}$ has full rank and $n = m$, we define the determinant ideal of $\mathcal{A}$ as* $\det(\mathcal{A}) = \det(A) \cdot \mathfrak{a}_1 \cdots \mathfrak{a}_m$.

Note: We define the *determinant ideal* of $A \in \mathcal{O}^{n\times n}$ as $\det(\mathcal{A})$, where $\mathcal{A}$ is the corresponding pseudo-matrix with all the coefficients ideals being the unit ideal ($\mathfrak{a}_i = \mathcal{O}$ for all i).

Definition 2.4.11. *The pseudo-element $\mathfrak{a}v$ where $v \in V$ and $\mathfrak{a} \triangleleft K$ is said to be integral if $\mathfrak{a}v \subset \mathcal{O}^n$.*

Definition 2.4.12. *A matrix $A \in \mathcal{O}^{n\times n}$ is called unimodular if* $\det(A)$ *is a unit. A pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i), A)$ is called unimodular if* $\det(\mathcal{A})$ *is a unit.*

In the integer case, Lemma 2.2.4 uses Hermite form to find a minimal triangular denominator. We will use a similar method with the notion of pseudo-Hermite normal form over Dedekind domains.

Theorem/ Definition 2.4.13. *[Coh96, Theorem 2.5] Let $\mathcal{A} = ((\mathfrak{a}_i), A)$ be a pseudo-matrix with $A = (a_{i,j}) \in \mathcal{O}^{m\times n}$ such that A has rank r. We say that $\mathcal{A}$ is in pseudo-Hermite normal form if and only if the following hold:*

1. *The first r rows of A are nonzero. For $1 \le i \le r$ let a_{i,j_i} be the last nonzero entry in row i. Then $1 \le j_1 < j_2 < \cdots < j_r \le m$.*
2. *We have $a_{i,j_i} = 1$ for $1 \le i < k \le r$.*

Let $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le m}, A)$ be a pseudo matrix over $\mathcal{O}$ with n columns. Then the $\mathcal{O}$-module generated by the pseudo-matrix $\mathcal{A}$ is denoted by $\mathrm{Mod}(\mathcal{A}) = \sum_{i=1}^{m} \mathfrak{a}_i A_i$ where A_i are the rows of A.

Definition 2.4.14. *Two pseudo-matrices $\mathcal{A}$ and $\mathcal{B}$ are called equivalent if and only if* $\mathrm{Mod}(\mathcal{A}) = \mathrm{Mod}(\mathcal{B})$.

Theorem 2.4.15. *[Hop98, Lemma 3.2.3] Let $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le m}, A)$ and $\mathcal{B} = ((\mathfrak{b}_j)_{1\le j\le m}, B)$ be two pseudo-matrices with n columns. Then $\mathcal{A}$ and $\mathcal{B}$ are equivalent if and only if there exists a matrix $T = (t_{ij})_{i,j} \in K^{m\times m}$ such that $B = TA$ and $t_{ij}\mathfrak{b}_j \subset \mathfrak{a}_i$, and there exists a matrix $U = (u_{ij})_{i,j} \in K^{m\times m}$ such that $A = UB$ and $u_{ij}\mathfrak{a}_j \subset \mathfrak{b}_i$. Thus, we referred that $\mathcal{A}$ and $\mathcal{B}$ are equivalent via T, and $\mathcal{B}$ and $\mathcal{A}$ are equivalent via U.*

Now we can extend the Lemma 2.4.8 for pseudo-matrices as follows.

Theorem 2.4.16. *Let $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$ and $\mathcal{B} = ((\mathfrak{b}_j)_{1\le i\le n}, B)$ be two pseudo-bases of R-modules* $\mathrm{Mod}(\mathcal{A})$ *and* $\mathrm{Mod}(\mathcal{B})$ *in a same vector space V such that* $\mathrm{Mod}(\mathcal{B}) \subset \mathrm{Mod}(\mathcal{A})$. *Then, the order-ideal of the quotient module* $\mathrm{Mod}(\mathcal{A})/\mathrm{Mod}(\mathcal{B})$ *is given by* $[\mathrm{Mod}(\mathcal{A}) : \mathrm{Mod}(\mathcal{B})] = \det(\mathcal{B})/\det(\mathcal{A})$.

Proof. Let $T = (t_{i,j})$ be the $n \times n$ basis transformation matrix such that $B = TA$. Set $\mathfrak{a} = \mathfrak{a}_1 \cdots \mathfrak{a}_n$ and $\mathfrak{b} = \mathfrak{b}_1 \cdots \mathfrak{b}_n$. Then $[\mathrm{Mod}(\mathcal{A}) : \mathrm{Mod}(\mathcal{B})] = \mathfrak{b}\mathfrak{a}^{-1} \det(T)$ by Lemma 2.4.8. □

Given a pseudo-matrix, Hoppe in [Hop98, Section 4.10] defines the dual pseudo-matrix.

Definition 2.4.17. *Let $\mathcal{A} = ((\mathfrak{a}_i), A)$ be a square pseudo-matrix with n rows and rank n. If $M = \mathrm{Mod}(\mathcal{A})$ then the dual pseudo-matrix is the pseudo-matrix corresponding to the $\mathcal{O}$-module $M_D = \{x \in V \mid x^t M \subseteq \mathcal{O}^n\}$. The dual can be obtained as: $\mathcal{A}_D = ((\mathfrak{a}_i^{-1}), A^{-t})$.*

Now, we can extend equivalence relations for dual pseudo-matrices as follows:

Lemma 2.4.18. *Let $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$ and $\mathcal{B} = ((\mathfrak{b}_j)_{1\le j\le n}, B)$ be two pseudo-matrices of rank n. Suppose they are equivalent with $T : \mathcal{A} \to \mathcal{B}$, $T = (t_{ij})_{i,j} \in K^{n\times n}$ and $U : \mathcal{B} \to \mathcal{A}$, $U = (u_{ij})_{i,j} \in K^{n\times n}$. Then, their duals are equivalent with $T^{-t} : \mathcal{A}_D \to \mathcal{B}_D$, $T^{-t} = (t'_{ij})_{i,j} \in K^{n\times n}$ and $U^{-t} : \mathcal{B}_D \to \mathcal{A}_D$, $U^{-t} \in K^{n\times n}$.*

Proof. Consider the isomorphism $T : \mathcal{A} \to \mathcal{B}$. From Definition 2.4.15, it follows that $TA = B$ and $t_{ij}\mathfrak{b}_j \subset \mathfrak{a}_i$. Similarly, from the isomorphism $U : \mathcal{B} \to \mathcal{A}$, we get $UB = A$ and $u_{ij}\mathfrak{a}_j \subset \mathfrak{b}_i$. Since $T, U \in \mathrm{GL}_n(K)$, it holds: $TA = B \iff A^tT^t = B^t \iff T^{-t}A^{-t} = B^{-t}$ and $A = UB \iff U^{-1}A = B \iff U^tA^{-t} = B^{-t}$. This implies: $U^t = T^{-t}$ and $t'_{ij} = u_{ji}$. From the properties of U it holds: $u_{ij}\mathfrak{a}_j \subset \mathfrak{b}_i \iff u_{ji}\mathfrak{a}_i \subset \mathfrak{b}_j \iff u_{ji}\mathfrak{b}_j^{-1} \subset \mathfrak{a}_i^{-1} \iff t'_{ij}\mathfrak{b}_j^{-1} \subset \mathfrak{a}_i^{-1}$. Hence the isomorphism $T^{-t} : \mathcal{A}_D \to \mathcal{B}_D$ holds. Similarly, the isomorphism $U^{-t} : \mathcal{B}_D \to \mathcal{A}_D$ can be proven. □

Definition 2.4.19. *Let $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$ be a pseudo-basis of an O-module and $\mathfrak{d}$ be an ideal in K. Then, we define $\mathfrak{d}\mathcal{A} := ((\mathfrak{d}\mathfrak{a}_i)_{1\le i\le n}, A)$. For some non-zero $d \in O$ we define $d\mathcal{A} := ((\mathfrak{a}_i)_{1\le i\le n}, dA)$.*

Definition 2.4.20. *The denominator of a pseudo-matrix $\mathcal{A}$ is the minimal natural number d such that $d\mathcal{A}$ is an integral pseudo-matrix.*

The denominator ideal of a pseudo-matrix $\mathcal{A}$ is the maximal integral ideal $\mathfrak{d}$ such that $\mathfrak{d}\mathcal{A}$ is an integral pseudo-matrix.

2.4.3 Extension to the Determinant Algorithm

In this section, we present an algorithm to compute the determinant ideal of a matrix $A \in O^{n\times n}$, generalizing the method for determinant computation over integers by Storjohann. We use pseudo-matrices and pseudo-normal forms as tools to achieve this. The basic idea of our approach is as follows:

$\mathfrak{d} = O$ and $\mathfrak{a}_i = O$ for $i = 1, \ldots, n$.
$\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$
while $\mathcal{A}$ is not unimodular **do**
  b = random matrix in $K^{n\times 1}$
  Compute $\mathcal{A}'$ such that $\mathrm{Mod}(\mathcal{A}') \simeq \mathrm{Mod}\,((O, (\mathfrak{a}_i)_{1\le i\le n}), [b \mid A])$.
  $\mathfrak{d} = \mathfrak{d}\,[\mathrm{Mod}(\mathcal{A}') : \mathrm{Mod}(\mathcal{A})]$
  $\mathcal{A} = \mathcal{A}'$
end while

In the next few sections, we elaborate this idea in details. The method is theoretically attractive and efficient in practice.

Algorithm 7 Maximal Denominator Ideal (`MaximalDenominator`)

Input: A pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$, where $A \in O^{n\times n}$ and $b \in O^{n\times 1}$.
Output: $\mathcal{M}$ such that $\mathrm{Mod}(\mathcal{M}) \simeq Ob + \sum_1^n \mathfrak{a}_i A_i$ and the maximal denominator ideal of the pseudo-matrix $((\mathfrak{a}_i^{-1})_{1\le i\le n}, s)$, where s is the solution of the linear system $Ax = b$.

1: $s = \mathtt{Solve}(A, b)$
2: $S = \begin{bmatrix} 1 & -s \end{bmatrix}^t$ and $\mathfrak{s} = (O, \mathfrak{a}_1^{-1}, \ldots, \mathfrak{a}_n^{-1})$.
3: $\mathcal{S} = ((\mathfrak{s}_i)_{1\le i\le n+1}, S)$
4: $\mathcal{H}, T = \mathtt{pseudo\_HNF\_with\_transform}(\mathcal{S})$
5: M = the lower block matrix of $(T^{-t}[b \mid A]^t)$ with rows from 2 to $n+1$.
6: $(\mathfrak{b}_i)_{1\le i\le n+1}$ = Coefficient ideals of $\mathcal{H}$.
7: **return** $\mathcal{M} = ((\mathfrak{b}_i^{-1})_{2\le i\le n+1}, M)$, $\prod_{i=1}^n \mathfrak{a}_i \prod_{i=2}^{n+1} \mathfrak{b}_i^{-1} \det(AM^{-1})$

Given a linear system $Ax = b$, Algorithm 7 can compute the denominator ideal of the solution. This is, with ideas from Posur [Pos] in private conversation. The construction works exactly as the minimal triangular denominator computation in the integer case. Here we use `pseudo_HNF_with_transform` algorithm which is implemented in Hecke [FHHJ17] based on [BFH17, Algorithm 5].

We prove that the Algorithm 7 is correct in two steps.

Theorem 2.4.21. *Given a pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i)_{1\leq i\leq n}, A)$, where $A \in \mathcal{O}^{n\times n}$ and $b \in \mathcal{O}^{n\times 1}$, Algorithm 7 constructs an equivalent pseudo-matrix $\mathcal{M}$ for $((\mathfrak{a}'_i)_{1\leq i\leq n+1}, A')$ where $A' = \begin{bmatrix} b & A \end{bmatrix}^t$ and $\mathfrak{a}' = (\mathcal{O}, \mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_n)$.*

Proof. Let $As = b$, with the usual notation $s = \begin{bmatrix} s_1 & \ldots & s_n \end{bmatrix}^t$. Consider the $\mathcal{O}$-module corresponding to the pseudo-matrix $\mathcal{S}$: $\mathrm{Mod}(\mathcal{S}) = \mathcal{O}(1) + \sum_{i=1}^{n} \mathfrak{a}_i^{-1}(-s_i) = \sum_{i=1}^{n+1} \mathfrak{s}_i S_i$, where $S = \begin{bmatrix} 1 & -s \end{bmatrix}^t$ and $\mathfrak{s} = (\mathcal{O}, \mathfrak{a}_1^{-1}, \ldots, \mathfrak{a}_n^{-1})$. We compute $\mathcal{H}$, the pseudo-HNF of $\mathcal{S}$ with the transformation matrix $T = (t_{ij})_{i,j} \in \mathrm{GL}_{n+1}(K)$. As a matrix multiplication equivalence:

$T : \mathcal{S} \to \mathcal{H}$, where $\mathrm{Mod}(\mathcal{H}) = \mathfrak{h}_1 \cdot 1 + \sum_{2}^{n+1} \mathfrak{h}_i \cdot 0$.

From the Definition 2.4.15, as abstract modules it holds:

$$\mathcal{O} \oplus \bigoplus_{i=1}^{n} \mathfrak{a}_i^{-1} = \bigoplus_{i=1}^{n+1} \mathfrak{s}_i \xrightarrow{T} \bigoplus_{i=1}^{n+1} \mathfrak{h}_i \iff t_{ij} \in \mathfrak{h}_j^{-1}\mathfrak{s}_i$$

From Lemma 2.4.18, on the dual side we have a map $(T^{-1})^t = (t'_{ij})_{i,j} \in \mathrm{GL}_{n+1}(K)$ such that:

$$t'_{ij} \in \mathfrak{h}_j\mathfrak{s}_i^{-1} \implies \bigoplus_{i=1}^{n+1} \mathfrak{s}_i^{-1} = \mathcal{O} \oplus \bigoplus_{i=1}^{n} \mathfrak{a}_i \xrightarrow{(T^{-1})^t} \bigoplus_{i=1}^{n+1} \mathfrak{h}_i^{-1}$$

Let $N := (T^{-1})^t \begin{bmatrix} b & A \end{bmatrix}^t$, and denote $N = \begin{bmatrix} N_1 & \ldots & N_{n+1} \end{bmatrix}^t \in \mathcal{O}^{n+1,n}$ with rows N_i. We obtain a module equivalence as:

$$\mathcal{O}b + \sum_{i=1}^{n} \mathfrak{a}_i A_i \xrightarrow{(T^{-1})^t} \sum_{i=1}^{n+1} \mathfrak{h}_i^{-1} N_i \tag{2.4.1}$$

We claim that $N_1 = O_{1\times n}$, since

$$\begin{bmatrix} b & A \end{bmatrix} \begin{bmatrix} 1 & -s \end{bmatrix}^t = O_{n\times 1} \implies \begin{bmatrix} b & A \end{bmatrix} T^{-1} T \begin{bmatrix} 1 & -s \end{bmatrix}^t = O_{n\times 1}$$

$$T \begin{bmatrix} 1 & -s \end{bmatrix}^t = \begin{bmatrix} 1 & O_{n\times 1} \end{bmatrix}^t \implies \left(\begin{bmatrix} b & A \end{bmatrix} T^{-1}\right)^t = (T^{-1})^t \begin{bmatrix} b & A \end{bmatrix}^t = \begin{bmatrix} O_{1\times n} & M \end{bmatrix}^t,$$

where $M = \begin{bmatrix} N_2 & \ldots & N_{n+1} \end{bmatrix}^t$.

Since $T \in \mathrm{GL}_{n+1}(K)$, (2.4.1) gives a transformation equivalence of two modules as $\mathcal{O}b + \sum_{i=1}^{n} \mathfrak{a}_i A_i \simeq \sum_{i=1}^{n+1} \mathfrak{h}_i^{-1} N_i$. Elementary transformations allows us to delete the first row of $\sum_{i=1}^{n+1} \mathfrak{h}_i^{-1} N_i$ with its coefficient ideal, as it is a zero row. Hence, there is a module equivalence: $\mathcal{O}b + \sum_{i=1}^{n} \mathfrak{a}_i A_i \simeq \sum_{i=2}^{n+1} \mathfrak{h}_i^{-1} N_i = \mathrm{Mod}(\mathcal{M})$. In other words, $\mathrm{Mod}(\mathcal{M})$ is the module generated by the pseudo matrix $\mathcal{A}' = ((\mathfrak{a}'_i)_{1\leq i\leq n+1}, A')$ where $A' = \begin{bmatrix} b & A \end{bmatrix}^t$ and $\mathfrak{a}' = (\mathcal{O}, \mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_n)$.

□

Theorem 2.4.22. *Given a pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i), A)$, where $A \in \mathcal{O}^{n\times n}$ and $b \in \mathcal{O}^{n\times 1}$, Algorithm 7 computes the maximal denominator ideal of the pseudo-matrix $((\mathfrak{a}_i^{-1})_{1\leq i\leq n}, s)$, where s is the solution of the linear system $Ax = b$.*

Proof. Consider the two modules corresponding to the two pseudo-matrices $\mathcal{A}_1 = ((\mathfrak{a}_i)_{1\le i\le n}, A)$ and $\mathcal{A}' = ((\mathfrak{a}'_i)_{1\le i\le n+1}, A')$ as: $\mathcal{M}_1 = \mathrm{Mod}(\mathcal{A}_1) = \sum_{i=1}^{n} \mathfrak{a}_i A_i$ and $\mathcal{M}' = \mathrm{Mod}(\mathcal{A}') = \sum_{i=1}^{n+1} \mathfrak{a}'_i A'_i$. Using Algorithm 7 with Theorem 2.4.21, we can compute a module $\mathcal{M}_2 = \mathrm{Mod}(\mathcal{A}_2)$ with a pseudo-basis $\mathcal{A}_2 = ((\mathfrak{b}_i)_{1\le i\le n}, A_2)$ for some $A_2 \in K^{n\times n}$ such that $\mathrm{Mod}(\mathcal{A}_2) \simeq \mathrm{Mod}(\mathcal{A}')$. Since $\mathrm{Mod}(\mathcal{A}_1)$ and $\mathrm{Mod}(\mathcal{A}_2)$ are modules of full rank n and $\mathrm{Mod}(\mathcal{A}_1) \subset \mathrm{Mod}(\mathcal{A}_2)$, the quotient module $\mathcal{M}_2/\mathcal{M}_1 = \mathrm{Mod}(\mathcal{A}_2)/\mathrm{Mod}(\mathcal{A}_1)$ is a torsion $\mathcal{O}$-module. By Theorem 2.4.3 we have an elementary divisor form presentation as $\mathcal{M}_2/\mathcal{M}_1 = \sum_{i=1}^{n} \mathcal{O}/\mathfrak{d}_i$ for some elementary divisor ideals $\mathfrak{d}_i$ for $i = 1, \ldots, n$.

We claim that the exponent ideal of the quotient module $\mathcal{M}_2/\mathcal{M}_1$ is the same as the index-ideal $[\mathcal{M}_2 : \mathcal{M}_1] = \prod_{i=1}^{n} \mathfrak{d}_i = \mathfrak{d}$, and it is equal to the denominator ideal of $\mathcal{S} = ((\mathfrak{a}_i^{-1})_{1\le i\le n}, s)$.

Since elementary divisor ideals are uniquely determined by the isomorphism class of the module, we have that $(\mathcal{M}')/(\mathcal{M}_1) \simeq (\mathcal{M}_2)/(\mathcal{M}_1) = \prod_{i=1}^{n} \mathfrak{d}_i$. Since there is one additional basis element in $\mathcal{M}'$ that is not in $\mathcal{M}_1$, the quotient module $\mathcal{M}'/\mathcal{M}_1$ is cyclic. Therefore, by Lemma 2.4.5, the index-ideal is the same as the exponent ideal. Hence, we get the index-ideal $[\mathcal{M}_2 : \mathcal{M}_1] = \mathfrak{A}((\mathcal{M}_2/\mathcal{M}_1)/\mathcal{O}) = \mathfrak{d}$, where $\mathfrak{d}$ is the product of elementary divisors. We can precisely compute $\mathfrak{d}$ using the Theorem 2.4.16, which says that the index-ideal$[\mathcal{M}_2 : \mathcal{M}_1]$ is the same as $\det(\mathcal{A}_1)/\det(\mathcal{A}_2)$. Hence we have $\mathfrak{d} = \prod_{i=1}^{n} \mathfrak{a}_i \prod_{i=1}^{n} \mathfrak{b}_i^{-1} \det(A_1 A_2^{-1})$.

Let z be the common denominator of s. Since z is the smallest multiple of b such that s becomes integral, it holds $\mathcal{O}(zb) \in \sum_{i=1}^{n} \mathcal{O}A_i$. Therefore it holds that $\mathfrak{a}(\mathcal{O}zb) \in \sum_{i=1}^{n} \mathfrak{a}_i A_i = \mathcal{M}_1$ where $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{a}_i$. Since $z\mathfrak{a}$ is the largest possible ideal such that $z\mathfrak{a}(\mathcal{O}b) \in \mathcal{M}_1$ holds, it is the denominator ideal of $\mathrm{Mod}(\mathcal{S}) = \sum_{i=1}^{n} \mathfrak{a}_i s_i$. Since the exponent ideal $\mathfrak{d}$ is the largest possible ideal such that $\mathfrak{d}\mathcal{M}' \subset \mathcal{M}_1$ holds, we have $\mathfrak{d} = z\mathfrak{a}$.

□

Algorithm 8 Determinant Ideal (`PseudoDeterminant`)

Input: $A \in \mathcal{O}^{n\times n}$.
Output: Determinant ideal of matrix A.

1: $\mathcal{O}$ = maximal order of K
2: $n = \mathrm{rows}(A)$
3: $\mathfrak{b}_i = \mathcal{O}$ for $i = 2, \ldots, n+1$
4: $M = A$, $D = I_{n\times n}$
5: **while** true **do**
6: $b \in \mathcal{O}^{n\times 1}$, coefficients are chosen uniformly randomly from the interval $[-100, 100] \subset \mathbb{Z}$.
7: $s =$ `DixonSolver`(M, b)
8: $S = \begin{bmatrix} 1 & O_{1\times n} \\ -s & I_{n\times n} \end{bmatrix}$, $\mathfrak{s} = (\mathcal{O}, \mathfrak{b}_2, \ldots, \mathfrak{b}_{n+1})$
9: $\mathcal{S} = ((\mathfrak{s}_i)_{1\le i\le n+1}, S)$.
10: $\mathcal{H} = ((\mathfrak{b}_i)_{1\le i\le n+1}, H) =$ `Pseudo_HNF`$(\mathcal{S})$, with inverse transformation $U = SH^{-1}$
11: $D = D(H_{[2:n+1,2:n+1]})$
12: $\mathcal{M} = ((\mathfrak{b}_i^{-1})_{2\le i\le n+1}, M)$ for $M = (U^t \left[\begin{array}{c|c} b & M \end{array}\right]^t)_{[2:n+1,:]}$
13: **if** $\det(M)$ is a unit **then**
14: $\mathcal{D} = ((\mathfrak{b}_i)_{2\le i\le n+1}, D)$
15: **return** $\mathfrak{D} = \det(\mathcal{D})$
16: **end if**
17: **end while**

The Algorithm 8 `PseudoDeterminant` extends Algorithm 7 for determinant computation. Given a matrix A, we apply a step of Algorithm 7 with the inputs: $\mathcal{A} = ((\mathcal{O})_i, A)$ and an arbitrary row matrix

b. Then, we iterate the Algorithm 7, with the new inputs $\mathcal{A} = \mathcal{M}$ (this is $\mathcal{M} \leftarrow \langle \mathcal{M}, b\rangle$, the output pseudo-matrix from the previous step) and another arbitrary matrices b, until $\mathcal{M}$ becomes unimodular ($\det(\mathcal{M})$ is a unit). In Section 3 we discuss a sophisticated algorithm (Algorithm 14 `UniCert`) to test the unimodularity of a pseudo matrix efficiently. The pseudo-matrix $\mathcal{M}$ eventually becomes unimodular, since the $\mathcal{O}$-module generated by rows of A is a sub-module of $\mathrm{Mod}(\mathcal{M})$ and, in each iteration, we make random projections to reduce the determinant of $\mathcal{M}$ by a (large) factor of $\det(A)$, which is computed using denominators of the solutions of linear systems $Mx = b$. Then, we can obtain the determinant ideal $\mathfrak{D}$ from the product of denominator ideals in each step. Finally, we use the CRT approach to recover the determinant of the matrix A from the determinant ideal $\mathfrak{D}$ (In another word, we continue with Algorithm 6, passing the computed $\mathfrak{D}$ to Step 4).

We can illustrate the main idea of Algorithm 8 by the first two steps as follows:

Take $A_1 = A$, in each iteration i, for random $b_i \in K^{1\times n}$, let $s_i \in K^{1\times n}$ be the solution matrix of the linear system $A_i x = b_i$. In Algorithm 7, we have considered the HNF form of one column matrix $[1 \mid -s_i]^t$. Therefore, the transformation matrix T is not unique, and matrix entries of T can be arbitrarily large due to the kernel space. Thus, Algorithm 8 computes the pseudo-HNF of the full rank pseudo-matrix: $\mathcal{S}_i = \begin{matrix} \mathcal{O} \\ \mathcal{O} \end{matrix}\left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s_i & I_{n\times n} \end{array}\right]$ where the coefficient ideal $\mathcal{O}$ in the second row is an n-array of $\mathcal{O}$ ideals. The coefficient ideals of the pseudo-HNF of $\mathcal{S}_i$ are indicated by $(\mathfrak{v}_{(i)}, \mathfrak{h}_{(i)})$, where $\mathfrak{v}_{(i)}$ is one fractional ideal and $\mathfrak{h}_{(i)}$ is an n-array of fractional ideals. Similarly, we denote the n-array containing inverses of each ideal in $\mathfrak{h}_{(i)}$ by ${\mathfrak{h}_{(i)}}^{-1}$.

$$\mathcal{S}_1 = \begin{matrix} \mathcal{O} \\ \mathcal{O} \end{matrix}\left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s_1 & I_{n\times n} \end{array}\right] \begin{matrix} T_1 \\ \longmapsto \end{matrix} \begin{matrix} \mathfrak{v}_{(1)} \\ \mathfrak{h}_{(1)} \end{matrix} \left[\begin{array}{c|c} * & * \\ \hline O_{n\times 1} & H_1 \end{array}\right] \tag{2.4.2}$$

Let T_1 be the transformation matrix, and we can compute it as follows:

$$T_1 = \left[\begin{array}{c|c} * & * \\ \hline O_{n\times 1} & H_1 \end{array}\right]\left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline s_1 & I_{n\times n} \end{array}\right] = \left[\begin{array}{c|c} * & * \\ \hline H_1 s_1 & H_1 \end{array}\right]$$

Now we consider the relation of dual pseudo-matrices and apply the Algorithm 7 as follows. We obtain the pseudo-matrix $\mathcal{A}_2 = (\mathfrak{h}_{(1)}^{-1}, A_2)$ such that $\mathrm{Mod}(\mathcal{A}_2) \simeq \mathrm{Mod}(\mathcal{A}')$ where $\mathcal{A}' = ((\mathcal{O})_{1\le i\le n+1}, [b_1 \mid A_1]^t)$.

$$\begin{matrix} \mathcal{O}^{-1} \\ \mathcal{O}^{-1} \end{matrix}\left[\begin{array}{c} b_1 \\ \hline A_1 \end{array}\right] \begin{matrix} {T_1}^{-t} \\ \longmapsto \end{matrix} \begin{matrix} {\mathfrak{v}_{(1)}}^{-1} \\ {\mathfrak{h}_{(1)}}^{-1} \end{matrix} \left[\begin{array}{c} O_{1\times n} \\ \hline A_2 \end{array}\right]$$

If we apply the transformation T_1 on matrices as follows, we can see the relation between two $n \times n$ matrices A_1 and A_2.

$$\left[\begin{array}{c} b_1 \\ \hline A_1 \end{array}\right] = T_1^t \left[\begin{array}{c} O_{1\times n} \\ \hline A_2 \end{array}\right] = \left[\begin{array}{c|c} * & H_1 s_1 \\ \hline * & H_1 \end{array}\right]\left[\begin{array}{c} O_{1\times n} \\ \hline A_2 \end{array}\right] = \left[\begin{array}{c} * \\ \hline H_1 A_2 \end{array}\right]$$

This gives that $A_1 = H_1 A_2$. Let $T_1 = (t_{i,j})$, from 2.4.2 we have $t_{i,j} \in \mathcal{O} {\mathfrak{h}_{(1)}}_j^{-1}$ for $2 \le i, j \le n+1$. Since $H = T_{1[2:n+1,2:n+1]}$ we have the relation: $(\mathfrak{h}_{(1)}^{-1}, A_2) \xrightarrow{H_1} (\mathcal{O}, A_1)$.

By Theorem 2.4.22, the denominator ideal of $\mathcal{S}_1 = (\mathcal{O}, s_1)$ is $\prod_{i=1}^{n} {\mathfrak{h}_{(1)}}_i \mathcal{O}^{-1} \det(H_1)$.

If $(\mathfrak{h}_{(1)}^{-1}, A_2)$ is not unimodular, we continue Step 7 of Algorithm 8 with another arbitrary matrix b_2, and produce the linear system $A_2 x = b_2$ with the solution matrix s_2 (We repeat Algorithm 7 for the pseudo-matrix $(\mathfrak{h}_{(1)}^{-1}, A_2)$ and b_2). We use the coefficient ideals $\mathfrak{h}_{(1)}$ coming from the previous step in $\mathcal{S}_2$ as follows:

$$\mathcal{S}_2 = \begin{matrix} \mathcal{O} \\ \mathfrak{h}_{(1)} \end{matrix}\left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s_2 & I_{n\times n} \end{array}\right].$$

$$\begin{matrix} \mathcal{O} \\ \mathfrak{h}_{(1)} \end{matrix}\left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s_2 & I_{n\times n} \end{array}\right] \begin{matrix} T_2 \\ \longmapsto \end{matrix} \begin{matrix} \mathfrak{v}_{(2)} \\ \mathfrak{h}_{(2)} \end{matrix} \left[\begin{array}{c|c} * & * \\ \hline O_{n\times 1} & H_2 \end{array}\right]$$

$$\begin{matrix} \mathcal{O}^{-1} \\ {\mathfrak{h}_{(1)}}^{-1} \end{matrix}\left[\begin{array}{c} b_2 \\ \hline A_2 \end{array}\right] \begin{matrix} {T_2}^{-t} \\ \longmapsto \end{matrix} \begin{matrix} {\mathfrak{v}_{(2)}}^{-1} \\ {\mathfrak{h}_{(2)}}^{-1} \end{matrix} \left[\begin{array}{c} O_{1\times n} \\ \hline A_3 \end{array}\right]$$

This gives $A_2 = H_2A_3$ and the relation: $(\mathfrak{b}_{(2)}^{-1}, A_3) \xrightarrow{H_2} (\mathfrak{b}_{(1)}^{-1}, A_2)$. The denominator ideal of $\mathcal{S}_2 = (\mathfrak{b}_{(1)}, s_2)$ is $\prod_{i=1}^{n} \mathfrak{b}_{(2)_i} \prod_{i=1}^{n} \mathfrak{b}_{(1)_i}^{-1} \det(H_2)$.

Suppose that iteration stops at step k, when $(\mathfrak{b}_{(k)}^{-1}, A_{k+1})$ is unimodular. Combining all the modular equivalences, we get:

$$(\mathfrak{b}_{(k)}^{-1}, A_{k+1}) \xrightarrow{H_k} \cdots \xrightarrow{H_3} (\mathfrak{b}_{(2)}^{-1}, A_3) \xrightarrow{H_2} (\mathfrak{b}_{(1)}^{-1}, A_2) \xrightarrow{H_1} (\mathcal{O}, A_1)$$

Which can be reduced to the following relation:

$$(\mathfrak{b}_{(k)}^{-1}, A_{k+1}) \xrightarrow{H_1H_2\cdots H_k} (\mathcal{O}, A_1) \tag{2.4.3}$$

Let $\bar{H} = H_1H_2 \cdots H_k$ and $\bar{\mathfrak{b}}_{(k)} = \prod_{i=1}^{n} \mathfrak{b}_{(k)_i}$. We claim that the product of the denominator ideals $\bar{\mathfrak{b}}_{(k)} \det(\bar{H})$ is the determinant ideal of A.

From (2.4.3) we have $A_1 = \bar{H}A_{k+1}$. Then it holds:

$$\begin{aligned} \det(A_1) &= \det(\bar{H}A_{k+1}) \\ \det(A_1)\mathcal{O} &= \bar{\mathfrak{b}}_{(k)}\bar{\mathfrak{b}}_{(k)}^{-1} \det(\bar{H}A_{k+1}) \\ &= (\bar{\mathfrak{b}}_{(k)} \det(\bar{H}))(\bar{\mathfrak{b}}_{(k)}^{-1} \det(A_{k+1})). \end{aligned}$$

When the pseudo-matrix $(\mathfrak{b}_{(k)}^{-1}, A_{k+1})$ is unimodular for some k the determinant ideal is complete and it is $\mathfrak{D} = \bar{\mathfrak{b}}_{(k)} \det(\bar{H}) = \prod_{i=1}^{n} \mathfrak{b}_{(k)} \det(H_1H_2 \cdots H_k)$.

Theorem 2.4.23. *Algorithm 8 is correct.*

Proof. It is clear from the construction that the algorithm returns the determinant ideal of a matrix as explained in the above discussion. Compared to the discussion, in Algorithm 8 for each iteration i we have use the notation $\mathcal{M}$ for the output pseudo-matrix with the corresponding matrix $A_i = M$. The inverse transformation T_i^{-1} is denoted by U and the matrix $H_i = H_{[2:n+1,2:n+1]}$. □

Note that the sub-algorithm `UniCert` will be explained in Section 3. Practically this algorithm is quite slow, and we discuss the obstacles to optimizing this step in Section 3.3. In our implementation, we have tested the unimodularity of a matrix using the norm of $\det(A)$, which is computed as the determinant of the corresponding integer matrix of representation matrices ($nd \times nd$ integer matrix). Then, we compare $\text{norm}(\det(A))$ with the product of norms of denominator ideals and continue the iteration until we get the complete determinant ideal.

Similar to the integer case, in our application, we can generalize Definition 2.2.3 of the minimal triangular denominator as follows:

Definition 2.4.24. *A triangular denominator of a pseudo-matrix $\mathcal{S} = ((\mathfrak{s}_i), S)$ is defined by any pseudo-matrix $\mathcal{D} = ((\mathfrak{d}_i), D)$ such that $((\mathfrak{d}_i\mathfrak{s}_i), DS)$ is integral and, D is a non-singular upper triangular matrix. A minimal triangular denominator of $\mathcal{S}$ is a triangular denominator with minimal magnitude determinant (i.e.* $\min\{\text{norm}(\det(\mathcal{D}))|\ \mathcal{D}$ *is a triangular denominator of* $\mathcal{S}\}$*). In another word , if $\mathcal{D}$ is a minimal triangular denominator of $\mathcal{S}$, then* $\det(\mathcal{D})$ *is the maximal denominator ideal.*

Theorem 2.4.25. *Let $\mathcal{H} = ((\mathfrak{h}_i)_{1\le i\le n+1}, H)$ be the pseudo-HNF of $\mathcal{S}$ in Step 9 of the Algorithm 8. Here, $\mathcal{S}$ is corresponding to the solution matrix s of a linear system $Ax = b$. Then a minimal triangular denominator of the pseudo-matrix $((\mathfrak{s}_i)_{2\le i\le n+1}, s)$ is given by*

$$\left((\mathfrak{h}_i\mathfrak{s}_i^{-1})_{2\le i\le n+1},\ H_{[2:n+1,2:n+1]}\right)$$

Proof. Let $\mathfrak{s} = (\mathfrak{s}_i)_{2\le i\le n+1}$, $\mathfrak{h} = (\mathfrak{h}_i)_{2\le i\le n+1}$ and $H = H_{[2:n+1,2:n+1]}$. Then for some transformation matrix T we have the identity:

$$\mathcal{S} = \begin{matrix} O \\ \mathfrak{s} \end{matrix} \left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s & I_{n\times n} \end{array}\right] \begin{matrix} T \\ \longmapsto \end{matrix} \begin{matrix} \mathfrak{h}_1 \\ \mathfrak{h} \end{matrix} \left[\begin{array}{c|c} * & * \\ \hline O_{n\times 1} & H \end{array}\right]$$

We have the following identity by applying Algorithm 7, considering dual relations. The pseudo-matrix $\mathcal{A}_2 = (\mathfrak{h}^{-1}, A_2)$ satisfies the relation that $\mathrm{Mod}(\mathcal{A}_2) \simeq \mathrm{Mod}(\mathcal{A}')$ where $\mathcal{A}' = ((O, \mathfrak{s}^{-1}), [b \mid A]^t)$.

$$\begin{matrix} O \\ \mathfrak{s}^{-1} \end{matrix} \left[\begin{array}{c} b \\ \hline A \end{array}\right] \begin{matrix} T^{-t} \\ \longmapsto \end{matrix} \begin{matrix} \mathfrak{h}_1{}^{-1} \\ \mathfrak{h}^{-1} \end{matrix} \left[\begin{array}{c} O_{1\times n} \\ \hline A_2 \end{array}\right]$$

As we have explained above, we have that $A = HA_2$ and the relation $(\mathfrak{h}^{-1}, A_2) \xrightarrow{H} (\mathfrak{s}^{-1}, A)$. It holds:

$$As = b \iff AH^{-1}Hs = b \iff A_2(Hs) = b$$

Since, $Ob \subset \mathrm{Mod}(\mathcal{A}_2)$, we have that $(\mathfrak{h}, Hs)$ is integral. It follows that $(\mathfrak{h}\mathfrak{s}^{-1}, H)$, where $\mathfrak{h}\mathfrak{s}^{-1} = (\mathfrak{h}_i\mathfrak{s}_i^{-1})_{2\le i\le n+1}$, is a triangular denominator for $\mathcal{S} = (\mathfrak{s}, s)$. By Theorem 2.4.22, the maximal denominator ideal of $\mathcal{S} = (\mathfrak{s}, s)$ is $\prod_{i=2}^{n+1} \mathfrak{h}_i\mathfrak{s}_i^{-1} \det(H)$. The maximal denominator is the exponent ideal, from which it follows that $(\mathfrak{h}\mathfrak{s}^{-1}, H)$ is the minimal triangular denominator. □

Even though Algorithm 8 `PseudoDeterminant` theoretically correct, it is very slow in practice. In the next two sections, we discuss a method to replace the `Pseudo_HNF` computation by an optimized algorithm. After that, we can apply the normalization and reduction steps to avoid the coefficient swell problem (Section 2.4.6).

2.4.4 Euclidean Algorithm in O

A minimal triangular denominator is not necessary in Hermite form. Storjohann, in his algorithm `hcol` [PS13], applies unimodular row operations to triangularize the input matrix using extended gcd operations. Similarly, we use Algorithm 9 which generalizes the extended gcd to Dedekind domains ([Coh96, Theorem 1.2]). This is also a major step in pseudo-HNF algorithm and minimal triangular denominator computation.

Algorithm 9 Euclidean Step (`Euclidean_Step`)

Input: Two fractional ideals $\mathfrak{a}, \mathfrak{b}$ of $\mathcal{F}$ and $a, b \in \mathcal{F}$.
Output: $u, v \in \mathcal{F}$ such that $av + bu = 1$.

1: Compute $\mathfrak{d} = a\mathfrak{a} + b\mathfrak{b}$, $\mathfrak{d}^{-1}$, $\mathfrak{a}\mathfrak{d}^{-1}$ and $\mathfrak{b}\mathfrak{d}^{-1}$.
2: Compute $\bar{v} \in a\mathfrak{a}\mathfrak{d}^{-1}$ and $\bar{u} \in b\mathfrak{b}\mathfrak{d}^{-1}$ such that $\bar{v} + \bar{u} = 1$
3: **return** $v = \bar{v}a^{-1}$ and $u = \bar{u}b^{-1}$.

Suppose we are in the process of triangularizing the pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i)_i, A)$ and working on the i-th column. Let $\mathfrak{a}_i$ and $\mathfrak{a}_j$ be the coefficient ideal corresponding to the i-th row A_i and j-th row A_j. We proceed as follows:

1. Set $\mathfrak{d} \leftarrow a_{j,i}\mathfrak{a}_j + a_{i,i}\mathfrak{a}_i$.
2. Compute $v \in \mathfrak{a}_j\mathfrak{d}^{-1}$ and $u \in \mathfrak{a}_i\mathfrak{d}^{-1}$ such that $a_{j,i}v + a_{i,i}u = 1$ using `Euclidean_Step`.
3. Set $(\mathfrak{a}_j, \mathfrak{a}_i) \leftarrow (\mathfrak{a}_j\mathfrak{a}_i\mathfrak{d}^{-1}, \mathfrak{d})$ and $(A_j, A_i) \leftarrow (a_{i,i}A_j - a_{j,i}A_i,\ vA_j + uA_i)$.

More precisely:

$$\begin{matrix} \mathfrak{a}_i \\ \mathfrak{a}_j \end{matrix} \begin{pmatrix} * & a_{i,i} & * \\ * & a_{j,i} & * \end{pmatrix} \xrightarrow{T_j} \begin{matrix} \mathfrak{d} \\ \mathfrak{a}_j\mathfrak{a}_i\mathfrak{d}^{-1} \end{matrix} \begin{pmatrix} * & 1 & * \\ * & 0 & * \end{pmatrix}, \tag{2.4.4}$$

where the transformation matrix is $T_j = \begin{pmatrix} u & v \\ -a_{j,i} & a_{i,i} \end{pmatrix}$.

Experimental results show that these idempotents u and v computed in the HNF algorithm start large and then collapse. It is also observed in the `hcol` algorithm, as the gcd of arbitrary integers eventually becomes one. In the computation of the denominator ideal (Algorithm 7) we use the inverse transformation matrix of the HNF, which involve u and v terms as entries. This results in a huge unbalanced matrix making the computation more complicated, and we address this problem in Section 2.4.6.

2.4.5 Minimal Triangular Denominator

We can replace Step 10 in Algorithm 8 with a cheaper algorithm, as we do not need to compute the pseudo-HNF $\mathcal{H}$ of $\mathcal{S} = ((\mathfrak{s}_i)_{1\le i\le n+1}, S)$ precisely. In this section, we first explain the structure of $\mathcal{H}$, which can be obtained by algorithm `Euclidean_Step` and row operations. Then, we present an optimized algorithm, which constructs the pseudo-HNF of $\mathcal{S}$ as it is. Consider the matrix of $\mathcal{S}$:

$$S = \left[\begin{array}{c|c} 1 & O_{1\times n} \\ \hline -s & I_{n\times n} \end{array}\right] \tag{2.4.5}$$

Let $\bar{S}$ be the first column of S and take $\bar{S} = [1, s_2, \ldots, s_{n+1}]^t = [1 \mid -s]^t$ where $s = [s_2, \ldots, s_{n+1}]$ is the solution of the the linear system $Ax = b$ (Step 7). As explained in Section 2.4.4, we apply unimodular transformations (T_j in (2.4.4) for $j = 2, \ldots, n+1$ and $i = 1$) on S to reduce the first column vector $\bar{S}$, (starting from first two rows to the last row keeping the pivot in $S_{1,1}$):

The first step of Cohen's HNF algorithm is to create the upper-triangular matrix (See [Coh96, Algorithm 2.6]). In our case, because of the structure of S, we have to reduce the first column of S and replace coefficient ideals accordingly. In another words, we apply the transformation $T = T_{n+1}T_n \cdots T_2$ to $\mathcal{S}$. Then, we obtain an upper-triangular matrix H of the pseudo matrix denoted by: $\mathcal{H} = ((\mathfrak{h}_i), H) =$

$$\begin{array}{c} \mathfrak{h}_1 \\ \mathfrak{h}_2 \\ \mathfrak{h}_3 \\ \vdots \\ \mathfrak{h}_{j-2} \\ \mathfrak{h}_{j-1} \\ \mathfrak{h}_j \\ \vdots \\ \mathfrak{h}_{n-1} \\ \mathfrak{h}_n \\ \mathfrak{h}_{n+1} \end{array} \begin{pmatrix} 1 & v_2 & u_2v_3 & \cdots & u_2u_3\cdots u_{j-1}v_j & \cdots & u_2u_3\cdots u_nv_{n+1} \\ 0 & 1 & -v_3s_2 & \cdots & -u_3u_4\cdots u_{j-1}v_js_2 & \cdots & -u_3u_4\cdots u_nv_{n+1}s_2 \\ 0 & 0 & 1 & \cdots & -u_4u_5\cdots u_{j-1}v_js_3 & \cdots & -u_4u_5\cdots u_nv_{n+1}s_3 \\ \vdots & \vdots & & \ddots & \vdots & \vdots & \vdots \\ 0 & & & & -u_{j-1}v_js_{j-2} & \cdots & -u_{i-1}\cdots u_nv_{n+1}s_{i-2} \\ 0 & & & & -v_js_{j-1} & \cdots & -u_i\cdots u_nv_{n+1}s_{i-1} \\ 0 & & & & 1 & \cdots & -u_{i+1}\cdots u_nv_{n+1}s_i \\ \vdots & \vdots & & & & \ddots & \vdots \\ 0 & & & & & & -u_nv_{n+1}s_{n-1} \\ 0 & & & & & & -v_{n+1}s_n \\ 0 & & & & & & 1 \end{pmatrix}$$

Now we can represent matrix entries of H with the following equation. Even though $s_1 = 1$, in the following presentation, we consider $s_1 = -1$, to avoid the repetition of one more case.

$$H_{i,j} = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ -v_js_i\prod_{l=i+1}^{j-1} u_l & \text{if } i < j \quad (\text{Note} : s_1 = -1) \end{cases}$$

Finally, columns of the matrix H can be reduced, using row operations as follows:

1: **for** $j = 1, \ldots, n+1$ **do**
2: **for** $i = 1, \ldots, j-1$ **do**
3: Find $r_{i,j} \in \mathfrak{h}_j \mathfrak{h}_i^{-1}$ such that $H_{i,j} - r_{i,j}$ is small. (See [Coh96, Corollary 2.10])
4: Set ith row of H: $H_i = H_i - r_{i,j} H_j$ and, ith row of T: $T_i = T_i - r_{i,j} T_j$.
5: **end for**
6: **end for**

Now, we can define matrix entries of the pseudo-matrix $\mathcal{H} = ((\mathfrak{h}_i), H)$, which is in pseudo-HNF as follows:

$$H_{i,j} = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ -v_j s_i - r_{i,j} & \text{if } i+1 = j \\ -v_j s_i \prod_{l=i+1}^{j-1} u_l + \sum_{k=i+1}^{j-1} \left(v_j s_k r_{i,k} \prod_{l=k+1}^{j-1} u_l \right) - r_{i,j} & \text{if } i+1 < j \end{cases} \tag{2.4.6}$$

Note: Here also we consider $s_1 = -1$.

The corresponding transformation matrix T can be obtained directly from row operations. Now we present the new algorithm which computes the pseudo-HNF $\mathcal{H}$ of the pseudo-matrix $\mathcal{S}$. Algorithm 10 is an extension of the algorithm `hcol` [PS13, Figure 1] which does not use explicit row operations to compute the HNF.

Example 2.4.26. *As explained in section 2.4.4, we can reduce the column vector* $S = [1, s_2, \ldots, s_{n+1}]^t$ *starting from first two rows:*

$$\begin{matrix} \mathfrak{s}_1 \\ \mathfrak{s}_2 \end{matrix} \begin{pmatrix} 1 \\ s_2 \end{pmatrix} \xrightarrow{T_1} \begin{matrix} \mathfrak{d} \\ \mathfrak{s}_2 \mathfrak{s}_1 \mathfrak{d}^{-1} \end{matrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{where } T_1 = \begin{pmatrix} u & v \\ -s_2 & 1 \end{pmatrix} \text{ and } \mathfrak{d} = s_2 \mathfrak{s}_2 + \mathfrak{s}_1.$$

We can continue this proces upto the last row element s_n *to obtain the required trasformation matrix* $T = T_n \cdots T_1$ *and coefficient ideals of* $\mathcal{H}$. *Alternatively, we can do the reduction, starting from* s_n *to* s_1 *and obtain* $T = T_1 T_2 \cdots T_n$. *Here we can see the construction of* T *for* $n = 3$: $T_1 T_2 T_3 =$

$$\begin{pmatrix} u_2 & v_2 & & \\ -s_2 & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} u_3 & & v_3 & \\ & 1 & & \\ -s_3 & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} u_4 & & & v_4 \\ & 1 & & \\ & & 1 & \\ -s_4 & & & 1 \end{pmatrix} = \begin{pmatrix} u_2u_3u_4 & v_2 & u_2v_3 & u_2u_3v_4 \\ -u_3u_4s_2 & 1 & -v_3s_2 & -u_3v_4s_2 \\ -u_4s_3 & & 1 & -v_4s_3 \\ -s_4 & & & 1 \end{pmatrix}$$

When the transformation $T = T_1 T_2 T_3$ *is applied to* $\mathcal{S}$:

$$\mathcal{S} = \begin{matrix} \mathfrak{s}_1 \\ \mathfrak{s}_2 \\ \mathfrak{s}_3 \\ \mathfrak{s}_4 \end{matrix} \begin{pmatrix} 1 & & & \\ s_2 & 1 & & \\ s_3 & & 1 & \\ s_4 & & & 1 \end{pmatrix} \begin{matrix} \\ T_1T_2T_3 \\ \longmapsto \\ \\ \end{matrix} \begin{matrix} \mathfrak{h}_1 \\ \mathfrak{h}_2 \\ \mathfrak{h}_3 \\ \mathfrak{h}_4 \end{matrix} \begin{pmatrix} 1 & v_2 & u_2v_3 & u_2u_3v_4 \\ & 1 & -v_3s_2 & -u_3v_4s_2 \\ & & 1 & -v_4s_3 \\ & & & 1 \end{pmatrix}$$

After the column reduction, we have:

$$\mathcal{S} \begin{matrix} \\ T' \\ \longmapsto \\ \\ \end{matrix} \begin{matrix} \mathfrak{h}_1 \\ \mathfrak{h}_2 \\ \mathfrak{h}_3 \\ \mathfrak{h}_4 \end{matrix} \begin{pmatrix} 1 & v_2 - r_{1,2} & u_2v_3 + v_3s_2r_{1,2} - r_{1,3} & u_2u_3v_4 + u_3v_4s_2r_{1,2} + v_4s_3r_{1,3} - r_{1,4} \\ & 1 & -v_3s_3 - r_{2,3} & -u_3v_4s_2 + v_4s_3r_{2,3} - r_{2,4} \\ & & 1 & -v_4s_3 - r_{2,4} \\ & & & 1 \end{pmatrix}$$

Algorithm 10 Pseudo Minimal Triangular Denominator (`PseudoHcol`)

Input: A pseudo-matrix $\mathcal{S} = ((\mathfrak{s}_i), S)$, where $S \in \mathcal{O}^{n\times n}$ as in (2.4.5).
Output: `Pseudo_HNF`$(\mathcal{S})$.

1: $\mathfrak{h}_1 = \mathfrak{s}_1/S_{1,1}$
2: $H = I_n$
3: $w = [1, -s]$
4: **for** $i = n, \cdots, 2$ **do**
5: $\quad \mathfrak{d} = S_{i,1}\mathfrak{s}_i + \mathfrak{h}_1$
6: $\quad$ Compute $v_i \in \mathfrak{s}_i\mathfrak{d}^{-1}$ and $u_i \in \mathfrak{h}_1\mathfrak{d}^{-1}$ such that $v_iS_{i,1} + u_i = 1$ using `Euclidean_Step`.
7: $\quad \mathfrak{h}_i = \mathfrak{h}_1\mathfrak{s}_i/\mathfrak{d}$
8: $\quad \mathfrak{h}_1 = \mathfrak{d}$
9: **end for**
10: $w_1 = -1$
11: **for** $j = 2, \cdots, n$ **do**
12: $\quad$ **if** $\mathfrak{h}_j \neq 1$ **then**
13: $\quad\quad$ **for** $i = 1, \cdots, j-1$ **do**
14: $\quad\quad\quad H_{i,j} = -v_jw_i \pmod{\mathfrak{h}_j\mathfrak{h}_i^{-1}}$
15: $\quad\quad\quad w_i = w_i + H_{i,j}w_j$
16: $\quad\quad$ **end for**
17: $\quad$ **end if**
18: **end for**
19: **return** $\mathcal{H} = ((\mathfrak{h}_i), H)$

Theorem 2.4.27. *Algorithm 10 correctly computes the pseudo-HNF.*

Proof. The first part of the algorithm computes the coefficient ideals of the matrix in pseudo-HNF, which follows from the construction as explained above in Section 2.4.4. For the second part, we define $w_i^{(j)}$ to be the updated w_i at jth iteration in Step 15: $w_i^{(j)} = w_i^{(j-1)} + H_{i,j-1}w_{j-1}$. Take $w_i^{(1)} = 0$. We prove by induction on j that (2.4.7) holds for a fixed i.

$$w_i^{(j)} = \begin{cases} w_i & \text{if } i+1 = j \\ w_i \prod_{l=i+1}^{j-1} u_l - \sum_{k=i+1}^{j-1} \left(w_k r_{i,k} \prod_{l=k+1}^{j-1} u_l \right) & \text{if } i+1 < j \end{cases} \tag{2.4.7}$$

For the base case $j = 2$, we have $w_i^{(2)} = w_i^{(1)} + H_{1,1}w_1$. Hence, (2.4.7) holds because $w_i^{(1)} = 0$ and $H_{1,1} = 1$.

Now assume that (2.4.7) holds for $j = t-1$ where $3 \leq t < n+1$. Let $r_{i,k} \in \mathfrak{h}_k\mathfrak{h}_i^{-1}$ such that $-v_kw_i^{(k)} = H_{i,k} + r_{i,k}$ for $k = i+1, \cdots, t-1$ as computed in Step 14. Then, we can show that (2.4.7)

holds for $j = t$ as bellow. First, consider the case $i < t - 2$:

$$\begin{aligned} w_i^{(t)} &= w_i^{(t-1)} + H_{i,t-1} w_{t-1} \\ &= w_i^{(t-1)} - w_{t-1}(v_{t-1} w_i^{(t-1)} + r_{i,t-1}) \\ &= w_i^{(t-1)}(1 - v_{t-1} w_{t-1}) - w_{t-1} r_{i,t-1} \\ &= w_i^{(t-1)} u_{t-1} - w_{t-1} r_{i,t-1} \\ &= u_{t-1}\Big(w_i \prod_{l=i+1}^{t-2} u_l - \sum_{k=i+1}^{t-2} (w_k r_{i,k} \prod_{l=k}^{t-2} u_l)\Big) - w_{t-1} r_{i,t-1} \\ &= w_i \prod_{l=i+1}^{t-1} u_l - \sum_{k=i+1}^{t-1} (w_k r_{i,k} \prod_{l=k+1}^{t-1} u_l) \end{aligned} \tag{2.4.8}$$

Now, we suppose that $i = t - 2$. Substituting $w_i^{(t-1)} = w_i$ to (2.4.8), we get $w_i^{(t)} = u_{t-1} w_i - w_{t-1} r_{i,t-1}$ which is of the form (2.4.7). This shows that (2.4.7) holds for $j = t$. Now we can derive matrix entries of pseudo-HNF matrix $\mathcal{H} = ((\mathfrak{b}_i), H)$ using Step 14 with the remainder $r_{i,j}$ as explained before.

$$H_{i,j} = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ -v_j w_i - r_{i,j} & \text{if } i + 1 = j \\ -v_j w_i \prod_{l=i+1}^{j-1} u_l + \sum_{k=i+1}^{j-1} \left(v_j w_k r_{i,k} \prod_{k+1}^{j-1} u_l\right) - r_{i,j} & \text{if } i + 1 < j \end{cases}$$

This is exactly the same representation as (2.4.6), and we fixed the sign of s_1 in Step 10. This shows that Algorithm 10 is correct and it constructs the pseudo-HNF of a pseudo matrix of the form (2.4.5). □

2.4.6 Normalization and Reduction

In Algorithm 7, we compute the pseudo-HNF instead of computing the minimal triangular denominator. Experimental results show that in both approaches, the matrix T^{-1} becomes an unbalanced matrix which contains entities with few big-integer coefficients and many zeros. Eventually this would result in an expensive pseudo-HNF (Step 4) computation and linear system solving (Step 1), even for small matrices. Also, faster algorithms in vector reconstruction methods fail in the presence of unbalanced linear systems. Biasse and Fieker in their paper [BF12] provide a new polynomial-time algorithm to prevent the coefficient explosion in the HNF computation of an $\mathcal{O}$-module based on the modular approach of Cohen [Coh12, Chaper 1].

Algorithm 11 Normalization of a one-dimensional module (`Normalization`)

Input: One dimensional matrix $A \in K^{n\times 1}$ and $\mathfrak{a} \in \mathcal{F}$.
Output: $A' \in K^{n\times 1}$, $\mathfrak{a}' \in \mathcal{O}$ such that $N(\mathfrak{a}') \leq \gamma^{d^2} \sqrt{|\Delta_K|}$ and $\mathfrak{a}A = \mathfrak{a}'A'$, and a scalar $(\alpha/k) \in K$.

1: $\mathfrak{a} = k_0\mathfrak{a}$, $A = A/k_0$ where k_0 is the denominator of $\mathfrak{a}$.
2: $\mathfrak{b} = k\mathfrak{a}^{-1}$ where k is the denominator of $\mathfrak{a}^{-1}$.
3: α = shortest basis element of $\mathfrak{b}$ (the first element of a lattice reduced basis of $\mathfrak{b}$).
4: **return** $\mathfrak{a}' = (\alpha/k)\mathfrak{a}$, $A' = (k/\alpha)A$ and (α/k).

The strategy to prevent the coefficient swell is normalization which is the key difference between their approach and the one of [Coh12, Section 1.5]. The first step of Algorithm 11 ensures that the coefficient ideals are integral. Since $\mathfrak{a}_i A_i = \alpha\mathfrak{a}_i(1/\alpha)A_i$, for each of the row as a $\mathcal{O}$-module, we can adjust the coefficient ideals by scalars from K. Hence, given an integral ideal $\mathfrak{a}$ of K, the algorithm finds an integral ideal $\mathfrak{b}$ such that $\mathfrak{a}\mathfrak{b}^{-1}$ is principal and $N(\mathfrak{b})$ is bounded by field invariants only. In this

regards, the L^2-algorithm is applied to find a small integral representative of the ideal class of $\mathfrak{a}$. The given bound on the shortest basis element α depends on the discriminant Δ_K of the number field K and the constant γ which depends on the choice of the basis reduction algorithm. See [BFH17, Section 4].

Given an ideal $\mathfrak{a} \in \mathcal{F}$ and $\alpha \in K$, `Reduction`$(\alpha, \mathfrak{a})$ computes $\bar{\alpha} \in K$ such that $\alpha - \bar{\alpha} \in \mathfrak{a}$ and $\sqrt{T_2(\bar{\alpha})}$ is bounded. In the pseudo-HNF computation, this reduction step helps to achieve a polynomial complexity for the algorithm [BF12, Algorithm 3]. The algorithm `Reduction` which was introduced in [BF12] has been generalized for integral ideals with a detailed complexity analysis in [BFH17, Proposition 15 & Algorithm 1].

Algorithm 12 Reduction and normalization of a module (`ReductionNormalization`)

Input: A pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i)_{1 \le i \le n}, A)$ which is in Hermit form and the inverse $U \in K^{n \times n}$ of the corresponding transformation matrix.

Output: Normalized and reduced pseudo-matrix $\mathcal{A}$ with the updated inverse transformation matrix U.

for $i = 2, \ldots, \text{rows}(A)$ **do**
  for $j = 1, \ldots, i-1$ **do**
    if $A_{j,i} = 0$ **then**
      continue
    end if
    $\mathfrak{I} = \mathfrak{a}_i A_{i,i} \mathfrak{a}_j^{-1}$
    $\alpha =$ `Reduction`$(A_{j,i}, \mathfrak{I})$
    $q =$ `divexact`$(A_{j,i} - \alpha, A_{i,i})$
    Add $-q\times$ i-th row of A to the j-th row of A.
    Add $q\times$ j-th column of U to the i-th column of U.
  end for
end for
for $i = 1, \ldots, \text{rows}(A)$ **do**
  $\mathfrak{a}_i, A_i, s =$ `Normalization`$(\mathfrak{a}_i, A_i)$
  $U_i^t = sU_i^t$
end for
return $((\mathfrak{a}_i)_{1 \le i \le n}, A)$ and U.

Since we have a special structure in the input pseudo-matrix, we don't use the pseudo-HNF algorithm. We use the optimized `PseudoHcol` Algorithm 10 to get the Hermit normal form on the pseudo-matrix. Then we use the normalization and reduction steps to avoid the failures in the determinant computation as we mentioned above. Algorithm 12 is applied to the output of the `PseudoHcol` algorithm with the inverse of the transformation matrix, to obtain a nice pseudo-matrix (As explained in [BF12], practically we could obtain a balanced output). Here we denote the i-th row of the matrix A as A_i and the operation `divexact`(a, b) for $a, b \in K$ returns a/b where the quotient is expected to be exact.

2.4.7 Completion of the Main Algorithm and Complexity Analysis

In order to optimize the Algorithm 8 for determinant ideal computation, we require one more algorithm for the completion.

Algorithm 13 Inverse of an upper-triangular matrix (`InverseUTriang`)

Input: Non-singular upper-triangular matrix $T \in K^{n\times n}$
Output: Inverse matrix of T

1: $V = I_{n\times n}$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad V_{i,i} = 1/T_{i,i}$
4: $\quad$ **for** $j = i+1, \ldots n$ **do**
5: $\quad\quad V_{i,j} = -(\sum_{k=i}^{j-1} V_{i,k}T_{k,j})/T_{j,j}$
6: $\quad$ **end for**
7: **end for**
8: **return** V

Theorem 2.4.28. *Algorithm 13 is correct.*

Proof. Given a non-singular matrix $T \in K^{n\times n}$, the algorithm computes the inverse of T by solving the linear system $UT = I_{n\times n}$. Since, U is an upper-triangular matrix, the diagonal satisfies $U_{i,i} = 1/T_{i,i}$ for $i = 1, \ldots, n$. The rest follows from the construction. We compute $U_{i,j}$ row wise (given i, compute $U_{i,j}$ for $j = i+1, \ldots, n$), by solving equations $U_{i,i}T_{i,j} + U_{i,i+1}T_{i+1,j} + \ldots + U_{i,j-1}T_{j-1,j} + U_{i,j}T_{j,j} = 0$. □

In the `PseudoDeterminant` algorithm need the inverse transformation matrix of the pseudo-HNF of $\mathcal{S}$. Since, `PseudoHcol` does not compute the transformation matrix, we use Algorithm 13 to compute the inverse transformation separately. We replace Step 10 of Algorithm 8 by the following steps with the input pseudo-matrix $\mathcal{S} = ((\mathfrak{s}_i)_{1\le i\le n+1}, S)$.

$\mathcal{H} = ((\mathfrak{h}_i)_{1\le i\le n+1}, H) =$ `PseudoHcol`$(\mathcal{S})$
$U = S($`InverseUTriang`$(H))$
$\mathcal{H}, U =$ `ReductionNormalization`$(\mathcal{H}, U)$

Note that even though we have used the matrix (2.4.5) in $\mathcal{S}$, for better understanding of the construction, it can be changed into an integral matrix as: $S = \left[\begin{array}{c|c} z & O_{1\times n} \\ \hline -zs & I_{n\times n} \end{array}\right]$, where z is the common denominator of s, which can be obtained by `VecAlgRecon` directly. All the theorems, proofs and the implementation work with this new form of the matrix S.

We analyze the complexity of our approach over $\mathbb{Q}$. As we explained before, once we have the complete determinant ideal $\mathfrak{D}$, we use CRT approach to recover the determinant of the matrix A from units of the base field K. This is done by passing the computed $\mathfrak{D}$ to Step 4 of Algorithm 6. Hence, the complexity of the `ModularDeterminant` algorithm is as before, except for the above mentioned steps. More details regarding size and complexities can be found in [BFH17].

Let $s_A = d\log(\|A\|_\varphi)$ and $d\log(\|\det(A)\|_\varphi) \le B$, then we have that $B \in O(ns_A)$.

- `PseudoHcol`$(\mathcal{S})$: The first part of the algorithm uses ideal arithmetic which is an application of `Euclidean_Step`, n times with the cost approximately $O^{\sim}(ndM_d(B)))$.

  The second part computes each entry of the matrix H. The cost to compute $n^2/2$ entries is approximately $O^{\sim}(n^2dM_d(B))$.

- `InverseUTriang`(H): Algorithm computes computes each entry of the inverse matrix. First the diagonal is constructed from the inverses of existing diagonal entries with the cost $O^{\sim}(ndM_d(B))$. Worst case complexity for computing the strictly upper-triangular part of H^{-1} takes $O^{\sim}(n^2M_d(B))$ bit operations (n-number of element products with n-number of summation to compute $n^2/2 - n$-number of coefficients).

- `ReductionNormalization`$(\mathcal{H}, U)$: Reduction is done for all the non-zero entries of the matrix H. Worst case the complexity is $O^{\sim}(nd^5(B/d)^2)$ for applying the lattice reduction to n fractional ideals. The reduction output is normalized with the same complexity, and applied the corresponding changes to matrix entries with the total cost $O^{\sim}(nd^5(B/d)^2 + ndM_d(B))$. Here, B is the worst case bound on the size of matrix entries and ideals. After the redcution and normalization the paseudo-matrix has entries of the size s_A. This phenomenon has been observed in the integer case as explained in [PS13, Section 6], without any reasoning.

- The determinant ideal $\mathfrak{D}$ is computed as $\mathfrak{D} = \det(\mathcal{D})$ where $\mathcal{D} = ((\mathfrak{b}_i)_{2\leq i\leq n+1}, D)$. Since D is an upper-triangular matrix, the computation of $\det(D)$ costs $O^{\sim}(nM_d(s_A))$. The product of two ideals using the two-element representation takes $O^{\sim}(M_d(s_A))$ operations as the storage requirement for one ideal is $O^{\sim}((d+2)\log(\min(\mathfrak{b}_i)))$ and we multiply generators. Therefore, the product of n-ideals cost $O^{\sim}(nM_d(s_A))$ operations and the total complexity of this step is $O^{\sim}(nM_d(s_A))$ approximately.

- Unimodular certification is done by comparing norms of the product of denominator ideals in each step with the norm of $\det(A)$. The $\text{norm}(\det(A))$ is computed from the determinant of an $nd \times nd$ integer matrix, which is constructed from $d \times d$ blocks of representstion-matrices for each entries of A. The integer-determinant computation takes a runtime of approximately $O^{\sim}(n^3d^3)\log(\|A\|_\varphi)$. The representation matrix of any $\alpha \in \mathcal{O}$ can be computed with the complexity $O^{\sim}(d^2M_d(s_A))$, and for the whole matrix, it cost $O^{\sim}(n^2d^2s_A)$.

Hence, the total complexity for determinant ideal computation is $O^{\sim}(ndM_d(B) + nd^5(B/d)^2)$, which is approximately $O^{\sim}(n^2d^2\log(\|A\|_\varphi) + n^2d^5\log^2(\|A\|_\varphi))$ bit operations. We need unimodular certification only as a verification of the determinant ideal. It uses norm computation with complexity $O^{\sim}((n^3d^3)\log(\|A\|_\varphi))$ bit operations, which can be replaced by the complexity of `UniCert` algorithm (Algorithm 14). In the Section 3, we discuss the obstracles to use the `UniCert` algorithm in details.

2.4.8 Performance Analysis Based on Timings

We have implemented the above method (see [Sur21]) for determinant computation using the Hecke [FHHJ17] software package. The timings in Table 2.2 show that the new method performs better than the existing implementation in Hecke which is based on the deterministic algorithm [BFH17, Algorithm 3]. To illustrate the efficiency of the new method, we have computed timings using number fields: $K_1 = \mathbb{Q}[x]/(x^3 + 7x + 1)$, $K_2 = \mathbb{Q}[x]/(f_2)$ and $K_3 = \mathbb{Q}[x]/(f_3)$ for $f_2 = x^{10} + \sum_{i=0}^{9} 2(-x)^i$ and $f_3 = x^{20} + \sum_{i=0}^{19} 2(-x)^i$.

Table 2.2 shows times in seconds (m -for minutes and h- for hours) for the new approach (which uses Algorithm 8 `PseudoDeterminant` (`PDet`)) for different choices of parameters: n- size of the matrix, r- range of the input coefficients and d- the degree of the number field. Closer inspection of the results shows that the new algorithm works much better for big matrices in small degree fields: for 500×500 matrix the new implementation computes the determinant, nearly 10 times faster than the existing one. Nevertheless, over fields with larger degrees, the implementation becomes slow, due to the expensive basis reduction algorithm. Also, the current unimodular certification method fails when the field degree increases, and we can solve this problem with the ideas from next sections. Over fields of fixed degrees, our approach performs well for large matrices, proving the complexity results in the previous section. The algorithm also includes a proof of correctness of the determinant, using unimodular certification.

d	r / n	$-10:10$			$-100:100$			$-10000:1000$		
		PDet	Hecke	Quot	PDet	Hecke	Quot	PDet	Hecke	Quot
3	50	0.64	0.27	0.4	0.88	0.34	0.4	0.91	0.62	0.7
	100	3.65	3.63	1.0	6.81	4.84	0.7	8.77	7.93	0.9
	150	12.86	17.07	1.3	17.09	24.03	1.4	19.00	41.56	2.2
	200	26.50	54.47	2.1	31.98	1.3 m	2.5	58.19	2.4 m	2.4
	250	52.73	2.3 m	2.6	1.2 m	3.4 m	2.8	1.6 m	6.3 m	4.0
	300	1.5 m	5 m	3.3	1.9 m	7.6 m	4.1	2.6 m	14.1 m	5.3
	500	7.1 m	50 m	6.6	8.2 m	1.2 h	9.0	14.7 m	2.4 h	9.6
10	50	13.81	2.65	0.2	12.59	3.23	0.3	21.67	5.26	0.2
	100	1 m	35.27	0.6	1.6 m	52.85	0.6	2 m	1.6 m	0.8
	150	3.2 m	3.24 m	1.0	3.8 m	5 m	1.3	6.7 m	9.2 m	1.4
	200	7.7 m	11.7 m	1.5	9.2 m	17.5 m	1.9	15.8 m	32.6 m	2.1
	250	16.3 m	29.4 m	1.8	19.4 m	48.1 m	2.5	43.2 m	1.5 h	2.0
	300	33.6 m	1.1 h	2.0	43.2 m	1.7 h	2.4	51.5 m	3.3 h	3.9
20	50	3.7 m	8.91	0.0	2.8 m	13.12	0.1	5.3 m	23.22	0.1
	100	13.3 m	2.8 m	0.2	29.7 m	4.2 m	0.1	34.5 m	7.4 m	0.2
	150	1.2 h	16 m	0.2	1.3 h	28 m	0.3	2 h	40.3 m	0.3
	200	2.3 h	53.9 m	0.4	5.7 h	1.6 h	0.3	3.7 h	2.3 h	0.6

Table 2.2: Timing for Determinant Computation Using Determinant Ideal.

2.4.9 Empirical Results

Here, we discuss some approaches which can be used to optimize the implementation.

Recovering Units

In the integer case, when the absolute value of the determinant is certified by the `UniCert` test, it is easy to recover the sign of the determinant by computing determinant modulo an odd prime. In our application, we can use a similar strategy to recover the missing unit, when the determinant ideal is at the hand. Specially for small degree number fields, we can extend this idea easily by computing all the units of K, and then recovering their exponents by CRT. This is not practical in fields of large degrees.

In our implementation, we can use a different approach. Consider Algorithm 8 for determinant ideal computation. For a matrix $A \in K^{n\times n}$, let $\mathfrak{D} = \det(\mathcal{D})$ where $\mathcal{D} = ((\mathfrak{b}_i)_i, D)$ be the output of the algorithm. Let $\mathcal{M} = ((\mathfrak{b}_i^{-1})_i, M)$ be the unimodular pseudo matrix which is tested to be `true` in Step 12. Due to the normalization and reduction steps, the coefficient ideals of $\mathcal{D}$ and $\mathcal{M}$ becomes trivial or very small. Now we can use the fact that $A = MD$, to compute $\det(A)$ easier. The matrix D ia an upper triangular matrix, hence the computation of $\det(D)$ is cheper. We can compute the determinant of M using CRT with few prime numbers, as $\det(M)$ is a small number or a unit. This computation is faster than we recovering units, using the CRT with the ideal $\mathfrak{D}$.

Unimodular Certification as a $\mathbb{Z}$-Module

Suppose we have to certify the unimodularity of a full-rank pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$ over $\mathcal{O}$ with $A \in \mathcal{O}^{n\times n}$ and associated module $M \subseteq \mathcal{O}^n$. Here, we explain a way to turn this pseudo-matrix into a

$dn \times dn$ matrix over integers ($\mathbb{Z}$-module M of rank dn), so that we can directly use the `UniCert` algorithm in [PS12, Figure 2]). For coefficient ideals $(\mathfrak{a}_i)_{1 \le i \le n}$, there is the isomorphism $\mathfrak{a}_i \to \mathbb{Z}^d$. Therefore, as a $\mathbb{Z}$-module, we have $M = \mathfrak{a}_1 A_1 + \cdots + \mathfrak{a}_n A_n \cong \mathbb{Z}^{dn}$. Consider the coefficient ideal $\mathfrak{a}_i$ and an element of the i-th column of A as $\beta \in O$. Let $\alpha_1, \ldots, \alpha_d$ be the HNF basis of $\mathfrak{a}_i$. Then the coefficients of the d products $\beta\alpha_1, \ldots, \beta\alpha_d$ form a $d \times d$ integer matrix. We can obtain the $dn \times dn$ integer matrix, by applying the same procedure for all the matrix entries of A. This construction has the complexity of $O^{\sim}(n^2 d^3)$ (See [BFH17, Section 6]).

Dixon Solver without Reconstruction

In our implementation, we use norm(det(A)), which is computed using the integer matrix of representation matrices of A as a tool for unimodular certification. We can also use this information to get rid of the expensive lattice reduction step in the reconstruction of the solution of $Ax = b$. Using the fact that, denominator is a divisor of the determinant, we can solve the linear system $Ax = \text{norm}(\det(A)) \cdot b$, instead of solving $Ax = b$. We can use Dixon solver without the reconstruction step as we do not get denominators for the solution. Eventually, the p-adic expansion gives the solution of the linear system, when the required precision for the prime power is reached. However, we have to iterate Dixon lifting steps to a higher precision than it is required for algebraic reconstruction. Therefore, in practice, we could not observe any runtime improvement, even though the complexity gets better than before.

Chapter 3

Unimodular Certification

In terms of cost, both in theory and in practice, the dominant step of the determinant algorithm for integer matrices is the unimodular certification. A deterministic unimodular certification algorithm was presented by Storjohann in [PS12]. It uses the asymptotically fast `double-plus-one` lifting (which works in the usual symmetric residue system) to compute the high-order residue. In this section, we generalize the unimodular certification with its double-plus-one lifting over pseudo matrices.

3.1 Unimodular Certification for Matrices over $\mathbb{Z}$

Given a square, non-singular, integer matrix $A \in \mathbb{Z}^{n\times n}$ along with a lifting modulus $p \in \mathbb{Z}$ relatively prime to the determinant of A, double-plus-one lifting begins with the initial inverse $B_0 = A^{-1} \pmod{p}$ computation, then a step of division-free quadratic lifting doubles the precision yielding $B_1 = A^{-1} \pmod{p^2}$. Then a step of linear lifting corrects the overflow occurred in previous step and increases the precision by a power of p yielding $B_2 = A^{-1} \pmod{p^3}$. Alternating between quadratic and linear lifting, at each stage, the precision of the computed inverse B_i (which is congruent to $A^{-1} \pmod{p^k}$ for some $k \in \mathbb{Z}$) is doubled, plus an additional power of p (yielding $B_{i+1} \cong A^{-1} \pmod{p^{2k+1}}$). (See [PS12])

The motivation to double-plus-one method is that: neither linear nor straight quadratic lifting is ideal to compute residues of high-order. Because linear lifting computes small coefficients at each step, but the total number of steps grows. Conversely, quadratic lifting requires only a few steps to reach the precision, but each subsequent residue is twice the size of the last. Double-plus-one interleaves linear and quadratic lifting such that it computes small coefficients and reach the required precision within a few steps.

Also it produces a high-order residue $R \in \mathbb{Z}^{n\times n}$ such that $A(A^{-1} \pmod{p^k}) = I + Rp^k$. If A is a unimodular matrix, then A^{-1} is integral and, it holds $A^{-1} = A^{-1} \pmod{p^k}$ for a large k. In other words, A^{-1} has a finite p-adic expansion with symmetric residues. Then the high-order residue R becomes a zero matrix, provided that p and k are large enough.

3.2 Unimodular Certification for Matrices over Number Fields

For the unimodular certification of pseudo matrices we apply the same argument as in the $\mathbb{Z}$ case. Here, we use the two element presentation for ideals. The algorithm checks whether the corresponding dual of the pseudo matrix $\mathcal{A} = ((\mathfrak{a}_i), A)$ is integral: by computing a high-order residue $P \in \mathcal{O}^{n\times n}$ such that $A^{-1}D = (A^{-1}D \pmod{p^k}) + A^{-1}Pp^k$ where D is a diagonal matrix containing each generators of $(\mathfrak{a}_i^{-1})_i$ on the diagonal. The basic idea is as follows:

$\mathcal{A} = ((\mathfrak{a}_i)_{1\le i\le n}, A)$
$D_1 = \texttt{diag}(\texttt{gen\_one}(\mathfrak{a}_i^{-1})_i)$, $D_2 = \texttt{diag}(\texttt{gen\_two}(\mathfrak{a}_i^{-1})_i)$.
Choose p and k appropriately.
for $j = 1, 2$ **and** $i = 0, \ldots, k$ **do**
  $S_0 = A^{-1}D_j \pmod p$
  Use `double-plus-one` to compute S_i and P_i such that $A^{-1}D_j = S_i + A^{-1}P_{j,i}p^{2^{i+1}-1}$.
end for
if $P_{j,i} = O_{n\times n}$ for $j = 1, 2$ and $i \le k$ **then**
  return true
end if

Here also we use the modular reduction in the symmetric range: for a number field element $a \in K$ and an integer p which is relatively prime to the denominator of a, $a \pmod p$ gives the number field element with unique coefficients a_i for $1 \le i \le d$ in the usual symmetric range modulo p. That is $a_i \pmod p \in [-\lfloor (p-1)/2 \rfloor, \lfloor p/2 \rfloor]$. The next two lemmas which will be used in the main theorem, directly follow from this explanation.

Lemma 3.2.1. $|(a \pmod p)_i|/p \le 1/2$

Lemma 3.2.2. *If $a_i \in \mathbb{Z}$ satisfies $|a_i| < p/2$ then $a \pmod p = a$.*

3.2.1 Linear and Quadratic Lifting

The following theorem provides the idea of linear lifting especially in our application (see [PS12, Theorem 3]).

Theorem 3.2.3. *Let $A \in K^{n\times n}$ be non-singular and p be relatively prime to* $\det(A)$. *If $A^{-1}b = S + A^{-1}Pp^k$ for some $b, S, P \in K^{n\times n}$ and $k \in \mathbb{Z}_{>0}$, then for $N \in K^{n\times n}$ such that $N = A^{-1}P \pmod{p^l}$ it holds $A^{-1}b = S + Np^k + A^{-1}P'p^{k+l}$ where $P' = (1/p^l)(P - AN)$.*

We have chosen $l = 1$ and $N = A^{-1}P \pmod p$ in this section. The Theorem 3.2.4 which gives the idea of division free quadratic p-adic lifting in our context is a generalization of [PS12, Theorem 4]

Theorem 3.2.4. *Let A and p be as before. For any b, B, S, R, P that satisfy $A^{-1} = B + A^{-1}Rp$ and $A^{-1}b = S + A^{-1}Pp$, we have $A^{-1} = B(I + Rp) + A^{-1}R^2p^2$ and $A^{-1}b = S + BPp + A^{-1}RPp^2$.*

Proof. Both side of the equation $A^{-1} = B + A^{-1}Rp$ is multiplied on the right by Rp, and we substitute in it self replacing $A^{-1}Rp$ by the new equation. $A^{-1} = B + A^{-1}Rp = B + BRp + A^{-1}R^2p^2$. Similarly, both side of the equation $A^{-1} = B + A^{-1}Rp$ are multiplied on the right by Pp and substitute $A^{-1}b = S + A^{-1}Pp$ replacing the term $A^{-1}Pp$ by the new equation. □

3.2.2 Double-Plus-One Lifting

In this section we devise a modified double-plus-one lifting step over $\mathcal{O}$, interleaving linear and quadratic lifting as usual in the integer case. Define $p_i = p^{2^{i+1}-1}$. Then $p_{i+1} = p_i^2 p$ for all $i \ge 0$. Here, we compute matrix $B_i \equiv A^{-1} \pmod{p_i}$ and matrix $S_i \equiv A^{-1}b \pmod{p_i}$ for $i = 0, 1, 2 \ldots$ in succession, as bellow.

We initialize with $B_0 = A^{-1} \pmod p$. It holds $A^{-1} = B_0 + A^{-1}R_0p$ for the residue $R_0 = (1/p)(I - AB_0)$. We initialize the expansion of $A^{-1}b$ by computing $S_0 = B_0b \pmod p$, which is congruent to $A^{-1}b \pmod p$. Then, we compute the residue $P_0 = (1/p)(b - AS_0)$, and we obtain $A^{-1}b = S_0 + A^{-1}P_0p$.

Combining linear lifting modulo p terms of $A^{-1}b$ and A^{-1}, we obtain the quadratic lifting terms as below using the Theorem 3.2.4:

$$A^{-1}b = S_0 + B_0P_0p + A^{-1}R_0P_0p^2$$
$$A^{-1} = B_0(I + R_0p) + A^{-1}{R_0}^2p^2$$

Now we compute the additive terms M_0 and N_0 which encode a single step of linear p-adic lifting as explained in the Theorem 3.2.3.

$$N_0 = A^{-1}R_0P_0 \pmod p = B_0R_0P_0 \pmod p$$
$$M_0 = A^{-1}{R_0}^2 \pmod p = B_0{R_0}^2 \pmod p$$

Then, it holds $A^{-1}b = \underbrace{S_0 + B_0P_0p + N_0p^2}_{S_1} + A^{-1}P_1p^3$, where the residue P_1 is $(1/p)(R_0P_0 - AN_0)$.

Similarly, $A^{-1} = \underbrace{B_0(I + R_0p) + M_0p^2}_{B_1} + A^{-1}R_1p^3$, where $R_1 = (1/p)({R_0}^2 - AM_0)$.

If, we let B_1 and S_1 be as above, then we have the same format:

$$A^{-1}b = S_1 + A^{-1}P_1p^3$$
$$A^{-1} = B_1 + A^{-1}R_1p^3$$

Again, we can obtain the division free quadratic lifting as bellow:

$$A^{-1}b = S_1 + B_1P_1p^3 + A^{-1}R_1P_1p^6$$
$$A^{-1} = B_1(I + R_1p^3) + A^{-1}R_1^2p^6$$

Continuing this, we obtain the following sparse inverse expansion, as a sum of matrices having small coefficients.

$A^{-1} \pmod{p_i} = (\cdots((B_0(I + R_0p_0) + M_0p^2)(I + R_1p_1) + M_1p_1^2) + \cdots + M_{i-1}p_{i-1}^2)$

The expansion for $A^{-1}b \pmod{p_i}$:

$$A^{-1}b \pmod{p_i} = S_0 + B_0P_0p_0 + N_0p^2 + B_1P_1p_1 + N_1p_1^2 + \cdots + N_{i-1}p_{i-1}^2. \tag{3.2.1}$$

3.2.3 Bounds on the Maximum Coefficient of $A^{-1}b$

Let $C = \max\{\|A\|_\varphi, \|b\|_\varphi\}$. Here, we derive a bound for the maximum coefficient of the numerators of matrix entries of $A^{-1}b$ in Lemma 3.2.5 (this is similar to the construction of Lemma 1.3.23 for bound on $\|\det(A)\|_\varphi$). Then, we use Lemma 3.2.5 and Lemma 1.3.22, to explains Algorithm 14 in the Theorem 3.2.6. Moreover, we can extend these results with (3.2.1) to a linear system solving algorithm. We could obtain better runtime as we use high-order lifting and a faster vector-reconstruction method (Section 1.3.6).

Lemma 3.2.5. *Suppose $A^{-1}b$ is integral. Then, it holds $\|A^{-1}b\|_\varphi \le (c_2^{1/2}(c_1nd)^{n/2})C^n$.*

Proof. Using the Hadamard bound, we can compute a bound on the size of coefficients of $A^{-1}b$ as follows. Consider the corresponding conjugate matrices of A and b as $A^{(i)}$ and $b^{(i)}$ for $i = 1, \ldots, d$. Let e be the matrix entry of A or b with the maximum norm. For all $i = 1, \ldots, d$, let $e^{(i)}$ be the corresponding ith conjugates of e and e_i be the ith coefficient of e. Since $A^{-1}b$ is integral, the denominators of the matrix entries of $(A^{(i)})^{-1}b^{(i)}$ are all one, and the absolute values of the numerators are bounded by $n^{n/2}|e^{(i)}|^n$.

Now we let m be the matrix entry (number field element) with the maximum coefficient of $A^{-1}b$ (in another word, matrix entry which has $\|A^{-1}b\|_\varphi$ as a coefficient). Applying T_2-norm, it satisfied: $T_2(m) \leq \sum_{i=1}^{d} n^n |e^{(i)}|^{2n}$. It is easy to see that: $\sum_{i=1}^{d} n^n |e^{(i)}|^{2n} = n^n \sum_{i=1}^{d} \left(|e^{(i)}|^2\right)^n \leq n^n \Big(\sum_{i=1}^{d} |e^{(i)}|^2 \Big)^n$ Now using the fact that $|e_i| \leq C$ we can change the norm and obtain a bound on coefficients: $\sum_{i=1}^{d} |e^{(i)}|^2 \leq c_1 \sum_{i=1}^{d} |e_i|^2 \leq c_1 \sum_{i=1}^{d} C^2 = c_1 d C^2$. Combining above three inequalities, we get: $T_2(m) \leq n^n (c_1 d C^2)^n$. Let m_i for $1, \ldots, d$ be the coefficients of m. Then, we have $\|A^{-1}b\|_\varphi \leq \big(\sum_{i=1}^{d} {m_i}^2 \big)^{1/2}$. Combining this with changing norms on m, we obtain a bound on $\|A^{-1}b\|_\varphi$ as follows: $\sum_{i=1}^{d} {m_i}^2 \leq c_2 T_2(m) \leq c_2 n^n (c_1 d C^2)^n \implies \|A^{-1}b\|_\varphi \leq (c_2^{1/2} (c_1 n d)^{n/2}) C^n$. □

Algorithm 14 Unimodular Certification (`UniCert`)

Input: A pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i), A)$.
Output: `true` if $\mathcal{A}$ is unimodular, `false` otherwise.

$c_1, c_2 =$ `norm_change_constants`(O).
$D_1 =$ `diag` (`gen_one` $({\mathfrak{a}_i}^{-1})_i$), $D_2 =$ `diag` (`gen_two`$({\mathfrak{a}_i}^{-1})_i$)
$C = \max\{\|A\|_\varphi, \|D_1\|_\varphi, \|D_2\|_\varphi\}$.
Choose a prime $p \in \mathbb{Z}$ with $p \nmid \det(A)$ and $p \geq \max\{10000, 3.61 n^2 d^2 c_1^2 c_2 C\}$
Choose k such that $p_{k-1}^2 \geq (c_2 (c_1 n d)^n)^{1/2} C^n / (n^2 d^2 C)$.
$B_0 = \mod (A^{-1}, p)$
$M_0 = B_0, R_0 = I$
for $i = 1, 2$ **do**
  $N_0 = \mod (B_0 D_i, p)$
  $P_0 = D_i$
  for $j = 0 : k$ **do**
    $T_p = R_j^2, U_p = R_j P_j$
    $R_{j+1} = \frac{1}{p}(T_p - A M_j)$
    $P_{j+1} = \frac{1}{p}(U_p - A N_j)$
    if $P_{j+1} = O$ **then**
      break
    end if
    $M_{j+1} = \mod (B_0 T_p, p)$
    $N_{j+1} = \mod (B_0 U_p, p)$
    if $j = k$ **then**
      return `false`
    end if
  end for
end for
return `true`

3.2.4 Algorithm for Unimodular Certification

Theorem 3.2.6. *Algorithm 14 is correct.*

Proof. The pseudo-matrix $\mathcal{A} = ((\mathfrak{a}_i), A)$ is unimodular if and only if `dual`$(\mathcal{A}) = \mathcal{A}_D = (({\mathfrak{a}_i}^{-1}), A^{-t})$ is integral. Let D be a diagonal matrix which has the generators of the ideals $({\mathfrak{a}_i}^{-1})_i$ on the diagonal (as explained in Section 1.3.9, generators can be computed, and in Algorithm 14 we take $D_{1i,i} =$ `gen_one`$({\mathfrak{a}_i}^{-1})$ and $D_{2i,i} =$ `gen_two`$({\mathfrak{a}_i}^{-1})$ for $i = 1, \ldots, n$). In order to decide the unimodularity of $\mathcal{A}$,

it is sufficient to check whether DA^{-t} is integral for both generator sets separate. Since D is diagonal it holds $DA^{-t} = A^{-1}D$. Therefore, Algorithm 14 checks whether $A^{-1}D$ is integral for the two cases $D = D_1$ and $D = D_2$. Now we prove the following claim for one case. Let $C = \max\{\|A\|_\varphi, \|D\|_\varphi\}$.

Claim: Let $p \geq \max\{10000, 3.61n^2d^2c_1^2c_2C\}$, then it holds that $\|B_i\|_\varphi, \|S_i\|_\varphi < 0.6p_i$, and $\|R_i\|_\varphi, \|P_i\|_\varphi < 0.6001ndc_1\sqrt{c_2}\,C$ for i, $0 \leq i \leq k$. Moreover, if k is chosen large enough to satisfy $p_{k-1}^2 \geq (c_2(c_1nd)^n)^{1/2}C^n/(n^2d^2C)$, then $A^{-1}D$ is integral if and only if P_k is the zero matrix.

Here, the bound on p (i.e. $p \geq 10000$) is chosen such that, the size of p does not effect to obtain the bounds on the intermediate output as shown in the proof, and any lower bound which is greater than 10000 would work.

Using induction on i, we will prove that the following identities and bounds hold on outputs.

$$A^{-1} = B_i + A^{-1}R_ip_i \quad \text{and} \quad A^{-1}D = S_i + A^{-1}P_ip_i \tag{3.2.2}$$

$$\|B_i\|_\varphi < 0.6p_i \quad \text{and} \quad \|S_i\|_\varphi < 0.6p_i \tag{3.2.3}$$

where $B_i = B_{i-1}(I + R_{i-1}p_{i-1}) + M_{i-1}p_{i-1}^2$ and $S_i = S_{i-1} + B_{i-1}P_{i-1}p_{i-1} + N_{i-1}p_{i-1}^2$.

For $i = 0$: the identities (3.2.2) directly follows from Theorem 3.2.3 (we can consider b to be the identity matrix in the first case). Since we use the modular reduction in symmetric range all the coefficient of matrix entries of B_0 and S_0 are lie in the range $[-\lfloor(p-1)/2\rfloor, \lfloor p/2\rfloor]$. Hence, the inequalities (3.2.3) holds for $i = 0$ due to 3.2.1.

Now we suppose that (3.2.2) and (3.2.3) hold for some i, $i \geq 0$. Applying division free quadratic lifting step from Theorem 3.2.4, we have

$$\begin{aligned} A^{-1} &= B_i(I + R_ip_i) + A^{-1}{R_i}^2p_i^2 \\ A^{-1}D &= S_i + B_iP_ip_i + A^{-1}R_iP_ip_i^2 \end{aligned} \tag{3.2.4}$$

Then, Theorem 3.2.3 gives:

$$\begin{aligned} A^{-1} &= B_i(I + R_ip_i) + M_ip_i^2 + A^{-1}R_{i+1}p_{i+1} \\ A^{-1}D &= S_i + B_iP_ip_i + N_ip_i^2 + A^{-1}P_{i+1}p_{i+1} \end{aligned}$$

which shows that (3.2.2) is satisfied for $i + 1$.

Given bounds on $\|B_i\|_\varphi$ and $\|S_i\|_\varphi$, we can compute bounds on $\|R_i\|_\varphi$ and $\|P_i\|_\varphi$ using (3.2.2).

$$\begin{aligned} R_i &= (1/p_i)(I - AB_i) \\ \|R_i\|_\varphi &\leq (1/p_i) + 0.6ndc_1\sqrt{c_2}\,\|A\|_\varphi \qquad \text{from } p \geq 10000, \text{ (3.2.3) and Lemma 1.3.22} \\ &\leq 0.6001ndc_1\sqrt{c_2}\,\|A\|_\varphi \\ &\leq 0.6001ndc_1\sqrt{c_2}\,C \end{aligned}$$

$$\begin{aligned} P_i &= (1/p_i)(D - AS_i) \\ \|P_i\|_\varphi &\leq (\|D\|_\varphi/p_i) + 0.6ndc_1\sqrt{c_2}\,\|A\|_\varphi \\ &\leq 0.0001\|D\|_\varphi + 0.6ndc_1\sqrt{c_2}\,\|A\|_\varphi \\ &\leq 0.6001ndc_1\sqrt{c_2}C \end{aligned}$$

The computed bounds on R_i and P_i gives the following two bounds, which can be used to prove (3.2.3) for $i+1$.

$$\begin{aligned}\|B_i(I+R_ip_i)\|_\varphi &\le \|B_i\|_\varphi + ndc_1\sqrt{c_2}\,\|B_i\|_\varphi\|R_i\|_\varphi p_i \\ &< 0.6p_i + ndc_1\sqrt{c_2}(0.6p_i)(0.6001ndc_1\sqrt{c_2}\,C)p_i \\ &= (0.6/p_i + 0.36006\,n^2d^2c_1^2c_2\,C)p_i^2 \\ &\le 0.36012\,n^2d^2c_1^2c_2\,C\,p_i^2 \end{aligned} \tag{3.2.5}$$

$$\begin{aligned}\|S_i + B_iR_ip_i\|_\varphi &\le \|S_i\|_\varphi + ndc_1\sqrt{c_2}\,\|B_i\|_\varphi\|R_i\|_\varphi p_i \\ &< 0.6p_i + ndc_1\sqrt{c_2}(0.6p_i)(0.6001ndc_1\sqrt{c_2}\,C)p_i \\ &= (0.6/p_i + 0.36006n^2d^2c_1^2c_2C)p_i^2 \\ &\le 0.36012n^2d^2c_1^2c_2\,Cp_i^2 \end{aligned} \tag{3.2.6}$$

Then, we apply computed bounds on expansions of B_{i+1} and S_{i+1} as bellow:

$$\begin{aligned}\|B_{i+1}\|_\varphi/p_{i+1} &= \|B_i(I+R_ip_i) + M_ip_i^2\|_\varphi/(p_i^2p) \\ &\le 0.36012n^2d^2c_1^2c_2C/p + 1/2 && \text{from (3.2.5) and Lemma 3.2.1} \\ &\le 0.6 && \text{as } p \ge 3.61n^2d^2c_1^2c_2C\}\end{aligned}$$

$$\begin{aligned}\|S_{i+1}\|_\varphi/p_{i+1} &= \|S_i + S_iR_ip_i + N_ip_i^2\|_\varphi/(p_i^2p) \\ &\le 0.36012n^2d^2c_1^2c_2C/p + 1/2 && \text{from (3.2.6) and Lemma 3.2.1} \\ &\le 0.6 && \text{as } p \ge 3.61n^2d^2c_1^2c_2C\end{aligned}$$

This shows that (3.2.3) holds for $i+1$ and similar to the case i the bound on R_{i+1} and P_{i+1} can be shown by induction.

Finally, we will show how to get the bound on number of lifting steps k needed, to certify the integrality of $A^{-1}D$. Suppose P_k is zero for the chosen k. Then, from (3.2.2) we have $A^{-1}D = S_k$ which shows it is integral. By Lemma 3.2.5, we have that $\|A^{-1}D\|_\varphi \le (c_2^{1/2}(c_1nd)^{n/2})C^n$. Then, we get the bound on $\|A^{-1}D\|_\varphi$ with respect to the chosen p_{k-1} such that $p_{k-1}^2 \ge (c_2(c_1nd)^n)^{1/2}C^n/(n^2d^2C)$:

$$\|A^{-1}D\|_\varphi \le n^2d^2Cp_{k-1}^2 \tag{3.2.7}$$

In Algorithm 14 at the step $k-1$ of double-plus one lifting, we have $S_{k-1} + B_{k-1}P_{k-1}p_{k-1} \cong A^{-1}D \pmod{p_{k-1}^2}$ from quadratic lifting. Equivalently, for some error term E we have

$$A^{-1}D = S_{k-1} + B_{k-1}P_{k-1}p_{k-1} + Ep_{k-1}^2 \tag{3.2.8}$$

The size of the error term can be bounded using the presumed size of p_{k-1}^2 and bounds on the other terms.

$$
\begin{aligned}
E &= (A^{-1}D - (S_{k-1} + B_{k-1}P_{k-1}p_{k-1}))/p_{k-1}^2 \\
\|E\|_\varphi &= \|(A^{-1}D - (S_{k-1} + B_{k-1}P_{k-1}p_{k-1}))\|_\varphi/p_{k-1}^2 \\
&\leq \|A^{-1}D\|_\varphi/p_{k-1}^2 + \|S_{k-1} + B_{k-1}P_{k-1}p_{k-1}\|_\varphi \\
&\leq n^2d^2C + 0.36012n^2d^2C && \text{from (3.2.6) and (3.2.7)} \\
&< p/2 && \text{as } p \geq 3.61n^2d^2C \text{ by assumption}
\end{aligned}
$$

Consider the quadratic lifting step: equation (3.2.4) for $i = k-1$. Now we can relate the residue P_{k-1} and R_{k-1} of (3.2.4) to the error E of the equation (3.2.8) and obtain:

$$A^{-1}R_{k-1}P_{k-1} = E. \tag{3.2.9}$$

Using linear p-adic lifting, the last step of the double-plus one lifting computes the correction $N_{k-1} = A^{-1}R_{k-1}P_{k-1}$ which is also equal to $E \pmod p$ by (3.2.9). Since the size of the error satisfies $\|E\|_\varphi < p/2$, by Lemma 3.2.2, we get $E \pmod p = E$. Hence N_{k-1} exactly captures the error: $N_{k-1} = E$.

Finally,

$$
\begin{aligned}
P_k &= (1/p)(R_{k-1}P_{k-1} - AN_{k-1}) && \text{by Lemma 3.2.3} \\
&= (1/p)(R_{k-1}P_{k-1} - AE) && \text{as } N_{k-1} = E \\
&= 0 && \text{by equation (3.2.9).}
\end{aligned}
$$

□

According to Algorithm 14, the double-plus-one lifting performs k lifting steps. The cost of each step being six multiplications of matrices in which the size of coefficients of each operands A, B_0, Ri, P_i, Mi, N_i are bounded by $M = O(\log(p) + \log(ndC))$.

Theorem 3.2.7. *The complexity of Algorithm 14 is $O^\sim(n^3d^2 + n^2d^3)$.*

Proof. The number of steps satisfies $k \in O(\log(n))$.

- The computation of $A^{-1} \pmod p$ has the complexity $O^\sim(n^3d^2)$.
- The complexity of one matrix multiplication is $O^\sim(n^2d^3)$.

Hence the total complexity is $O^\sim(n^3d^2 + n^2d^3)$ over $\mathbb{Q}$. □

3.3 Residue Number System

This chapter outlines an approach to obtain an optimized (high performance) implementation of the `UniCert` algorithm over number fields. As mentioned in [PS12] over integers, Storjohann's algorithm is efficient in practice. Even though we could expand the concept and obtain similar complexities over number fields, practically we could not achieve better performances as in [PS12], as we do not have access to a highly optimized library such as BLAS- Basic Linear Algebra Subprogram in [BPP+02] in Julia.

Residue Number System (RNS) is defined by a set of coprime integers $m = \{m_1, m_2, \ldots, m_k\}$, where each individual members is called a module. For any given base m, the residue representation of an integer x is the k-tuple, $(x_1, x_2, \ldots, x_k)$, where x_i are integers defined by $x_i = x \mod m_i$ and $0 \leq x_i < m_i$ for every i.

Basic Linear Algebra Subprograms over $\mathbb{Z}$

The overarching approach to the implementation is mainly due to the usage of a highly optimized implementation called the *Basic Linear Algebra Subprograms (BLAS)* [BPP+02]. It provides low-level linear algebra routines by the Automatically Tuned Linear Algebra Software (ATLAS)[WD98] which operates only on matrices of double-precision floating-point numbers. They use CRT modular scheme to work with matrices of arbitrary large integers. Since double-precision numbers can only be represented by integers less than 2^{53} (as 12 bits are used for the sign and exponent), computed values in each stage must meet this bound to maintain the exact result. Therefore, it computes the largest value p such that a residue number system backed with BLAS operations: the result of a product between two matrices of dimension n must be exactly representable in the 53 bits of a double's mantissa. Hence, each modulus p_i must satisfy $n(p_i - 1)^2 \leq 2^{53} - 1$ and the largest prime p_k in each basis is chosen such that $p_k \leq (\sqrt{4n(2^{53} - 1) + 1} + (2n - 1))/2n$.

High-Order Residue Implementation over Number Fields

In general, the idea to multiply together two integer matrix A and B over integers is:

(i) Use multi-modular reduction to get A and B in a residue number system (RNS) modulo a bunch of primes p that satisfy $n(p - 1)^2 \leq 2^{53} - 1$.

(ii) For each prime p in the residue number system compute the matrix product $AB \pmod{p}$ using the BLAS implementation.

(iii) Reconstruct AB over $\mathbb{Z}$ using CRT.

The main idea to obtain a highly optimized implementation is minimizing the number of calls to Step (*i*) and the expensive Step (*iii*), and keeping all matrices in the RNS throughout the computation. For example, in our application with the double-plus-one lifting, all matrix multiplications which are actually modulo $p = p_1 p_2 \cdots p_l$ and $q = q_1 q_2 \cdots q_m$, are carried out with matrices in the RNS with respect to p and q as explained below.

As was mentioned in the previous section, in the unimodular certification we compute two more parameters p and k (the base of the p-adic lifting and number of lifts required) to be used in the iteration with double-plus-one lifting. As required by Theorem 3.2.6, p and k were selected to be sufficiently large. Similar to the implementation of Storjohann, here also we employ a standard modular scheme to allow the algorithm to work with matrices of arbitrary large integers despite the limitations coming from the interface libraries. Using the bound on intermediate outputs R_i and P_i (Theorem 3.2.6), we choose moduli set $\mathcal{Q} = \{q_1, q_2, \cdots, q_m\}$ with $q = q_1 q_2 \cdots q_m$ such that $q \geq 1.2002ndc_1\sqrt{c_2}C$ and $n(q_i - 1)^2 \leq 2^{53} - 1$. Also, we use another set of prime moduli $\mathcal{P} = \{p_1, p_2, \ldots, p_l\}$ with $p = p_1 p_2 \cdots p_l$ such that $p \geq 3.61n^2d^2c_1^2c_2C$ and $n(p_i - 1)^2 \leq 2^{53} - 1$. Then, the p-adic lifting steps are replaced by operations modulo p_i for $i = 1, \ldots, l$. It is required to convert between two residue systems, while working with two co-prime bases. Hence the following conversion steps have to be introduced to Algorithm 14. The code `Convert`$(A_{\mathcal{P}}, \mathcal{Q})$ converts the array of matrices $A_{\mathcal{P}}$ which is over basis $\mathcal{P}$, to the basis $\mathcal{Q}$ and vice versa. Methods of conversions are explained in the next sub-section.

$B_0 = \mod (A^{-1}, \mathcal{P})$
$R_{\mathcal{P}0} = \mod (I, \mathcal{P})$, $P_{\mathcal{P}0} = \mod (b, \mathcal{P})$
for $j = 0, \cdots, k$ **do**
$T_{\mathcal{P}} = R_{\mathcal{P}j}^2$, $U_{\mathcal{P}} = R_{\mathcal{P}i} P_{\mathcal{P}i}$
$M_{\mathcal{P}j} = B_0 T_{\mathcal{P}} \pmod{\mathcal{P}}$
$N_{\mathcal{P}j} = B_0 U_{\mathcal{P}} \pmod{\mathcal{P}}$
$M_{Qj} = \texttt{Convert}(M_{\mathcal{P}j}, Q)$
$N_{Qj} = \texttt{Convert}(N_{\mathcal{P}j}, Q)$
$R_{Qj+1} = ((R_{Qj} - A_Q M_{Qj})/p) \pmod{Q}$
$P_{Qj+1} = ((P_{Qj} - A_Q N_{Qj})/p) \pmod{Q}$
$R_{\mathcal{P}j+1} = \texttt{Convert}(R_{Qj+1}, \mathcal{P})$
$P_{\mathcal{P}j+1} = \texttt{Convert}(P_{Qj+1}, \mathcal{P})$
end for

Here, $\mathcal{P}$ must be distinct from Q as p is required to be invertible with respect to the basis Q. Moreover, the two coprime moduli p and q must satisfy $p \geq 3.61n^2d^2c_1^2c_2C \geq q \geq 1.2002ndc_1\sqrt{c_2}C$.

3.3.1 Basis conversion in a residue number system

In this section we discuss few possible methods which can be used for converting between residues in the two co-prime residue number systems: Given a representation of $x \in \mathbb{Z}$, as $[[x]_{p_1}, [x]_{p_2}, \ldots, [x]_{p_l}]$, in basis $\mathcal{P}$ move to an equivalent representation in basis Q, as $[[x]_{q_1}, [x]_{q_2}, \ldots, [x]_{q_m}]$. Here, the square brackets with indices will denote modular computation $x \pmod{p} = [x]_p$. In our application, we can take $x = A$, and given the residues in base $\mathcal{P}$, the residue representation for the basis Q can be computed using same methods.

Conversion using CRT

We can use the CRT to get the fixed-radix representation from RNS as:

$$x = [x]_p = \Big[\sum_{i=1}^{l} s_i \big[[x]_{p_i} s_i^{-1} \big]_{p_i} \Big]_p,$$

where $s_i = p/p_i$ and s_i^{-1} the multiplicative inverse of $s_i \pmod{p_i}$, such that $s_i^{-1} s_i \pmod{p_i} = 1$. We can take the use of CRT to achieve the base extension as $[x]_{q_j} = \big[[x]_p\big]_{q_j}$ for $j = 1, 2, \ldots, m$. But the usual CRT computation is expensive, as the intermediate values such as the inner sum computed to the full precision could be arbitrary large. Hence, Shenoy and Kumaresan in [SK89] introduce an additional *redundant modulus* which helps to cut-down the bit length of the arithmetic to be used.

We can slightly modify the CRT result as $x = \sum_{i=1}^{l} s_i \big[[x]_{p_i} s_i^{-1} \big]_{p_i} - Rp$ with an unknown error term R. The correction term can be computed using the redundant module $\bar{p}$ and the corresponding redundant residue $[x]_{\bar{p}}$ as $R = \Big[\big[\frac{1}{p}\big]_{\bar{p}} \Big(\Big[\sum_{i=1}^{l} s_i \big[[x]_{p_i} s_i^{-1} \big]_{p_i} \Big]_{\bar{p}} - [x]_{\bar{p}} \Big) \Big]_{\bar{p}}$. The requirement that the redundant module has to obtained independently, makes this approach unusable in some applications. So Posch and Posch in [PP93] present a different method to compute the correction term R.

The Approximated Base Extension

Consider the equation $x = \sum_{i=1}^{l} s_i \big[[x]_{p_i} s_i^{-1} \big]_{p_i} - Rp$, dividing each term by p yields: $R + \frac{1}{p} x = \sum_{i=1}^{l} \frac{1}{p_i} \big[[x]_{p_i} s_i^{-1} \big]_{p_i}$. For each i, we denote the weights $w_i = [s_i^{-1}]/p_i$ and we can rewrite the equation as a weighted sum of all $[x]_{p_i}$ as: $R + \frac{1}{p} x = \sum_{i=1}^{l} [x]_{p_i} w_i$. Clearly R is an integer, since x is reduced

modulp p, A/p is a matrix having rational entries less than one. Thus we can ignore the term x/p and get an approximated expression for R as $R^* = \lfloor \sum_{i=1}^{l} [x]_{p_i} w_i \rfloor$ which gives either R or $R-1$. A method to figure-out the error term R, has been described in [PP93, Section 3] with an associate error bound ε on R^*. In this application $\varepsilon < 1/p_{\max}$, as explained in [Pau13, Section 4.2.1] for integer case. Hence, we get the error term R as $R = \lfloor \sum_{i=1}^{l} [x]_{p_i} w_i \rceil$, and once we have R in hand we can continue the basis conversion without reconstructing $[x]_p$ to the full precision: For $j = 1, 2, \ldots, m$ the representation corresponding to the basis Q is:

$$[x]_{q_j} = \Big[\Big[\sum_{i=1}^{l} \big[s_i \big[[x]_{p_i} s_i^{-1} \big]_{p_i} \big]_{q_j} \Big]_{q_i} - [pR]_{q_j} \Big]_{q_j}$$

The Mixed Radix Number System and Conversion

Given the radices $p_1, p_2, \ldots, p_l$, any $x \in \mathbb{Z}$ can be uniquely represented in mixed-radix form as $x = a_l \prod_{i=1}^{l-1} p_i + a_{l-1} \prod_{i=1}^{l-2} p_i + \cdots + a_2 p_1 + a_1$ where $a_i < p_i$. This particular formulation is called the *mixed radix number system* and this weighted, positional number representation is also called *mixed radix representation*. The coefficients a_i are called the mixed radix coefficients, which can be obtained recursively in $l-1$ steps from the residues $[x]_{p_1}, [x]_{p_2}, \ldots, [x]_{p_l}$ corresponding to the modulis $p_1, p_2, \ldots, p_l$. Given the residues we can briefly explain the construction of mixed-redix representation as follows:

For $i = 1$, let $a_i = [x]_{p_1}$. To find a_2, lets consider $x - a_1 = a_l \prod_{i=1}^{l-1} p_i + a_{l-1} \prod_{i=1}^{l-2} p_i + \cdots + a_2 p_1$ and reduction modulo p_2 yields $[x - a_1]_{p_2} = [a_2 p_1]_{p_2}$. Rearranging terms and substituting $[x]_{p_2}$ we get: $a_2 = \big[[p_1^{-1}]_{p_2} ([x]_{p_2} - a_1) \big]_{p_2}$. Continuing this process, the mixed radix coefficient a_l, can be retrieved from the residues as:

$$a_l = [[(\prod_{i=1}^{l-1} p_i)^{-1}]_{p_l} ([x]_{p_l} - (a_{l-1} \prod_{i=1}^{l-2} p_i + \cdots + a_2 p_1 + a_1))]_{p_l}.$$

For computational purposes we can reform these terms to exploit some more parallelism with a recurrence relation (refer [OP07, Section 7.2]). Thus, the residues in a base extended modulus can be obtained by computing

$$[x]_{p_{l+1}} = \Big[[a_l \prod_{i=1}^{l-1} p_i]_{p_{l+1}} + [a_{l-1} \prod_{i=1}^{l-2} p_i]_{p_{l+1}} + \cdots + [a_2 p_1]_{p_{l+1}} + [a_1]_{p_{l+1}} \Big]_{p_{l+1}}.$$

3.3.2 Conclusion

Depending on the method we used for the `Convert` implementation, the quantities s_i, multiplicative inverse s_i^{-1} weights w_i and $\prod_{j=1}^{i} p_j$ can be used throughout the algorithm as precomputed data. The new unimodular certification algorithm uses higher-order liftings and residue number systems. The same algorithm can be extended to solve linear systems using fast lifting methods. In our implementation, we use residue fields of O corresponding to degree one primes(totally split primes) to obtain better performance as in the integer case. We have been able to extend the techniques which are used in the state of the art method for determinant computation in [PS13] to number fields. We could achieve theoretically faster methods with better complexities. In communications with Storjohann, we figured out that we need the access to a highly optimized low-level libraries such as such as BLAS [BPP+02], ATLAS [WD98] or OpenBLAS, for further improvement. However, without having a highly optimized library, it is difficult to obtain a fast implementation for unimodular certification. The purpose of this chapter was to provide suggestions for further improvement and speculating on future directions.

Chapter 4

Algorithms for Solving Non-square Linear Systems over Number Fields

In this chapter we present a deterministic algorithm for solving a non-square linear system over number fields. As the solution is not unique, we compute a basis for the kernel to normalize the solution. The implementation accompanied by a fast algorithm to compute the kernel of a matrix of any size. In both two cases, a modified version of the Dixon algorithm is used. Preconditioning techniques can be applied to optimize this computation using maximum rank sub-system. The algorithm works for integer matrices as well. We rigorously assess its complexity as $O^{\sim}(m^3d^2 + m^2nd + m^2d^5)$ bit operations, where as the Gaussian method takes $O^{\sim}(m^3n^2d^2)$ bit operations to solve a linear $m \times n$ system $Ax = b$ over number field K of degree d.

4.1 Introduction

Solving linear nonsingular, square systems is a well studied classical problem. The classical method using Cramers rule and Gaussian elimination methods would take cubic number of operations. Here also we face the computational phenomenon: *intermediate coefficient swell*. Cabay and Lam in [CL77] introduced a CRT (Chinese Remainder Theorem) based modular method: one can compute the solution modulo many small prime numbers such that the product of primes is large enough. This computation takes $O^{\sim}(n^4)$ number of arithmatic operations for an $n \times n$ system $Ax = b$ over $\mathbb{Q}$. Dixon in [Dix82] presented a p-adic Hensel lifting method for linear system solving which can accelerate the bit complexity beyond CRT approach. There are two phases for the algorithm. First, the inverse $C = A^{-1} \pmod p$ is computed with complexity $O^{\sim}(n^3)$. The second step computes a p-adic approximation $\bar{x}$ with $O(ln^2 \log n)$ integer operations for l iterations. (See Algorithm 3) Dixon's algorithm is efficient in practice with the total complexity $O(n^3 \log^2 n)$.

We have already generalized DixonâĂŹs algorithm to number fields. Here we generalize it again to non-square underdetermined systems. We use Hadamard's determinant estimate with an apropriate norm, to bound rational enries in the solution as shown by the Cramer's rule. Reduced-row echelon form is used as a preconditioning technique to find a maximum rank subsystem.

This new deterministic algorithm takes $O^{\sim}(m^3d^2 + m^2nd + m^2d^5)$ bit operations, where as the Gaussian method takes $O^{\sim}(m^3n^2d^2)$ bit operations to solve a linear $m \times n$ system $Ax = b$ over number field K of degree d. The total complexity for an integer linear system is $O(m^2n + m^3 \log^2 m)$ in $\mathbb{Z}$, same as the Dixon algorithm.

4.1.1 Reduced Row Echelon Form

The main tool we use in our linear system solving and kernel comptation algorithms is *elementary row and column operations*. In this section we consider F to be any field and a matrix $A \in F^{m\times n}$ (or, equivalently, its associated linear map $f_A : F^n \to F^m$).

Lemma 4.1.1. *If $A \in F^{m\times n}$ is a matrix and R is obtained from A by a sequence of elementary row operations, then* $\ker(A) = \ker(R)$.

Proof. Let $R = TA$, where T be the transformation matrix which is invertible, then we have
$x \in \ker(A) \iff Ax = 0 \iff TAx = 0 \iff Rx = 0 \iff x \in \ker(R)$ □

If we want to compute generators for the kernel of a matrix $A \in F^{m\times n}$, then, according to Lemma 4.1.1, we may replace A by any row equivalent matrix. It suffices to understand how to determine generators for the kernel of matrices in reduced row echelon form, as it is is easy to work on (See Theorem 4.1.2 by Stoll in [Sto07, Lemma 12.12]). In particular, every matrix can be brought into row echelon form by a sequence of elementary row operations.

Reduced Row Echelon Form

A matrix over a field F is in reduced row echelon form (rref) when it satisfies the following conditions.

- All zero rows, if any, are at the bottom of the matrix.
- Each leading nonzero entry in a row is to the right of the leading nonzero entry in the preceding row.
- Each pivot (leading nonzero entry) is equal to $F(1)$.
- Each pivot is the only nonzero entry in its column.

Theorem 4.1.2. *If $A = (a_{i,j}) \in F^{m\times n}$ is a matrix in reduce row echelon form with r nonzero rows and pivots in the column numbered $j_1 < \ldots < j_r$, then the kernel is generated by the $n - r$ elements*

$$v_k = e_k - \sum_{\substack{1\le i\le r \\ j_i<k}} a_{ik} e_{j_i}, \quad k \in \{1,\ldots,n\} \setminus \{j_1,\ldots,j_r\}$$

where $e_1, \ldots e_n$ is the canonical basis of F^n.

Theorem 4.1.3. *The reduced row echelon form of a matrix is unique ([HK71, p 56]).*

Lemma 4.1.4. *The transformation matrix of rref is not affected by non-pivot columns.*

Proof. Transformation matrix is corresponding to elementary row operations. Since a non-pivot column is dependent on previous columns it does not have a pivot element to apply row operations. □

4.2 Solving Linear Systems

Consider the non-square linear system over a field F: $Ax = b$ where $A \in F^{m\times n}$ and $b \in F^{m\times k}$. Suppose T be a transformation matrix of the reduced row echelon form A_R, so that $TA = A_R$. Now we permute columns of the matrix A and A_R using matrix S to get a nice shape, which is easy to work on. Let r be the rank of matrix A.

$$TAS = A_R S = \left[\begin{array}{c|c} I_r & * \\ \hline O_{m-r\times r} & O_{m-r\times n-r} \end{array}\right] \tag{4.2.1}$$

Here, $[I_r \mid O_{(m-r,r)}]^t$ are pivot columns and $[* \mid O_{(m-r,n-r)}]^t$ are non-pivot columns. We will get zero rows as $[O_{(m-r,r)} \mid O_{(m-r,c-r)}]$ in the bottom, if there are dependent rows in the matrix A.

Now we consider the modified linear system $AS(S^{-1}x) = b$. Suppose $AS = A_s = [A_{Piv} | A_K]$ which is obtained by permuting columns of matrix A. Here columns of A_{Piv} corresponding to pivot columns of A_R and A_K corresponding to non-pivot columns. Let $S^{-1}x = y$ then we have $A_s\ y = b$. For $z = Tb$ we have that:

$$TA_s = T\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right]$$

$$TA_s\ y = T\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right] y = z = Tb \tag{4.2.2}$$

Note: If the system has row dependencies, TA_S is of the form (4.2.1). Then $z = \left[\begin{array}{c} \bar{z} \\ \hline O_{m-r\times k} \end{array}\right]$ for some $\bar{z} \in F^{r\times k}$ and a solution exists iff $Tb = \left[\begin{array}{c} \bar{z} \\ \hline O_{m-r\times k} \end{array}\right]$.

If $n - r > 0$ and $m - r > 0$, then:

$$TA_s\ y = T\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right]\left[\begin{array}{c} \bar{z} \\ \hline O_{n-r\times k} \end{array}\right] = z = \left[\begin{array}{c} \bar{z} \\ \hline O_{m-r\times k} \end{array}\right] = Tb$$

Define a map f: $y = f(z)$ as follows:

$$y = f(z) = \begin{cases} \left[\begin{array}{c} z \\ \hline O_{n-m\times k} \end{array}\right] \text{ add } n-m \text{ zero rows to } z & \text{if } n - m > 0 \\ z & \text{if } m = n \\ \left[\begin{array}{c} \bar{z} \\ \hline O_{n-r\times k} \end{array}\right] \text{ remove } m-n \text{ rows from } z & \text{if } n - m < 0 \end{cases}$$

Since $x = Sy$, we can recover the solution x from $z = Tb$ and map f.

4.2.1 Dixon's Approach

In order to find a solution of non-square linear system $Ax = b$, we solve the linear system $A_s\ y = b$ using p-adic lifting. We compute $\bar{y}$ where $A_s\bar{y} \equiv b \pmod{p^l}$ in l iterations.

- Compute the transformation matrix T_p and rref A_{Rp} of $A \pmod{p}$. Then the permutation matrix S is computed for the matrix A_{Rp} to obtain the nice shape as (4.2.1). At this point the consistency of the system can be checked using modulo p solution.
- Compute a p- adic approximation $\bar{y}$. Take $b_0 = b$

$$z_i \equiv T_p b_i \pmod{p} \quad \text{and} \quad y_i = f(z_i)$$

Algorithm 15 Dixon Solver (`DixonSolver`)

Input: Matrix $A \in F^{m\times n}$, $b \in F^{m\times k}$, a prime number p and a bound B.
Output: Solution matrix X of the linear system $AX = b$.

1: Compute the rref A_{Rp} and transformation matrix T_p of $A \pmod p$.
2: Compute the permutation matrix S for A_{Rp} as (4.2.1).
3: $A_s = AS$
4: $p_1 = 1,\ s = 0$
5: **while** true **do**
6: $\quad z \equiv T_p b \pmod p$
7: $\quad y = f(z)$
8: $\quad s = s + yp_1$
9: $\quad p_1 = p_1 p$
10: $\quad$ **if** $p_1 > B$ **then**
11: $\quad\quad t, Y =$ `VectorReconstruction`(s, p_1)
12: $\quad\quad$ **if** t **and** $A_s Y = b$ **then**
13: $\quad\quad\quad$ **return** $X = SY$
14: $\quad\quad$ **end if**
15: $\quad$ **end if**
16: $\quad b = p^{-1}(b - A_s y)$
17: **end while**

The map f would add or remove zero rows from bottom of z_i such that $\#\,\text{rows}(y_i) = \#\,\text{cols}(A)$.

$$b_{i+1} = p^{-1}(b_i - A_s y_i) \quad \text{for } i = 0, \ldots, l-1.$$

Then $\bar{y} = \sum_{i=0}^{l-1} y_i p^i$.

Theorem 4.2.1. *If the linear system is consistent, Algorithm 15 finds a correct solution.*

Proof. The first phase computes the rref with transformation matrix T_p for $A \pmod p$. From this we have $T_p A_s \equiv A_R S \pmod p$ as (4.2.1). In other word T_p is the same rref-transformation matrix for A_s, due to Lemma 4.1.4.

Phase two computes b_{i+1} vectors. Here we have $(b_i - A_s y_i) \equiv 0 \pmod p$ because: $TA_s\ y_i = TA_{Piv}\bar{z}_i \equiv TA_{Piv}T_p b_i \pmod p \equiv T_p b_i \pmod p$ implies $T_p^{-1}(TA_s y_i - T_p b_i) \equiv 0 \pmod p$. In the last step, it holds:

$$A_s\bar{y} = \sum_{i=0}^{l-1} p^i A_s y_i = \sum_{i=0}^{l-1} p^i (b_i - p b_{i+1}) = b_0 - p^l b_l$$

Now we use a vector reconstruction method to reconstruct the solution y from modular image $\bar{y} \pmod p$. As $A_s\, y = AS(S^{-1}x) = b$, a solution for the linear system $Ax = b$ can be obtained as $x = S\,y$. This permutation matrix S is due to the Theorem 4.1.2. □

4.2.2 Kernel

We present an algorithm to compute a basis for the kernel of the matrix $A \in F^{m\times n}$. One can implement Algorithm 16 as a p-adic modular algorithm. Here we used `DixonSolver` to facilitate the task.

Theorem 4.2.2. *Algorithm 16 is correct.*

Algorithm 16 Kernel (`ModularKernel`)

Input: Matrix $A \in F^{m\times n}$, a prime number p and a bound B.
Output: Kernel matrix X of A.

1: Compute the rref A_{Rp} and transformation matrix T_p of $A \pmod p$.
2: Compute the permutation matrix S for A_{Rp} as (4.2.1).
3: $A_s = AS = [A_{Piv}|A_K]$
4: $z =$ `DixonSolver`$(A_{Piv}, -A_K)$ (Can be continue with Step 3 of Algorithm 15, given transformation matrix T_p and permutation matrix I_m).
5: $y = \left[\dfrac{z}{I_{n-r}}\right]$ where $r = \text{rank}(A_{Rp})$.
6: **return** $n - r, X = Sy$

Proof. Consider the matrix A with transformation matrix T and permutation matrix S as before: $TA = A_R$ with rank r and $AS = A_s = [A_{Piv} \mid A_K]$. Solve the linear system $A_{Piv}x = -A_K$ using Dixon's algorithm: we get a solution as $x = f(-TA_K)$.

$$[A_{Piv}]\,[x] = -\,[A_K] \tag{4.2.3}$$

Then we convert the linear system and the solution back to the original system.

$$\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right]\left[\frac{x}{I_{n-r}}\right] = [O_{n\times n-r}]$$

$$\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right]S^{-1}S\left[\frac{x}{I_{n-r}}\right] = AS\left[\frac{x}{I_{n-r}}\right] = [O_{n\times n-r}]$$

Hence, a basis for the kernel of A is given by $S\left[\dfrac{x}{I_{n-r}}\right]$. □

The role of the permutation matrix can be discribed as follows: Let $v = n - r$

$$\left[\begin{array}{c|c} A_{Piv} & A_K \end{array}\right]\left[\frac{x}{I_v}\right] = [O_{n\times v}]$$

In order to get the original matrix A from $[A_{Piv}|A_K]$ we have to interchange columns using the permutation matrix. Similarly, we should interchange corresponding rows of the solution vector $\left[\dfrac{x}{I_v}\right]$ to get the kernel matrix of A. Let us number the columns of A_{Piv} according to the order of the matrix A as $\{p_1, \ldots, p_r\}$ and non-pivot columns in A_K as $\{k_1, \ldots, k_v\}$. Also, rows in the solution matrix are labelled as follows:

$$\left[\begin{array}{c|c} A_{p_1}\ldots A_{p_r} & A_{k_1}\ldots A_{k_t} \end{array}\right]\left[\begin{array}{c} x_1 \\ \vdots \\ x_r \\ \hline e_1 \\ \vdots \\ e_v \end{array}\right] = [O_{n\times v}]$$

where $x = [x_1, \ldots, x_r]^t$ and $I_v = [e_1, \ldots, e_v]^t$. Suppose that $p_i < k_1 < p_{i+1}$ for the first non-pivot column A_{k_1}. When we shift the column A_{k_1} back to the original place in A, the corresponding row in the solution vector has to be shifted accordingly.

$$\left[\begin{array}{c|c} A_{p_1} \ldots A_{p_i} A_{k_1} A_{p_{i+1}} \ldots A_{p_r} & A_{k_2} \ldots A_{k_v} \end{array}\right] \left[\begin{array}{c} x_1 \\ \vdots \\ x_i \\ e_1 \\ x_{i+1} \\ \vdots \\ x_r \\ \hline e_2 \\ \vdots \\ e_v \end{array}\right] = [O_{m\times v}]$$

We shift columns, until we obtain the matrix A. Then, the resultant solution is the kernel of the matrix A.

Example 4.2.3. *In order to explain this idea, we provide a simple example over $\mathbb{Q}$. Consider the linear system*

$$\begin{aligned} x_1 + 2x_2 + 4x_3 &= 2 \\ 2x_1 + 3x_2 + 7x_3 &= 3 \\ 3x_1 + 3x_2 + 9x_3 &= 3 \end{aligned}$$

Let $A = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 3 & 7 \\ 3 & 3 & 9 \end{bmatrix}$ *and* $b = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}$. *Now we concatenate matrix A and b and compute rref of* $[A|b]$ *to find a solution for the linear system and compute kernel for the matrix A.* $M = \left[\begin{array}{ccc|c} 1 & 2 & 4 & 2 \\ 2 & 3 & 7 & 3 \\ 3 & 3 & 9 & 3 \end{array}\right]$ *The rref of the matrix M is:* $R = \left[\begin{array}{ccc|c} 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right]$

Non-pivot columns in the rref of A provides a basis for the kernel: $\begin{bmatrix} -2 & -1 & 1 \end{bmatrix}^t$. *The last column of M provides a solution for the linear system and it is normalized by the kernel basis:* $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^t$. *In this example, the permutation matrix is the identity matrix.*

4.2.3 Computing Reduced Row Echelon Form

Here we present a lifting method to compute the reduced row echelon form of the matrix $A \in F^{m\times n}$, by extending the kernel computation algorithm. Let y_1 be the solution of the linear system (4.2.2) as:

$$[A_{Piv}]\,[y_1] = -\,[A_K]$$

According to the previous notations, let r be the rank, S be the permutation matrix and T be the transformation matrix of A as given in (4.2.1).

Theorem 4.2.4. *With the above notations,* $\left[\begin{array}{c|c} I_r & -y_1 \\ \hline O_{m-r\times r} & O_{m-r,n-r} \end{array}\right] S^{-1}$ *is the reduced row echelon form of A.*

Proof. We apply the function f in Section 4.2 for the linear system $[A_{Piv}]\,[x] = -[A_K]$. Then, by (4.2.3) we have that:

$$T_1 A_{Piv} S_1 y_1 = -T_1 A_K.$$

Since the matrix A_{Piv} is already full rank the permutation matrix S_1 is the identity matrix I_r. We have that $T_1 = T$ as T is only affected by pivot columns (Lemma 4.1.4). By the definition of f, we have that $-TA_K = f^{-1}(y_1)$. That is $-TA_K = [y_1 \,|\, O_{m-r,n-r}]^t$ if $m - r > 0$ or $-TA_K = y_1$ if $m = r$. Also, it holds that $TA_{Piv} = [I_r \,|\, O_{m-r,r}]^t$ as A_{Piv} is full rank. Therefore we have that:

$$\begin{aligned} TA &= TAS\,S^{-1} \\ &= T\,[A_{Piv} \,|\, A_K]\,S^{-1} \\ &= [TA_{Piv} \,|\, TA_K]\,S^{-1} \\ &= \left[\begin{array}{c|c} I_r & -y_1 \\ \hline O_{m-r\times r} & O_{m-r,n-r} \end{array}\right] S^{-1}. \end{aligned} \tag{4.2.4}$$

□

In order to compute the rref of $A \in F^{m\times n}$, we take the output at Step 4 of Algorithm 16 as $y_1 = \texttt{DixonSolver}(A_{Piv}, -A_K)$. Then, the above theorem gives the rref of A. Here, the block matrix $\left[\begin{array}{c|c} O_{m-r\times r} & O_{m-r,n-r} \end{array}\right]$ at the bottom of (4.2.4) occurs only when $m - r > 0$.

4.2.4 Complexity Analysis and Bounds on Solutions

Consider the linear system $Ax = b$ where $A \in F^{m\times n}$ and $b \in F^{m\times 1}$. Here, we compute complexities, considering a solution of the linear system $A_s y = b$. As explained before, if r is the rank of A, we have $y = \left[\begin{array}{c} \bar{z} \\ \hline O_{n-r} \end{array}\right]$ and $z = Tb = TA_{Piv}\bar{z}$.

Let λ_i denote the Euclidean length of the ith column of A_{Piv} and $|b|$ be the length of b. If A_{Sub} is the maximum rank square sub-matrix of A_{Piv}, we know $\det(A_{Sub}) \leq \prod_{i=1}^{r} \lambda_i$. We can show that the solution we obtain from Algorithm 15 is the same solution we get from maximum rank square sub-system of $Ax = b$. Cramer's rule shows that numerators and denominators of the solution are bounded by absolute value of $\delta = |b| \prod_{i=1}^{r} \lambda_i$.

First we compute the complexity of Algorithm 15 for an integer linear system. Let β be a bound on absolute values of the entries of A_s and b.

i. The first phase computes the transformation matrix of the rref of $A \pmod p$. It takes $O(m^2 n)$ operations between integers of size up to p.

ii. The second step computes a p-adic approximation $\bar{y}$ to y with total of $O(lm^2 \log(m))$ operations: The entries of b_i are integers of length $O(\log(m) + \log\beta) = O(\log(m))$. So, y_i and b_i computation takes $O(m^2 \log(m))$ operations. As we do this iteration l times, construction of $\bar{y}$ takes total of $O(lm^2 \log(m))$ operations with $l \approx m \log(m)$.

iii. The final step to find the solution y from modular result $\bar{y}$ using a vector reconstruction algorithm takes $O(m^2 \log^2(m))$ operations.

Hence the total complexity for integer matrices is $O(m^2n + m^3\log^2(m))$ in $\mathbb{Z}$.

When there is a linear system over number field K of degree d, we can compute a bound for the solution using conjugate matrices (See Section 1.3.8). Using T_2-norm we have: $\delta^2 = \sum_{i=1}^{d} \delta^{(i)^2}$, where $\delta^{(i)} = |b^{(i)}| \prod_{j=1}^{r} \lambda_j{}^{(i)}$. (Here $\lambda_j^{(i)}$ is the length of jth column of the conjugate matrices $A_{Piv}{}^{(i)}$ and $|b^{(i)}|$ is the length of $b^{(i)}$). A bound on coefficients of denominators and numerators of the solution can be obtained by norm change constant (1.3.2) as $c = c_2\delta^2$. Hence by Theorem 1.3.18, we need a bound on prime power as $p^l > \sqrt[d]{N(\mathfrak{a})} > (4c\sqrt{c_1^3 c_2/d}) \cdot (3\sqrt{\gamma(\theta)}/2)^{d-1} = B$ to obtain a correct solution. Let $s_A = d\log(\|A\|_\varphi)$, then we have the bound on determinant $B_d \in O^{\sim}(ns_A)$ such that $B_d \geq d\log(\|\det(A)\|_\varphi)$.

i. Computing rref of $A \pmod p$ takes $O(m^2nM_d(s_A))$ operations.

ii. Dixon solver computes the p-adic expansion of the solution with $O^{\sim}(m^3M_d(s_A))$ bit complexity:

   One lifting iteration contains matrix-matrix multiplications which cost $O^{\sim}(m^2M_d(s_A))$. The number of iteration required to obtain the solution is $O^{\sim}(m)$ (See [Dix82]).

iii. Vector reconstruction to recover the solution takes $O^{\sim}(d^5(B_d/d))$ bit operations for random input matrices (as mentioned in the Theorem 1.3.20, using $\tilde{L}^1$-algorithm). Worst case the bit complexity would be $O^{\sim}(md^5(B/d))$, which is equivalent to $O^{\sim}(m^2d^5(s_A/d))$.

The total complexity of the Algorithm 15 is $O(m^3d + m^2nd + m^2d^5)\log(\|A\|_\varphi)$ bit operations.

We consider the linear system $A_{Piv}x = -A_K$ to compute a basis of the kernel. Similar to the above case we can get a bound on denominators and numerators of each kernel vector using δ, if we replace vector b by columns of A_K. Also, the complexity of phase (i) to compute rref $A \pmod p$ stays the same, but lifting and reconstruction cost would apply for $n - r$ many vectors.

While the new implementation has complexity as above, the direct Gaussian method to obtain the rref takes $O(m^3n^2d^2)\log(\|A\|_\varphi)$ bit operations, because inputs are as large as the size of the matrix $O(mn)$.

4.2.5 Performance Analysis Based on Timing

We have implemented the Algorithm 16 in [Sur21] using Hecke [FHHJ17]. To illustrate the efficiency of the new method, we have computed timings using the number fields $\mathbb{Q}[x]/(x^2 + 7x + 1)$, $\mathbb{Q}[x]/(x^3+7x+1)$ and $\mathbb{Q}[x]/(f)$ for $f = x^5+\sum_{i=0}^{4} 2(-x)^i$, $f = x^{10}+\sum_{i=0}^{9} 2(-x)^i$ and $f = x^{20}+\sum_{i=0}^{19} 2(-x)^i$. Table 4.1 and 4.2 shows timing in seconds (m -for minutes and h- for hours) for the new algorithm `ModularKernel` (`MKer`) for different choice of parameters: d- degree of the number field, size of matrices ($m \times n$) and size of coefficients. Comparisons have been made with the existing implementation in Hecke, which is based on rref without modular methods. Experimental results (with quotient times: Quot) shows that the new algorithm works much better for matrices with large dimensions and large entries in large degree fields, proving the complexity results. Also, the Hecke implementation does not terminate for very large matrices (e.g: for a matrix of dimension 300×300 over a number field of degree 2; $\mathbb{Q}[x]/(x^2 + 7x + 1)$).

d	Input size		-100:100			-1000:1000			-1000000 :1000000		
	m	n	MKer	Hecke	Quot	MKer	Hecke	Quot	MKer	Hecke	Quot
2	100	90	0.26	1.2 m	281	0.29	2 m	419	0.83	5.5 m	395
		100	0.30	1.7 m	332	0.30	3 m	584	0.33	7.7 m	1404
		110	3.45	2.2 m	38.6	4.04	3.8 m	56.3	6.95	10.1 m	87.5
		120	6.53	2.8 m	25.9	7.87	4.8 m	36.2	13.69	12.6 m	55.1
		130	10.54	3.4 m	19	12.47	5.7 m	27.5	21.93	15 m	41.3
		140	14.96	4 m	15.7	18.17	6.6 m	21.9	30.85	17.6 m	34.3
	150	140	0.84	15.6 m	1117	1.15	28 m	1461	0.70	53.3 m	4551
		150	0.91	19.6 m	1281	0.90	33.5 m	2233	1.08	1.5 h	4991
		160	10.29	23.5 m	137	12.55	39.5 m	189	21.12	1.7 h	295
		170	19.42	27.8 m	85.9	23.89	46.5 m	117	41.72	2 h	174
		180	30.37	31.8 m	62.8	36.84	53.4 m	87	62.6	2.3 h	133
		190	43.31	27.2 m	51.5	51.79	1 h	69.9	89.21	2.6 h	105
	300	280	6.35	-	∞	6.17	-	∞	6.39	-	∞
		300	7.06	21.9 h	11163	6.97	-	∞	7.49	-	∞
3	10	11	0.01	0.00	0.75	0.01	0.01	0.82	0.01	0.01	0.83
		15	0.02	0.01	0.40	0.03	0.01	0.39	0.05	0.02	0.43
		30	0.08	0.02	0.28	0.11	0.03	0.30	0.20	0.06	0.31
		60	0.26	0.06	0.22	0.33	0.14	0.42	0.55	0.14	0.25
		70	0.31	0.07	0.21	0.40	0.09	0.22	0.69	0.16	0.23
		90	0.47	0.08	0.18	0.58	0.11	0.19	2.55	0.22	0.08
	50	51	0.31	4.57	14.89	0.40	7.23	17.90	0.72	18.13	25.18
		55	1.25	5.82	4.67	1.69	9.04	5.36	3.23	22.90	7.10
		70	4.90	10.20	2.08	6.74	17.04	2.53	12.73	40.51	3.18
		100	12.87	20.24	1.57	17.34	31.66	1.83	34.86	1.2 m	2.14
		110	15.25	21.98	1.44	22.27	35.16	1.58	40.62	1.5 m	2.17
		130	20.86	29.42	1.41	28.28	45.94	1.62	59.36	1.8 m	1.83
	80	81	1.24	1.1 m	52.33	1.64	1.8 m	64.37	3.13	4.4 m	83.91
		85	5.25	1.3 m	14.32	7.15	2 m	17.27	13.57	5 m	22.31
		100	20.37	1.9 m	5.65	27.74	3.1 m	6.70	53.15	7.6 m	8.57
		130	51.80	3.3 m	3.78	1.2 m	5.2 m	4.45	2.3 m	12.7 m	5.62
		140	61.29	3.4 m	3.62	1.4 m	6 m	4.30	2.8 m	14.4 m	5.16
		160	87.26	4.6 m	3.16	2 m	7.3 m	3.66	3.7 m	17.7 m	4.77
	100	101	2.43	4.1 m	100.5	3.20	6.5 m	122.86	5.94	15.8 m	159.9
		105	10.01	4.6 m	27.40	13.92	7.3 m	31.64	26.76	18.9 m	42.4

Table 4.1: Timing for Kernel Computation.

d	Input size		-100:100			-1000:1000			-1000000 :1000000		
	m	n	`MKer`	`Hecke`	Quot	`MKer`	`Hecke`	Quot	`MKer`	`Hecke`	Quot
5	10	11	0.01	0.01	0.83	0.11	0.01	0.14	0.04	0.03	0.73
		15	0.05	0.02	0.36	0.06	0.02	0.39	0.11	0.05	0.46
		30	0.18	0.05	0.25	0.25	0.06	0.26	0.63	0.12	0.19
		60	0.71	0.11	0.15	0.91	0.14	0.15	1.47	0.26	0.18
		70	0.84	0.11	0.13	1.28	0.17	0.13	1.96	0.31	0.16
		90	1.32	0.18	0.13	1.63	0.21	0.13	2.83	0.40	0.14
	50	51	0.92	11.39	12.34	1.06	18.37	17.37	2.07	45.11	21.82
		55	3.71	14.33	3.87	4.79	23.00	4.80	9.25	56.53	6.11
		70	14.36	25.55	1.78	18.77	40.86	2.18	35.73	1.7 m	2.78
		100	37.24	47.34	1.27	48.49	1.3 m	1.57	1.5 m	3.1 m	1.99
	80	81	3.25	2.8 m	51.93	4.27	4.5 m	62.86	7.52	10.6 m	84.86
		85	14.09	3.3 m	14.01	18.56	5.2 m	16.68	35.95	12.2 m	20.38
		100	57.13	4.9 m	5.17	1.2 m	7.7 m	6.10	2.4 m	18.8 m	7.94
		130	4.6 m	8.7 m	1.91	3.2 m	13 m	4.04	8.2 m	34.6 m	4.23
	100	101	2.6 m	16.4 m	6.30	45.37	17.4 m	22.96	24.65	41.1 m	100.1
		105	7.5 m	21.2 m	2.82	2.7 m	19.6 m	7.23	2.8 m	52.8 m	18.66
		120	27.7 m	38.7 m	1.40	11.5 m	33.8 m	2.95	27.5 m	1.5 h	3.32
10	10	11	0.15	0.08	0.53	0.11	0.12	1.11	0.10	0.31	3.10
		15	0.18	0.10	0.57	0.25	0.15	0.61	0.47	0.42	0.89
		30	0.92	0.20	0.22	1.08	0.29	0.27	1.87	0.70	0.38
		60	2.13	0.41	0.19	2.81	0.57	0.20	4.88	1.49	0.31
	50	30	0.12	8.46	67.71	0.13	13.16	104.2	0.13	31.94	248
		45	0.20	35.16	175.3	0.19	55.68	287.7	0.19	2.3 m	710
		51	3.00	51.84	17.27	4.06	1.4 m	19.97	7.60	3.2 m	25.42
		55	13.29	1.1 m	4.85	18.16	1.7 m	5.55	35.08	4 m	6.76
		70	59.11	1.9 m	1.96	1.2 m	3.1 m	2.44	2.3 m	6.7 m	2.96
		100	2.3 m	3.4 m	1.46	3.1 m	5.5 m	1.74	5.9 m	12.4 m	2.11
	80	75	0.72	9 m	747.5	0.57	13.8 m	1469	0.58	31.9 m	3306
		81	5.3 m	15 m	2.80	18.84	17.7 m	56.34	34.48	40.3 m	70.22
		85	55.12	13.1 m	14.29	1.2 m	20.1 m	16.36	2.3 m	46.2 m	19.68
		100	3.7 m	21 m	5.65	9.7 m	30.2 m	3.10	10 m	1.3 h	7.78
	100	95	1.04	33.1 m	1904	1.00	50.8 m	3042	1.01	1.9 h	6860
		101	22.19	40 m	108.2	29.62	1 h	123.7	58.84	2.3 h	141.8
		105	1.9 m	44.7 m	24.09	2.3 m	1.2 h	29.5	4.9 m	2.7 h	32.86
		120	29.1 m	1.3 h	2.58	13.2 m	1.8 h	8.23	41.4 m	5 h	7.19
20											
	10	11	0.19	1.12	5.76	0.25	1.77	7.15	0.54	5.04	9.41
		15	1.13	1.22	1.08	1.19	2.38	2.01	2.30	5.02	2.18
		30	3.46	1.55	0.45	4.71	2.72	0.58	9.55	6.28	0.66
	50	30	0.31	57.91	185.9	0.31	1.6 m	302.4	0.31	3.9 m	766.1
		45	0.46	3.8 m	500.1	0.46	6 m	789.2	0.44	15 m	2049
		51	16.19	5.5 m	20.28	21.29	8.6 m	24.37	43.01	21.3 m	29.74
		55	1.2 m	6.4 m	5.20	1.6 m	9.9 m	6.10	3.2 m	27.8 m	8.71

Table 4.2: Timing for Kernel Computation.

Chapter 5

Computing Characteristic and Minimal Polynomials

Computing the characteristic polynomial of a matrix is a classical and fundamental problem in mathematics. This is closely related to other problems such as computing the minimal polynomial and computing the Frobenius canonical form. There are many optimized algorithms to compute characteristic polynomials and minimal polynomials for integer matrices and matrices over polynomial rings. The goal of this chapter is to address these problems over number fields.

This chapter begins with a survey of algorithms for characteristic polynomial computation, with their complexity improvements. Then we compute bounds for coefficients of the minimal polynomial and characteristic polynomial using bounds for complex matrices. We use modular methods to optimize the existing algorithms. Finally, we present an additional improvement to the minimal polynomial implementation, that reduces the runtime.

5.1 Literature Review

5.1.1 Algorithms for Computing the Characteristic Polynomial

Let A be an $n \times n$ matrix over any field F. If $Au = \lambda u$, for some $\lambda \in F$ and $u \in F^{n\times 1}$, then λ and u are called the *eigenvalue* and *eigenvector* of A, respectively. The eigenvalues of A are the roots of the *characteristic polynomial* $C_A(\lambda) = \det(\lambda I_n - A)$. The eigenvectors are the solutions to the *homogeneous system* $(\lambda I_n - A)x = 0$.

Theorem 5.1.1 (Cayley-Hamilton Theorem)**.** *If $C_A(\lambda) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n$ is the characteristic polynomial of the $n \times n$ matrix A, then $C_A(A) = A^n + c_1A^{n-1} + \cdots + c_{n-1}A + c_n = O_{n\times n}$.*

The Method of Direct Expansion

Let the characteristic polynomial of $A = (a_{i,j}) \in F^{n\times n}$ be $C_A(\lambda) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n$. In [Gan60, Section III.7] Gantmacher shows that for $i = 1, \ldots, n$ the coefficients c_i of $C_A(\lambda)$ are the alternate sum of all the $(n-i)\times(n-i)$ diagonal minors of A. Therefore the coefficients can be computed as follows:

$c_1 = -\sum_{i=1}^{n} a_{i,i} = -\operatorname{trace}(A)$, which is the sum of all first-order diagonal minors of A.

$c_2 = \sum_{i<j} \det\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ the sum of all second-order diagonal minors of A. Similarly, the coefficients $c_3, c_4, \ldots, c_{n-1}$ can be obtained from the minor sums of the matrix A and, finally $c_n = (-1)^n \det(A)$.

In order to compute the coefficient c_i, we need to compute determinants of $\binom{n}{i}$ diagonal minors of A. Hence, the direct computation of the coefficients of $C_A(\lambda)$ is equivalent to computing $\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{n} = 2^n - 1$ determinants of various orders of minor matrices. Since this computation has exponential complexity it is not practical. Instead a wealth of polynomial-time methods for computing the characteristic polynomial of matrices over integers (or any field F) can be found in the literature. In this section, we shall explain some of these methods.

Leverrier's Algorithm

Theorem 5.1.2 (Newton's Identity)**.** *Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the roots of the polynomial $C(\lambda) = \lambda^n + c_1\lambda^{n-1}+\cdots+c_{n-1}\lambda+c_n$. If $s_k = \lambda_1^k+\lambda_2^k+\cdots+\lambda_n^k$, then $c_k = -(s_k+s_{k-1}c_1+s_{k-2}c_2+\cdots+s_2c_{k-2}+s_1c_{k-1})/k$.*

In 1840, Leverrier introduced a method to find the characteristic polynomial of any matrix $A \in F^{n\times n}$ using the trace of the powers A^k, where $k = 1, 2, \ldots, n$. His algorithm make the use of the Newton's identities (See [Lev40]). Later Souriau, Faddeev, Frame and Csanky improved Leverrier's Algorithm in [FF63], based on Matrix multiplications with the complexity of $O(n^4)$.

Let $\sigma(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, be the set of all eigenvalues of A which is also called the spectrum of A. Then Leverrier uses the fact that $s_k = \text{trace}(A^k) = \sum_{i=1}^n \lambda_i^k$, for all $k = 1, 2, \ldots, n$. The method of Souriau (or Faddeev and Frame) in [FF63] use the recurrence relation $A_i = AB_{i-1}$, $c_i = -\text{trace}(A_i)$, $B_i = A_i + c_iI_n$ with $B_0 = I_n$; to compute coefficients c_i of C_A for $i = 1, 2, \ldots, n$.

The Method of Danilevskii

Definition 5.1.3. *Consider an $n \times n$ matrix A and let $C_A(\lambda) = \lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n$ be its characteristic polynomial. Then the companion matrix of $C_A(\lambda)$:*

$$\mathcal{F}[A] = \begin{pmatrix} -c_1 & -c_2 & -c_3 & \cdots & -c_{n-1} & -c_n \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

is similar to A is called the Frobenius form of A.

The method of Danilevskii in [Dan37] applies the Gauss-Jordan method to obtain the Frobenius form of an $n \times n$ matrix. The algorithm successively transforms the matrix A to $\mathcal{F}[A]$, by applying $n - 1$ similarity transformations as explained bellow:

$$\text{Let } A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

We begin the algorithm by computing transformation matrices to carry the n-th row of A into the row $\begin{pmatrix} 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$.

Assuming that $a_{n,n-1} \neq 0$, we take the transformation matrix U_{n-1} and its inverse matrix V_{n-1} as:

$$U_{n-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-1} & a_{n,n} \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}, \text{ and}$$

$$V_{n-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ v_{n-1,1} & v_{n-1,2} & v_{n-1,3} & \cdots & v_{n-1,n-1} & v_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

where $v_{n-1,i} = -a_{n,i}/a_{n,n-1}$ for $i \neq n-1$ and $v_{n-1,n-1} = 1/a_{n,n-1}$.

Multiplying the matrix A by V_{n-1} and U_{n-1}, we obtain the matrix B as $B = U_{n-1}AV_{n-1}$. Then B is similar to A and is of the form

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,n-1} & b_{1,n} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,n-1} & b_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{n-1,1} & b_{n-1,2} & b_{n-1,3} & \cdots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

Now, if $b_{n-2,n-1} \neq 0$, then similar operations can be performed on matrix B, and continue in the same way. Finally, we obtain the Frobenius form $\mathcal{F}[A] = U_1U_2\cdots U_{n-1}AV_{n-1}V_{n-2}\cdots V_1$. In some cases, these $n-1$ intermediate transformations are not possible. For example: In the first step if $a_{n,n-1} = 0$, we apply unimodular transformations to obtain a similar matrix with $a_{n,n-1} \neq 0$. If it was found that $b_{k,k-1} = 0$ in some intermediate step, as k-th row being the last non-zero row of the matrix $B = U_kU_{k+1}\cdots U_{n-1}AV_{n-1}V_{n-2}\cdots V_k$, then we apply apply unimodular transformations to make $b_{k,k-1} \neq 0$ and, continue the algorithm as a separate block matrix considering the sub-matrix $B_{[1:k-1,1:k-1]}$. Danilevskiy method finds the characteristic polynomial, by computing $\mathcal{F}[A]$ with the complexity of $O(n^3)$.

Hessenberg's Algorithm

A square matrix $M = (m_{i,j}) \in F^{n\times n}$ is in upper Hessenberg form if $m_{i,j} = 0$ for all $i \geq j+2$, in other words, entries below the first subdiagonal are zero.

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & \cdots & m_{1,n-2} & m_{1,n-1} & m_{1,n} \\ m_{2,1} & m_{2,2} & m_{2,3} & \cdots & m_{2,n-2} & m_{2,n-1} & m_{2,n} \\ 0 & m_{3,2} & m_{3,3} & \cdots & m_{3,n-2} & m_{3,n-1} & m_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & m_{n-1,n-2} & m_{n-1,n-1} & m_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & m_{n,n-1} & m_{n,n} \end{pmatrix}$$

Any given matrix $A \in F^{n\times n}$ can be reduced in to the Hessenberg form using a series of elementary row and column operations, over F with complexity $O(n^3)$. If M is the Hessenberg form of A, then it holds $\det(\lambda I_n - A) = \det(\lambda I_n - M)$, and the characteristic polynomial $C_A(\lambda) = C_M(\lambda) = P_{n+1}(\lambda) \in F[\lambda]$ of the matrix M in Hessenberg form can be computed from the following recurrence for $P_k(\lambda)$.

$$P_{k+1}(\lambda) = \begin{cases} 1 & \text{if } k = 0 \\ (\lambda - m_{k,k})P_k(\lambda) - \sum_{i=1}^{k-1}\left(\prod_{j=i}^{k-1} m_{j+1,j}\right)m_{i,k}P_i(\lambda) & \text{if } 1 \leq k \leq n+1 \end{cases}$$

This can be efficiently computed using $O(n^2)$ operations in $F[\lambda]$ ($O(n^3)$ operations in F). The total complexity for transformation into quasi-upper triangular matrix and iteration of the determinant expansion formula is approximately $O(n^3)$ over F. The original algorithm can be found in Hessenberg's thesis [Hes42] and the modular method has been explained in [LMW05].

5.1.2 The Berkowitz Method

The Berkowitz algorithm computes the characteristic polynomial over a commutative ring R using $O(n^4)$ arithmetic operations over the ring. Consider the matrix $A \in R^{n\times n}$. Let $A_r = [a_{ij}]$, $1 \le i, j \le r$ be the principal $r \times r$ submatrix of A. The characteristic polynomial of A_r is

$$C_r = \det(A_r - \lambda I_r) = \sum_{k=0}^{r} c_{r,r-k}\lambda^k = c_{r,r} + c_{r,r-1}\lambda + \cdots + c_{r,0}\lambda^r.$$

The adjoint of the characteristic matrix satisfies

$$\begin{aligned}\operatorname{Adj}(A_r - \lambda I_r) &= \sum_{i=0}^{r-1}\sum_{j=0}^{r-i-1} c_{r,r-i-j-1}A_r^j\lambda^i = -\sum_{k=1}^{r}\sum_{j=0}^{r-k} c_{r,r-k-j}A_r^j\lambda^{k-1} \\ &= -\sum_{k=1}^{r}\left(c_{r,r-k}I_r + c_{r,r-k-1}A_r + c_{r,r-k-2}A_r^2 + \cdots + c_{r,0}A_r^{r-k}\right)\lambda^{k-1} \qquad (5.1.1)\end{aligned}$$

Consider the matrix A_{r+1}, which is written in terms of A_r, the diagonal term $a_{r+1,r+1}$, row vector R_r and column vector S_r. Then

$$\det(A_{r+1}) = \det\begin{pmatrix} A_r & S_r \\ R_r & a_{r+1,r+1}\end{pmatrix} = \det(A_r)a_{r+1,r+1} - R_r\operatorname{Adj}(A_r)S_r. \qquad (5.1.2)$$

Combining (5.1.1) and (5.1.2) on the characteristic matrix gives a recurrence formula for $C_{r+1}(\lambda)$ in terms of $C_r(\lambda)$

$C_{r+1}(\lambda) = C_r(\lambda)(a_{r+1,r+1} - \lambda) + \sum_{k=1}^{r}\sum_{j=0}^{r-k} c_{r,r-k-j}\left(R_rA_r^jS_r\right)\lambda^{k-1}$.

The Berkowitz algorithm uses matrix and vector multiplications. Given a polynomial $C_r(\lambda) = \sum_{k=0}^{r} c_{r,k}\lambda^{r-k}$, let $\vec{C}_r = (c_{r,0}, c_{r,1}, \ldots, c_{r,r})^t$.

For $C_r(\lambda) = \sum_{k=0}^{r} c_{r,k}\lambda^{r-k}$, the $(r+1)\times r$ lower triangular Toeplitz matrix is defined as

$$\operatorname{Toep}(C_r) = \begin{bmatrix} c_{r,0} & 0 & 0 & \cdots & 0 \\ c_{r,1} & c_{r,0} & 0 & \cdots & 0 \\ c_{r,2} & c_{r,1} & c_{r,0} & \cdots & 0 \\ \cdots & \cdots & \cdots & \ddots & 0 \\ c_{r,r-1} & c_{r,r-2} & c_{r,r-3} & \cdots & c_{r,0} \\ c_{r,r} & c_{r,r-1} & c_{r,r-2} & \cdots & c_{r,1}\end{bmatrix}.$$

The corresponding recurrence formula for $\vec{C}_{r+1}$ is

$$\vec{C}_{r+1} = \operatorname{Toep}(Q_{r+1}) \times \vec{C}_r, \quad \text{where}$$

$Q_{r+1} = -\lambda^{r+1} + a_{r+1,r+1}\lambda^r + (R_rS_r)\lambda^{r-1} + \cdots + (R_rA_r^iS_r)\lambda^{r-1-i} + \cdots + (R_rA_r^{r-1}S_r)$.

Hence the characteristic polynomial is given by

$\vec{C}_n = \operatorname{Toep}(Q_n) \times \cdots \times \operatorname{Toep}(Q_3) \times \operatorname{Toep}(Q_2) \times \vec{C}_1$.

The Algorithm 17 explains the steps of the Berkowitz method to compute the characteristeristic polynomial.

Algorithm 17 Characteristic Polynomial - Berkowitz Method (`CharPolyBerkowitz`)

Input: $A = (a_{i,j}) \in R^{n\times n}$.
Output: Characteristic polynomial of A (coefficients in vector form).

1: $C = \begin{pmatrix} -1 \\ a_{11} \end{pmatrix}$.
2: **for** $i = 1, 2, 3, \ldots, n-1$ **do**
3: Obtain row vector R_i, column vector S_i and principal matrix A_i.
4: $Q = -\lambda^{i+1} + a_{i+1,i+1}\lambda^i + \sum_{j=1}^{i} R_i A_i^{j-1} S_i \lambda^{i-j}$
5: $C = \text{Toep}(Q) \times C$
6: **end for**
7: **return** C

Nemo/Hecke Algorithm

In Hecke [FHHJ17, Section 4.2], characteristic polynomials are computed using the generic implementation of division free Berkowitz algorithm in [Ber84]. Usually, fraction free algorithms can improve performance of matrix algorithms over integral domains. The algorithm in Hecke is developed by William Hart and, it is the fastest implemented algorithm for characteristic polynomial computation.

The State of the Art Method

The breakthrough by Keller-Gehrig uses Krylov bases (Section 5.1.3) to compute characteristic polynomial with the complexity of $O(n^\omega \log(n))$ (See [KG85, Section 3]). This came after the Strassen matrix multiplication was introduced in [SS71] with the complexity (n^ω). His second algorithm uses Danilevskii's algorithm with block matrix operations. Even though Algorithm [KG85, 6.1] has complexity of $O(n^\omega)$, it is only valid for generic matrices. Inspired by Gehrig's method in [PS07] Pernet and Storjohann presented a new Las Vegas randomized algorithm for computing the characteristic polynomial of a dense matrix over large fields ($\#F > 2n^2$) in $O(n^\omega)$ field operations. This fast and practical implementation uses a shifted Krylov extension and Hessenberg form in block matrices. The latest deterministic algorithm for computing the characteristic polynomial has been introduced by Pernet and Neiger in [NP20]. It computes the characteristic polynomial of a matrix over a field within the same asymptotic complexity, up to constant factors, as the matrix multiplication.

5.1.3 Algorithms for Computing the Minimal Polynomial

The Method of Krylov/ Spinning

Krylov method, also known as "spinning" in [Ste97] can be used to compute the minimal polynomial of a matrix $A \in F^{n\times n}$. Furthermore, we can extend this algorithm to compute the characteristic polynomial as we will explaine in Section 5.3.2.

The matrix A can be considered as a representation matrix of some linear operator on an n-dimensional vector space V in some basis. For any polynomial $f(x) = a_0x^m + a_1x^{m-1} + \cdots + a_{m-1}x + a_m \in F[x]$, we can define its value at A as $f(A) = a_0A^m + a_1A^{m-1} + \cdots + a_{m-1}A + a_mI_n$. There exists a polynomial f such that $f(A) = 0$. Such a polynomial is called an *annihilating polynomial* of A, and the one with the least degree is the *minimal polynomial* of A, and is denoted by M_A.

We use the following theorem for computing and verifying the minimal polynomial in our applications (See [Vin03, Section 6.5]).

Theorem 5.1.4. *Suppose A is a linear operator on a F-vector space V, and that $V = W_1 + W_2 + \cdots + W_r$ for invariant subspaces W_i. Then the minimal polynomial of A is the* $\mathrm{LCM}(M_{W_1}, \ldots, M_{W_r})$*, where M_{W_i} is the minimal polynomial of A restricted to W_i.*

Definition 5.1.5. *Given a matrix $A \in F^{n\times n}$ and an n-dimensional nonzero column vector v, we define the Krylov sequence associated to A and v as the sequence of vectors $v_i = A^i v$ $(i = 0, 1, 2, \ldots)$.*

Note that at most n vectors of the sequence $v_0, v_1, \ldots$ are linearly independent.

Definition 5.1.6. *Given a vector v in a vector space V, the linear subspace spanned by the Krylov sequence associated to A and v is defined as the Krylov subspace $K_A(V, v)$ associated to v.*

Theorem 5.1.7. *Let the matrix A, v and the Krylov sequence v_i for $i = 0, 1, 2, \ldots$. Suppose for some $r\ (\leq n)$, the vectors $v_0, v_1, \ldots, v_{r-1}$ are linearly independent and it holds that $a_r v_0 + a_{r-1} v_1 + \cdots + a_1 v_{r-1} + v_r = O_n$ where $a_i \in F$. If $W = K_A(V, v)$ then the monic polynomial $f(\lambda) = a_r + a_{r-1}\lambda + \cdots + a_1\lambda^{r-1} + \lambda^r$ is the minimal polynomial M_W (the minimal polynomial of A restricted to W).*

Proof. W is an invariant subspace, and we have that $a_r v_0 + a_{r-1} v_1 + \cdots + a_1 v_{r-1} + v_r = (c_0 I_n + c_1 A + \cdots + c_{r-1} A^{r-1} + A^r)v = O_n$. Hence we have that $f(A)(v) = 0$. It follows $f(A)(g(A)v) = 0$ for all polynomials $g(\lambda) \in F[\lambda]$ and so $f(A)$ annihilates all of W. Since $r \leq n$ is the minimum such that $v_0, v_1, \ldots, v_{r-1}$ are linearly independent, $f(\lambda)$ is the minimum polynomial M_W. □

Algorithm 18 computes the minimal polynomial of matrix A. The correctness of the algorithm follows from the Theorem 5.1.4. The Step 3 computes the minimal polynomial M_W of A restricted to $W = K_A(V, v)$. This can be computed as follows:

Consider the matrix $V_i = [v, Av, A^2v, \ldots, A^{i-2}v, A^{i-1}v]$ for some $2 \leq i \leq n$. We observed the least i (say $i = r$) such that the linear system $V_i a = -A^i v$ is consistent. Then the unique solution $a^t = [a_r, a_{r-1}, \ldots, a_1]$ gives M_W as $f(\lambda) = a_r + a_{r-1}\lambda + \cdots + a_1\lambda^{r-1} + \lambda^r$.

Algorithm 18 Minimal Polynomial (`MinPoly`)

Input: $A \in F^{n\times n}$.
Output: Minimal polynomial of A.

1: $v = e_1 = (1, 0, \ldots, 0)$
2: $W = K_A(V, v)$
3: Compute M_W
4: **while** $\mathrm{rank}(W) = n$ **do**
5: Pick $v \in V$ such that v is linearly independent of W.
6: $W' = K_A(V, v)$
7: Compute $M_{W'}$
8: $W = W + W'$
9: $M_W = \mathrm{LCM}(M_W, M_{W'})$
10: **end while**
11: $M_A = M_W$
12: **return** M_A

As explained in [FHHJ17, Section 4.2], Hecke uses a CRT approach to compute the minimal polynomial over $\mathbb{Z}$, using the Algorithm 18. In the worst case it requires $O(n^4)$ operations over $\mathbb{Z}$.

5.2 Bounds on the Coefficients of Characteristic Polynomial and Minimal Polynomial

In practice, when we use the above algorithms, first the polynomials are computed modulo several prime numbers and then only reconstructed via CRT (See [GG13, Theorem 10.25]). Therefore, we need precise bounds on the coefficients of characteristic and minimal polynomials to find the required number of primes. Here, we discuss some estimates for both two cases, as Dumas has presented in [Dum06] for matrices over $\mathbb{C}$. After that, we extend his idea to get bounds on coefficients of characteristic and minimal polynomials of matrices over number fields.

Here we denote the characteristic polynomial of A by $C_A(\lambda) = c_0\lambda^n + c_1\lambda^{n-1} + \cdots + c_{n-1}\lambda + c_n$ and the minimal polynomial of A by $M_A(\lambda) = m_0\lambda^r + m_1\lambda^{r-1} + \cdots + m_{r-1}\lambda + m_r$ (for a matrix A over $\mathbb{C}$ or K accordingly).

Definition 5.2.1. *Let $P = a_t x^t + a_{t-1}x^{t-1} + \cdots + a_0$ be a univariate polynomial over $\mathbb{C}$. The 2-norm of P is defined by*

$$\|P\|_2 = \Big(\sum_{i=0}^{t} |a_i|^2\Big)^{1/2}.$$

Here, $|a|$ denotes the absolute value of a. The 1-norm is defined by

$$\|P\|_1 = \sum_{i=0}^{t} |a_i|$$

and the ∞-norm by

$$\|P\|_\infty = \max_{0\le i\le t}(|a_i|).$$

We define the ∞-norm on a polynomial $P = a_t x^t + a_{t-1}x^{t-1} + \cdots + a_0$ over K, with respect to the T_2-norm as $\|P\|_\infty = \max\{T_2(a_j) \mid j = 0, \ldots, t\}$.

In order to get bounds on coefficients of $C_A(\lambda)$ for $A = [a_{i,j}] \in \mathbb{C}^{n\times n}$, we take the use of Hadamard's bound on the determinants of the minor matrices. Let $B \in \mathbb{R}$ such that $B \ge \max_{i,j}\{|a_{i,j}|\}$. Using Hadamard bound on A, we have that $|\det(A)| \le n^{n/2}B^n$ from [GG13, Theorem 16.6].

5.2.1 Bounds on Coefficients of the Characteristic Polynomial

Now we can use the above mentioned bounds on determinants to get bounds on coefficients of $C_A(\lambda)$. As it is mentioned in the Section 5.1.1, c_i, the i-th coefficient of the characteristic polynomial of $A \in \mathbb{C}^{n\times n}$, is the sum of all the $(n-i)\times(n-i)$ diagonal minors of A with the sign $(-1)^i$. Therefore c_i is bounded by $\binom{n}{i}\sqrt{(n-i)B^2}^{(n-i)}$, where B is the upper bound for absolute value of coefficients of A. Using this fact, the following lemma has been derived by Dumas in [Dum06, Lemma 2.1].

Lemma 5.2.2. *Let $A \in \mathbb{C}^{n\times n}$, with $n \ge 4$, $\|A\|_\infty \le B$. The coefficients of the characteristic polynomial $C_A(\lambda)$ of A are denoted by c_j, $j = 0, \ldots, n$ and $\|C_A\|_\infty = \max\{|c_j|\}$. Then*

$$\log_2(\|C_A\|_\infty) \le n(\log_2(n) + \log_2(B^2) + 0.21163175)/2$$

Dumas considers the function $F(n, j) = \binom{n}{j}\sqrt{(n-j)B^2}^{(n-j)}$, in the proof of [Dum06, Lemma 2.1]. From the symmetry of the binomial coefficients, it is only necessary to explore the $\lfloor n/2\rfloor$ to obtain the maximum of $F(n, j)$. The lemma is proven inductively. For $j = 0$ it is true by Hadamard's bound. For $j = 1$, $n \ge 2$, set $f(n) = \frac{2}{n}\Big(\log_2(F(n,1)) - \frac{n}{2}\log_2(n) - (n-1)\log_2(B)\Big)$. Computing the first derivative

f' of f, we have that f' is independent of B and maximum value of $f(n)$ is $5\log_2(5)/6 - 2\log_2(6)/3 < 0.21163175$. For $j > 1$, Stirling formula on binomial coefficient has been used to get a bound on the maximum value of $F(n, j)$ using a similar approach as before.

Dumas has presented a more precise estimate in [Dum06, Lemma 2.2] (here Lemma 5.2.3), by locating the largest coefficient of the characteristic polynomial. Usually, the largest coefficient of the characteristic polynomial can be found in the $O(\sqrt{n})$ last ones (which follows from the proof of [Dum06, Lemma 2.1]).

Lemma 5.2.3. *Let $A \in \mathbb{C}^{n\times n}$, with $n \geq 4$. Let the entries are bounded in absolute values by $B > 1$. Then*

$$\|C_A\|_\infty \leq \max_{i=0,\ldots,D} \binom{n}{i} ((n-i)B^2)^{(n-i)/2}$$

where $D = (-1 + \sqrt{1 + 2\delta n B^2}))/(\delta B^2)$, and $\delta \approx 5.418236$.

Lemma 5.2.4 gives a bound on the coefficients of characteristic polynomial over K, which immediately follows from the definition of T_2-norm and properties of conjugation.

Lemma 5.2.4. *Let $A = [a_{ij}] \in K^{n\times n}$, with $n \geq 4$, then the characteristic polynomial C_A satisfies*

$$\|C_A\|_\infty \leq \sum_{k=1}^{d} \|C_A^{(k)}\|_\infty^2$$

where $A^{(k)}$ for $k = 1, \ldots, d$ are the matrices with conjugate entries as $A^{(k)} = [a_{ij}^{(k)}]$.

5.2.2 Bounds for the Coefficients of the Minimal Polynomial.

In this section, we derive bounds for the coefficients of minimal polynomial for matrices over $\mathbb{C}$ and K. When the matrix $A \in \mathbb{C}^{n\times n}$ we have bounds for coefficients, in terms of the bounds for roots of the M_A (in other words eigenvalues of the matrix A). Also, given a bound on the coefficients of C_A, we provide an easy way to get bounds on M_A.

The Size of the Factors of a Polynomial

Here, we provide some relations between sizes of coefficients and factors of a polynomial in general (see [Mig92, Section 4.3 & 4.4] for more details). First we define a measure for any given polynomial over $\mathbb{C}$.

Definition 5.2.5. *Let P be a polynomial with complex coefficients. P can be written in terms of its roots z_i as*

$$P = a_r x^r + a_{r-1}x^{r-1} + \cdots + a_0 = a_r(x - z_1)\cdots(x - z_r),$$

with $a_r \neq 0$. The Mahler measure of P is defined by the formula

$$M(P) = |a_r| \prod_{j=1}^{r} \max\{1, |z_j|\}.$$

The next result from [Mig92, Definition 4.4.1] is important in our application.

Let $z_1, \ldots, z_r$ designate the roots of the polynomial P as above. Then, the well known Newton's identities gives the relation between coefficients and roots of a polynomial. If the roots are counted with their multiplicities, the formula $P = a_r(x - z_1)\cdots(x - z_r)$ gives elementary symmetric polynomials:

$$\frac{a_{r-j}}{a_r} = (-1)^{r-j} \sum_{i_1<\ldots<i_j} z_{i_1} \cdots z_{i_j}$$

(sum of products, where each product consist of $r - j$ factors and each of these factors being a root of P). Each of these factors is bounded by $M(P)$. Moreover there are $\binom{r}{j}$ of these factors for $j = 1, 2, \ldots, r$. Hence the inequality (5.2.1) holds.

$$\|a_j\| \leq \binom{d}{j} M(P). \tag{5.2.1}$$

This with the *Landau inequality* ([Lan05]) given rise to the following theorem, which can be used to get bounds on M_A and Q_A in terms of C_A.

Theorem 5.2.6. *Let $f = \sum_{i=0}^{n} f_i x^i$ and $h = \sum_{i=0}^{m} h_i x^i$ be two polynomials such that h divides f. Then we have*

$$\|h\|_2 \;\leq\; \|h\|_1 \;\leq\; 2^m\, M(h) \;\leq\; \left|\frac{h_m}{f_n}\right| 2^m\, \|f\|_2.$$

For $h, f \in \mathbb{Z}[x]$ as in the Theorem 5.2.6, *Mignotte's bound* gives the following inequality for coefficients of h and f (see [Maz08]).

$$\|h\|_\infty \;\leq\; (n+1)^{1/2}\, 2^n\, \|f\|_\infty. \tag{5.2.2}$$

Geršgorin's Theorem

Geršgorin Circle Theorem localizes eigenvalues in $\mathbb{C}$, and it appears in the paper of Geršgorin in [Ger31]. We use this result to get bounds on the coefficients of the minimal polynomial of $A \in K^{n\times n}$. The theorem says that the eigenvalues of any $n \times n$ complex matrix can be included in n disks in the complex plain which can be easily obtained from the matrix. Let $A = [a_{i,j}] \in \mathbb{C}^{n\times n}$ and the collection of all eigenvalues of A be the *spectrum* $\sigma(A)$. i.e.,

$$\sigma(A) := \{\lambda \in \mathbb{C} \mid \det(\lambda I_n - A) = 0\}.$$

Now, we take $N := \{1, 2, \ldots, n\}$, and define the i-th deleted absolute row sum of A as:

$$r_i(A) := \sum_{j\in N\setminus\{i\}} |a_{i,j}| \quad \text{for } i \in N.$$

We use the convention that $r_1(A) := 0$ when $n = 1$.

For $i \in N$ the *i-th Geršgorin disk* of A is defined as:

$$\Gamma_i(A) := \{z \in \mathbb{C} \;\big|\; |z - a_{i,i}| \leq r_i(A)\}.$$

and $\Gamma(A) := \cup_{i\in N}\Gamma_i(A)$ is called the *Geršgorin set*. The set is closed and bounded in the complex plane $\mathbb{C}$, as it is the union of closed disks having center $a_{i,i}$ and radius $r_i(A)$ for $i \in N$. For any $A \in \mathbb{C}^{n\times n}$ we have that $\sigma(A) \subseteq \Gamma(A)$. This is the original result of Geršgorin, which is stated in the Theorem 5.2.7 (the proof and more details can be found in [Var04, Section 1.1]).

Theorem 5.2.7. *For any $A = [a_{i,j}] \in \mathbb{C}^{n\times n}$ and any $\lambda \in \sigma(A)$, there is a positive integer k in N such that*

$$|\lambda - a_{k,k}| \leq r_k(A).$$

Consequently it holds that

$$\sigma(A) \subseteq \Gamma(A).$$

Given $A \in \mathbb{C}^{n\times n}$, the *spectral radius* $\rho(A)$ of A is defined by

$$\rho(A) := \max\{|\lambda| \;\big|\; \lambda \in \sigma(A)\}.$$

The next corollary ([Var04, Corollary 1.2]) is a consequence of the Theorem 5.2.7.

Corollary 5.2.8. *For any $A = [a_{i,j}] \in \mathbb{C}^{n\times n}$, it holds*

$$\rho(A) \leq \max_{i\in N} \sum_{j\in N} |a_{i,j}|.$$

The corollary 5.2.9 which is stated in [Var04, Appendix B] generalizes the above result for any arbitrary norm ϕ on $\mathbb{C}^n$. Consider any matrix $A \in \mathbb{C}^{n\times n}$, so that A maps $\mathbb{C}^n$ into $\mathbb{C}^n$. Then the *induced operator norm* of A, with respect to ϕ is given by $\|A\|_\phi := \sup_{x\neq 0} \frac{\phi(Ax)}{\phi(x)} = \sup_{\phi(x)=1} \phi(Ax)$ which is simply the maximum absolute row sum of the matrix.

Corollary 5.2.9. *Given $A \in \mathbb{C}^{n\times n}$, let $\sigma(A)$ denote its spectrum as usual and let $\rho(A)$ denote its spectral radius. Then, for any norm ϕ on $\mathbb{C}^n$, $\rho(A) \leq \|A\|_\phi$.*

Bounds on the Coefficients of the Minimal Polynomial of $A \in \mathbb{C}^{n\times n}$

Similar to the case of characteristic polynomial, one can use the Hadamard bound to get bounds on the coefficients of the minimal polynomial. But, this estimate is too pessimistic in practice, when the degree of the minimal polynomial is smaller compared to the characteristic polynomial. Mignotte (5.2.2) gives a bound on the coefficients of minimal polynomial $M_A(\lambda)$, as a factor of the coefficients of characteristic polynomial $C_A(\lambda)$ as: $\|M_A\|_\infty \leq (n+1)^{1/2} 2^n \|C_A\|_\infty$, where n is the degree of C_A. Using a bound on the eigenvalues of A, Dummas in [Dum06, Section 3], gives a better bound on the coefficients of the minimal polynomial as follows.

Lemma 5.2.10. *Let $A \in \mathbb{C}^{n\times n}$ with its spectral radius bounded by $\beta \geq 1$. Let the minimal polynomial $M_A(\lambda) = \sum_{i=0}^{r} m_i \lambda^i$. Then for all $i = 1, \ldots, r$,*

$$|m_i| \leq \begin{cases} \beta^r & \text{if} \quad r \leq \beta \\ \min\{\sqrt{\beta} r^r,\ \sqrt{2/r\pi} 2^r \beta^r\} & \text{otherwise.} \end{cases}$$

The Lemma 5.2.10 is based on (5.2.1). Since the spectral radius is bounded by β, all the roots of the minimal polynomial are also bounded by β. Hence it holds that $|m_i| \leq \binom{r}{i} \beta^{r-i}$ for coefficients of the $M_A(\lambda)$. Using this result with some other bounds and inequalities on the binomial coefficients, Dummas has proven the above bound in [Dum06, Lemma 3.1].

Bounds on the Coefficients of the Minimal Polynomial of $A \in K^{n\times n}$

Let $M_A(\lambda) = \sum_{i=0}^{r} m_i \lambda^i$ be the minimal polynomial of the matrix $A \in K^{n\times n}$. Similar to the previous setting, let $m_j^{(i)}$ be the j-th coefficient of the minimal polynomial $M_{A^{(i)}}(\lambda)$ of the i-th conjugate matrix $A^{(i)}$ for $i = 1, \ldots, d$. Now we extend Lemma 5.2.10 to number fields as follows.

For each conjugate matrix, the coefficients of the minimal polynomial satisfies the following bound for $i = 1, \ldots, d$ and $j = 1, \ldots, r$.

$$|m_j^{(i)}| \leq \begin{cases} {\beta^{(i)}}^r & \text{if} \quad r \leq \beta^{(i)} \\ \min\{\sqrt{\beta^{(i)}} r^r,\ \sqrt{2/r\pi} 2^r {\beta^{(i)}}^r\} & \text{otherwise.} \end{cases} \tag{5.2.3}$$

Here, $\beta^{(i)} \geq 1$ is the bound for $\rho(A^{(i)})$ which can be computed using Corollary 5.2.7. Let $\|M_{A^{(i)}}\|_\infty = |m_j^{(i)}|$ for each conjugate matrices. One can get a bound on M_A, by considering the bounds on $M_{A^{(i)}}$ as follows.

$$\|M_A\|_\infty \leq \sum_{i=1}^{d} \|M_{A^{(i)}}\|_\infty^2. \tag{5.2.4}$$

5.3 Optimizing Characteristic Polynomial and Minimal Polynomial Computations

In this section, we present details of our optimized implementations with their performance analysis. We have done our implementations using the Hecke software package, and both our methods for characteristic polynomial computation and minimal polynomial computation perform better than the existing implementations in Hecke. We first discuss the details of our modular approach for computing the characteristic polynomial. Then we extend the characteristic polynomial implementation to compute the minimal polynomial using factorization techniques. Finally, we provide a verification method for the minimal polynomial.

5.3.1 Modular Methods

Here, we present the details of our implementation which use modular methods for computing characteristic polynomial $C_A(\lambda)$ and minimal polynomial $M_A(\lambda)$ of a matrix $A \in K^{n\times n}$. The algorithm computes the characteristic polynomial and minimal polynomial modulo a sequence of prime ideals, and then applies CRT to recover $C_A(\lambda)$ and $M_A(\lambda)$. Here we choose good primes (i.e. primes which does not divides coefficients of the polynomial), as the degrees of the modular result can change in the presence of bad primes (as explained in [GS02, Lemma 2.3] for the $\mathbb{Z}$ case). We can detect bad primes during the computation, checking degree sequences of modular result as explained in [FHHJ17, Section 4.2]. Algorithm 19 explains the CRT-modular approach for computing C_A. Similarly, M_A can be computed replacing the bound $\|C_A\|_\infty$ by $\|M_A\|_\infty$ and the algorithm `CharPoly` by `MinPoly`.

Algorithm 19 Characteristic Polynomial (`Modular_CharPoly`)

Input: $A \in K^{n\times n}$.
Output: Characteristic polynomial C_A of A.
1: Computes a bound $\|C_A\|_\infty$ on the coefficients of C_A using Lemma 5.2.4.
2: Choose t matching prime ideals $\mathfrak{p}_1, \mathfrak{p}_2, \ldots, \mathfrak{p}_t$ such that
$N(\mathfrak{p}) \geq (2\sqrt{\|C_A\|_\infty/d} \cdot (3\sqrt{\gamma(\theta)}/2)^{d-1})^d$ where $\mathfrak{p} = \prod_{i=1}^{t} \mathfrak{p}_i$.
3: **for** $i = 1, \ldots, t$ **do**
4: $A_{\mathfrak{p}_i} = A \pmod{\mathfrak{p}_i}$
5: Compute $C_{\mathfrak{p}_i} =$ `CharPoly`$(A_{\mathfrak{p}_i})$.
6: **end for**
7: Apply CRT to compute C_A such that $C_{\mathfrak{p}_i} \cong C_A \pmod{\mathfrak{p}_i}$ for $i = 1, \ldots, t$.
8: **return** C_A

Theorem 5.3.1. *Algorithm 19 is correct, and it has the complexity $O^\sim(n^4 d)$ over $\mathbb{Q}$.*

Proof. The correctness follows from the construction. As we have chosen enough primes according to Belabas' bound in Theorem 1.3.10, CRT gives the correct output. For `CharPoly` in the Step 5, we can choose any algorithm, available in the literature, as mentioned in Section 5.1.2. If we choose the

complexity of `CharPoly` as $O^{\sim}(n^3)$, then one iteration cost $O^{\sim}(n^3 d)$ operations in $\mathbb{Q}$, with the complexity $O^{\sim}(d)$ for algebraic operations in K. Therefore the total complexity of the algorithm is $O^{\sim}(n^4 d)$ operations in $\mathbb{Q}$, as the size of the output is $O^{\sim}(n)$ according to the bound. □

However, in our implementation, we use Algorithm 19 with a probabilistic approach. We can use the bound on coefficients to get an approximation for the required number of primes. Instead of computing the characteristic polynomial modulo the product of several prime numbers using CRT at once, we keep on increasing primes and compute the polynomial using CRT, until it becomes stable. We check the stability for 5 consecutive outputs. We have also optimized the computation of minimal polynomial, using the same strategy.

d	r / n	$-100:100$			$-1000:1000$			$-1000000:1000000$		
		`M_Char`	Hecke	Quot	`M_Char`	Hecke	Quot	`M_Char`	Hecke	Quot
3	10	0.07	0.00	0.01	0.10	0.00	0.01	0.02	0.00	0.10
	30	0.13	0.12	0.94	0.15	0.12	0.81	0.29	0.15	0.52
	50	0.76	1.03	1.36	0.93	1.03	1.10	1.72	1.19	0.69
	100	9.81	17.66	1.80	12.45	19.09	1.53	23.41	22.70	0.97
	150	49.83	1.6 m	1.99	63.58	1.9 m	1.83	1.8 m	2.2 m	1.26
	300	10.7 m	29.6 m	2.76	14.2 m	32.4 m	2.28	28.5 m	54.8 m	1.92
	500	1.9 h	5 h	2.60	2.4 h	5.7 h	2.30	4 h	7.3 h	1.81
5	10	0.07	0.00	0.03	0.04	0.00	0.06	0.02	0.00	0.19
	30	0.42	0.25	0.59	0.43	0.28	0.65	0.68	0.30	0.45
	50	2.42	2.05	0.85	2.70	2.23	0.83	3.98	2.59	0.65
	100	23.88	31.26	1.31	41.73	42.11	1.01	58.01	51.24	0.88
	150	2.7 m	3.7 m	1.37	3.3 m	4.3 m	1.28	4.5 m	5.2 m	1.15
	300	32.3 m	1.1 h	2.03	56 m	1.2 h	1.27	1.5 h	1.5 h	1.01
	500	4.5 h	10.5 h	2.32	5.7 h	8.3 h	1.45	7.5 h	-	∞
10	10	0.04	0.00	0.09	0.05	0.01	0.13	0.06	0.01	0.18
	30	1.10	0.85	0.77	1.45	0.94	0.65	2.05	1.65	0.81
	50	6.51	8.02	1.23	7.17	9.53	1.33	10.55	17.42	1.65
	100	1.3 m	4.5 m	3.28	1.6 m	5.3 m	3.21	2.6 m	7.2 m	2.73
	300	1.6 h	13.3 h	8.13	3 h	19.7 h	6.53	4.7 h	-	∞
20	10	0.10	0.01	0.12	0.09	0.02	0.17	0.17	0.03	0.16
	30	3.17	2.61	0.82	4.41	2.25	0.51	5.63	4.04	0.72
	50	19.87	24.37	1.23	25.81	31.86	1.23	36.85	51.19	1.39
	100	5.1 m	13.1 m	2.55	6.4 m	16.5 m	2.57	10 m	24 m	2.39
	300	6.7 h	51.1 h	7.55	7.9 h	63.6 h	8.04	9.5 h	-	∞

Table 5.1: Timing for Computing Characteristic Polynomial with Modular Algorithm.

Performance Analysis

To illustrate the efficiency of the new method, we have implemented a variant of `Modular_CharPoly` algorithm which uses probabilistic approach as explained before (see [Sur21]). We have tested timings for the number fields $K = \mathbb{Q}[x]/(x^3 + 7x + 1)$, $K_1 = \mathbb{Q}[x]/(f_1)$, $K_2 = \mathbb{Q}[x]/(f_2)$ and $K_3 = \mathbb{Q}[x]/(f_3)$

for $f_1 = x^5 + \sum_{i=0}^{4} 2(-x)^i$, $f_2 = x^{10} + \sum_{i=0}^{9} 2(-x)^i$ and $f_3 = x^{20} + \sum_{i=0}^{19} 2(-x)^i$. Table 5.1 shows timing in seconds (m -for minutes and h- for hours), for this new approach `Modular_CharPoly` (in the table as `M_Char`) vs the existing implementation in Hecke [FHHJ17] (Section 5.1.2). We use the parameters: d- degree of the number field, n- size of the matrix and r- range of the coefficients of input matrix. Experimental results (with quotient times: Quot) show that the new algorithm works much better than the Hecke implementation for big matrices in large degree fields. When the matrix size and r increase, Hecke fails to compute the characteristic polynomial (does not terminate).

5.3.2 Computing Minimal Polynomial Using Characteristic Polynomial

As we can see in the Section 5.1, there are highly optimized algorithms for characteristic polynomial computing. The latest algorithms can compute it with the same asymptotic complexity as the matrix multiplication, up to constant factors.

Characteristic polynomial computation algorithms have better complexity than minimal polynomial computation. Here we present an algorithm to compute the minimal polynomial using the characteristic polynomial. The strategy which we used is straightforward: first we factorize the polynomial C_A as $C_A = {f_1}^{e_1} {f_2}^{e_2} \cdots {f_m}^{e_m}$. Then we obtain the valuations of M_A at the factors f_i's, by comparing the multiplicities of the factors of M_A modulo a smaller prime ideal $\mathfrak{p}$ with $f_i \pmod{\mathfrak{p}}$. The prime is chosen such that the factors are square-free. It holds $M_{A \pmod{\mathfrak{p}}}$ is the same as $M_A \pmod{\mathfrak{p}}$ for all but finitely many bad primes. We choose a good prime in this application, one choice would be a prime larger than the coefficients of M_A.

Algorithm 20 Minimal Polynomial (`MinPoly_Fac`)

Input: $A \in O^{n \times n}$.
Output: Minimal polynomial of A.

1: Compute $C_A =$ `Modular_CharPoly`(A)
2: Factorize $C_A = {f_1}^{e_1} {f_2}^{e_2} \cdots {f_m}^{e_m}$.
3: Choose a suitable prime ideal $\mathfrak{p}$ of O.
4: $M_p =$ `MinPoly`$(A \pmod{\mathfrak{p}})$.
5: Compute $g_i \cong f_i \pmod{\mathfrak{p}}$ for $i = 1, \ldots, m$.
6: Compute $h_i =$ `valuation`(M_p, g_i) for $i = 1, \ldots, m$.
7: **return** $M_A = {f_1}^{h_1} {f_2}^{h_2} \cdots {f_m}^{h_m}$

Here, the function `valuation`(M_p, g_i) compute the valuation of M_p at g_i, that is, the largest k such that g_i^k divides M_p.

In the Algorithm 20, we use randomized factorization algorithms to obtain polynomial run times (see [CZ81]) as deterministic algorithms such as Berlekamp's factorization algorithm in [Ber70] are of exponential complexity. The `CharPoly` algorithm in Step 1 has the complexity $O^{\sim}(n^4 d)$ over $\mathbb{Q}$. As we use non-deterministic algorithms, the complexity of factorization is negligible. Therefore, the total complexity is in polynomial time. Although we can not prove that the algorithm is asymptotically fast, practically it behaves really well.

Theorem 5.3.2. *Algorithm 20 is correct.*

Proof. It is clear from the construction that the algorithm is correct as M_A divides C_A. □

Performance Analysis

We have implemented Algorithm `MinPoly_Fac` ([Sur21]) using Hecke [FHHJ17] in the Julia language [BEKS17]. To illustrate the efficiency of the new method, we have computed timings using

d	r / n	$-100:100$			$-1000:1000$			$-1000000:1000000$		
		`Min_F`	Hecke	Quot	`Min_F`	Hecke	Quot	`Min_F`	Hecke	Quot
3	10	0.01	0.02	1.66	0.02	0.02	1.39	0.02	0.14	8.98
	30	0.16	14.93	93.49	0.20	25.72	127.77	0.30	1.2 m	239.41
	50	1.127	9.4 m	501.29	1.22	14.1 m	696.80	1.81	29.2 m	966.92
	100	11.87	18 h	5476.01	14.46	-	∞	23.09	-	∞
5	10	0.02	0.36	16.76	0.02	0.07	3.83	0.04	0.16	4.04
	30	0.43	48.82	113.19	0.45	57.15	126.22	0.69	2.2 m	193.52
	50	2.47	20.8 m	505.10	2.48	31.4 m	762.78	3.96	1.1 h	1036.90
	100	26.59	37.8 h	5114.34	35.40	-	∞	57.42	-	∞
10	10	0.12	0.36	3.06	0.09	0.53	5.56	0.10	1.32	13.16
	30	1.14	3.3 m	141.05	1.60	5 m	190.53	2.25	11.8 m	317.18
	50	8.26	1.4 h	642.50	9.75	2.1 h	788.60	14.44	4.5 h	1131.42
20	10	4.62	6.14	1.33	5.55	9.18	1.66	7.25	1.2 m	10.71
	30	4.48	34.3 m	459.78	5.37	57.5 m	643.35	6.87	2.8 h	1477.88
	50	24.97	10.6 h	1530.14	28.25	17.6 h	2248.67	42.18	-	∞

Table 5.2: Timing for Computing Minimal Polynomial Using Characteristic Polynomial.

the number field $K = \mathbb{Q}[x]/(x^3 + 7x + 1)$, $K_1 = \mathbb{Q}[x]/(f_1)$, $K_2 = \mathbb{Q}[x]/(f_2)$ and $K_3 = \mathbb{Q}[x]/(f_3)$ for $f_1 = x^5 + \sum_{i=0}^{4} 2(-x)^i$, $f_2 = x^{10} + \sum_{i=0}^{9} 2(-x)^i$ and $f_3 = x^{20} + \sum_{i=0}^{19} 2(-x)^i$. Table 5.2 shows timing in seconds (m -for minutes and h- for hours) for the new algorithm 20 (`Min_F`) for different choices of parameters: d- degree of the number field, n- size of the matrix and r- range of the coefficients of input matrix. Comparison have been made with the existing implementation in Hecke [FHHJ17], which uses Krylov method as explained in Section 5.1.3. Experimental results (with quotient times) shows that the new algorithm is slower for small examples, and superior for big matrices in large degree fields: For 100×100 matrix, over a field of degree 5 the new implementation computes the minimal polynomial, nearly 5000 times faster than the existing one.

5.3.3 Certification for the Minimal Polynomial

In the previous algorithm, as a certification, we have used bounds on the coefficients of the minimal polynomial to obtain the correct output. When we use the probabilistic approach (compute until the output stabilizes, without using bounds) we need a method to certify the minimal polynomial.

The minimal polynomial $g = M_A$ is a monic polynomial with the minimum degree (say r) such that $g(A) = 0$. Therefore, for all non-zero $v \in V = \mathbb{R}^{n \times n}$, it should holds that

$$g(A)v = 0 \implies g(A)/K_A(V, v) = 0 \tag{5.3.1}$$

where $K_A(V, v) = \left\langle A^i v \,|\, i = 0, 1, \ldots \right\rangle$ is the Krylov space of v. We use this fact in Algorithm 21 as a certification or the minimal polynomial of a given $A \in K^{n \times n}$.

Algorithm 21 Certification for Minimal Polynomial (`MinPoly_Cert`)

Input: $A \in K^{n\times n}$ and $g(\lambda)$.
Output: If the minimal polynomial of A is correct true, otherwise false.

```
1: v ∈ V
2: K_A = K_A(V, v).
3: while true do
4:    if g(A)v ≠ 0 then
5:       return false
6:    end if
7:    if rank(K_A) = n then
8:       return true
9:    else
10:      v ∈ V\K_A
11:   end if
12:   K_A = K_A ∪ K_A(V, v)
13: end while
```

Theorem 5.3.3. *The Algorithm 21 is correct and has the complexity $O(n^4d)$ over $\mathbb{Q}$.*

Proof. Let v_1 be a random vector in $V = \mathbb{R}^{n\times n}$ (i.e $v_1 = e_1$ the first canonical basis). Suppose that (5.3.1) is satisfied for $v = v_1$. If $\text{rank}(K_A(V, v_1)) = n$, we have that (5.3.1) is true for all the vectors in V. Otherwise, we choose another random vector $v_2 \in V$ independent from the subspace $K_A = K_A(V, v_1)$ and check whether (5.3.1) is satisfied for v_2. We continue this process, to check if (5.3.1) is true for all the vectors in V. That is, Algorithm 21 terminates when $V = \langle K_A(V, v_1), \ldots, K_A(V, v_k)\rangle$ for some $k \leq n$. If (5.3.1) is not true for any $v \in V$, then the $g(\lambda)$ is not the minimal polynomial, and returns false.

The steps of the algorithm are similar to the minimal polynomial computation, using Krylov method. Therefore the same complexity holds. □

5.3.4 A Factorization Free Algorithm

In this section, we present an algorithm which computes the minimal polynomial of an integer matrix, in the time of characteristic polynomial computation. We have used the coprime basis computation techniques to get rid of expensive factorization step.

Coprime Bases Computation

Here, we state the general theory of coprime bases and basic idea of the algorithm for the computation of coprime bases. Coprime base computation uses divisions and gcd computations of polynomials, Therefore, it has a better complexity than polynomial factorization. The following definitions and results are generalizations of Bernstein's results in [Ber05, Chapters 4, 13], over polynomial ring $F[x]$ over a field F (finite fields, $\mathbb{Z}$ or K).

Definition 5.3.4. *Let S and B be subsets of $F[x]$. If each element of S is product of powers of elements of B, then B is a base for S. B is called coprime, if all elements of B are pairwise coprime. B is called a coprime base for S, if B is a base for S and B is coprime.*

The existence of the coprime basis is given by the following theorem ([Ber05, Theorem 4.1]).

Theorem 5.3.5. *Every finite subset of $F[x]$ has a finite coprime base.*

In our application we need a method to compute the coprime base of two elements. We use an algorithm for two element input, due to Bach, Driscoll and Shallit [BDS90]. It keeps track of the order of elements and uses this to avoid superfluous computations.

Algorithm 22 Coprime Base of Two Polynomial (`CoprimeBase`)

Input: Two polynomials $f, g \in F[x]$.
Output: List B, such that B is the coprime base of $\{f, g\}$.

$h_1 = f$, $h_2 = g$ and $i = 1$.
$B = \{h_1, h_2\}$
while $i < \#B$ **do**
  $g = \gcd(h_i, h_{i+1})$
  if g is a unit **then**
    $i = i + 1$
  else
    replace h_i by $h_i g^{-1}$ and h_{i+1} by $h_{i+1} g^{-1}$ in B.
    insert g as the new $i + 1$-st element of B.
    remove h_i, h_{i+1}, h_{i+2} if they are units.
  end if
end while
return B

Using the same idea, Bernstein in [Ber05, Chapter 13] improves the computations by using asymptotically fast algorithms. Given a finite set S of monic polynomials over a finite field, his algorithm factors S into coprimes in essentially linear time.

Algorithm for Minimal Polynomial Computation

Algorithm 23 presents a factorization free approach to obtain the minimal polynomial from the characteristic polynomial. The method is straightforward: first, we compute the characteristic polynomial using a fast approach. Then we compute the coprime factorization of $C_p = C_A \pmod p$ and $M_p = M_A \pmod p$ over $\mathbb{Z}_p$ for some good prime p, and valuations of C_p and M_p for each factor. We use Hensel lifting to obtain the coprime factors of C_A in characteristic 0, using a large enough prime power precision. Finally, the computed valuations are used to get the correct exponents of the coprime-factors of M_A, using fast method for root computation.

Algorithm uses multi-factor Hensel lifting to lift a factorization into arbitrary many factors, as explained in [GG13, Chapter 15.5].

Given input: a prime $p \in F$, $f \in F[x]$ such that $\mathrm{lc}(f)$ is a unit modulo p, monic non-constant polynomials $f_1, \ldots, f_r \in F[x]$ that are pairwise coprime modulo p and satisfy $f \equiv \mathrm{lc}(f) f_1 \cdots f_r \pmod p$, and a lift $l \in \mathbb{N}$.

The function `MultFacHensel`$([f_1, \ldots, f_r], f, p, l)$ computes output: monic polynomials $F_1, \ldots, F_r \in F[x]$ with $f \equiv \mathrm{lc}(f) F_1 \cdots F_r \pmod{p^l}$ and $F_i \equiv f_i \pmod p$ for all i.

If $F = \mathbb{Z}$, $p \in \mathbb{N}$ is a prime, $\deg(f) = n$, $\|f\|_\infty < p^l$, and $\|f_i\|_\infty < p$ for all $i = 1, \ldots, r$, then the algorithm `MultFacHensel` takes $O^{\sim}(nl \log(p))$ operations in $\mathbb{Z}$ (see [GG13, Theorem 15.18]).

Theorem 5.3.6. *Algorithm 23 is correct and has the same asymptotic complexity, up to constant factors, as the matrix multiplication.*

Proof. Since coprime base computation and Hensel factorization have linear complexity, the total cost of the algorithm is similar to the cost of characteristic polynomial computation. As stated in Section 5.1.2,

Algorithm 23 Minimal Polynomial (`MinPoly_Coprime`)

Input: $A \in \mathbb{Z}^{n\times n}$
Output: Minimal polynomial of A.

1: Compute $C_A = \texttt{CharPoly}(A)$
2: Choose a small prime $p \in \mathbb{Z}$ and a prime precision k such that $p^k > 2\rho(A)$.
3: $C_p = C_A \pmod p$
4: $M_p = \texttt{MinPoly}(A \pmod p)$
5: Let $B = \texttt{CoprimeBase}(C_p, M_p)$ be $B = \{b_1, \ldots, b_m\}$.
6: **for** $i = 1, \ldots, m$ **do**
7: $e_i = \texttt{valuation}(C_p, b_i)$ and $h_i = \texttt{valuation}(M_p, b_i)$
8: **end for**
Then $C_p = b_1^{e_1} \ldots b_m^{e_m}$ and $M_p = b_1^{h_1} \ldots b_m^{h_m}$.
9: **if** $e_i = h_i$ for all $i = 1, \ldots, m$ **then**
10: **return** C_A
11: **else**
12: For $i = 1, \ldots, m$ compute $l_i \in \mathbb{Q}$ such that $e_i = l_i h_i$.
That is $C_p = b_1^{l_1 h_1} \cdots b_m^{l_m h_m}$ and $M_p = b_1^{h_1} \cdots b_m^{h_m}$.
13: Compute $[B_1, \ldots, B_m] = \texttt{MultFacHensel}([b_1^{l_1 h_1}, \ldots, b_m^{l_m h_m}], C_A, p, k)$ such that $B_i = b_i^{l_i h_i} \pmod{p^k}$.
14: For $i = 1, \ldots, m$, recover $\tilde{B}_i = b_i^{h_i} \pmod{p^k}$ using the l_i th root of B_i.
15: **return** $\tilde{B}_1 \cdots \tilde{B}_m$
16: **end if**

we have a algorithm which computes the characteristic polynomial in the time of matrix multiplication by Neiger in [NP20]. The root computation at Step 14 is done without the complete factorization (using $\gcd(f, f')$, where f' is the first derivative of f). □

Extension over Number Fields as a Future Work

We can extend the same strategy to obtain a faster minimal polynomial computation algorithm over number fields. There is also a version of Hensel's lemma over $\mathcal{O}_\mathfrak{p}$ (the p-adic completion of $\mathcal{O}$). We can combine this with the theory of coprime bases to compute the minimal polynomial in the time of matrix multiplication, as explained in the previous section.

Chapter 6

Linear Algebra over Univariate Polynomial Ring over Finite Fields

6.1 Introduction

There are many optimized algorithms in the literature for computing determinants for integer matrices and matrices over polynomial rings. The aim of this chapter is to discuss determinant computation algorithms, over polynomial ring over finite fields, using the CRT-approach and Storjohann's approach in [PS13]. The chapter begins with a literature survey of algorithms for determinant computation, with their complexity improvements. Then, we develop the required sub-algorithms for solving linear systems, reconstructing rational polynomials and certifying unimodularity. To develop an optimized reconstruction method, we have successfully used the half-gcd approach. Finally, we present a determinant computation algorithm, as an extension of Strojohann's determinant computation approach given for integer matrices.

6.1.1 Preliminaries on Polynomial Matrices

Before we dive into the algorithms, we will discuss some notations and definitions which will be used in this chapter. The algorithms in the literature use building blocks such as shifted degree, minimal kernel and column basis of matrices. Moreover, we need to know matrix transformations into normal forms, in particular, the *Hermite form* for triangularization and the *Popov form* for row reduction of matrices.

Basic Notations

Let $A = [a_{i,j}] \in \mathbb{F}[x]^{n\times m}$, where $\mathbb{F}[x]$ is a univariate polynomial ring with coefficients from a finite field $\mathbb{F}$. The problem of row reduction (or lattice reduction) of a matrix A over $\mathbb{F}[x]$, is about finding a basis with row degrees as small as possible for the lattice (denoted by $\Delta(A)$) generated by all $\mathbb{F}[x]$-linear combinations of rows of A. Two matrices $A, B \in \mathbb{F}[x]^{n\times m}$ are *left equivalent* (the rows of A and B generate the same lattice) if and only if $A = UB$ for a unimoduar matrix U. Similarly, it is right equivalent iff $A = BU$. Let $v \in \mathbb{F}[x]^n$ be a vector (row or column matrix over $\mathbb{F}[x]$), The *degree* of v denoted by $\deg(v)$, is the maximal degree of all entries of v. Similarly, we define $\deg(A)$ to be the maximum degree of the matrix entries of A. By $\mathrm{rdeg}(A)$ we denote the list of $\deg(A_i)$ for each row A_i of A. The *pivot index* of v, denoted by $\mathrm{piv}(v)$ is the index of the rightmost entry of degree $\deg(v)$. The leading coefficient matrix of A, denoted by $\mathrm{LC}(A)$, is the matrix over $\mathbb{F}$ formed by taking the leading coefficients (lc) of each entries of A as $\mathrm{LC}(A) = [\mathrm{lc}(a_{i,j})]$.

Normal Forms

Definition 6.1.1 (Row-reduce & Popov forms)**.**

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \left[\begin{array}{c} a_1 \\ \hline a_2 \\ \hline \vdots \\ \hline a_n \end{array}\right] \in \mathbb{F}[x]^{n\times m}$$

Matrix A is row reduced if $\operatorname{rdeg}(A) \leq \operatorname{rdeg}(UA)$ *for any unimodular matrix U. If, in addition, A satisfies the following normalization conditions it is in Popov form.*

(i) The pivot indices $\operatorname{piv}(a_1), \ldots, \operatorname{piv}(a_n)$ *are distinct.*

(ii) The pivot entries $a_{1,\operatorname{piv}(a_1)}, \ldots, a_{n,\operatorname{piv}(a_n)}$ *are monic.*

(iii) $\deg(a_i) \leq \deg(a_{i+1})$ *for* $1 \leq i \leq n$*, and if* $\deg(a_i) = \deg(a_{i+1})$ *then* $\operatorname{piv}(a_i) < \operatorname{piv}(a_{i+1})$.

(iv) Non-pivot entries have degree less than that of the pivot entry in the same column

If A satisfies only condition (i) it is said to be in weak Popov form.

For any nonsingular $A \in \mathbb{F}[x]^{n\times n}$, A has a unique decomposition $A = UP$ with U unimodular and P in Popov form, in particular it holds that $\deg(P) \leq \deg(A)$ (See [MS03] and [SS11]). There is a similar definition for *row reduce form* as $\operatorname{LC}(A)$ being full row rank. In [Zho13, Lemma 2.14] the equivalence of two definitions has been proven.

Example 6.1.2. *Consider the matrix* $A \in \mathbb{F}_7[x]^{3\times 3}$.

$$A = \begin{bmatrix} 2x+4 & 5x^4+4x^2+5x+1 & x^3+3x^2+x+2 \\ 5x^5+5x^3+2x^2 & 2x^2+1 & 3x^2+5x+5 \\ x^4+x^3+2x+6 & 5x^2+2x+3 & 4x^4+6x^3+6 \end{bmatrix}$$

The weak Popov form of A is given by

$$W = \begin{bmatrix} 2x+4 & \underline{5x^4+4x^2+5x+1} & x^3+3x^2+x+2 \\ \underline{5x^5+5x^3+2x^2} & 2x^2+1 & 3x^2+5x+5 \\ x^4+x^3+2x+6 & 5x^2+2x+3 & \underline{4x^4+6x^3+6} \end{bmatrix}$$

The Popov form of A is

$$P = \begin{bmatrix} 6x+5 & \underline{x^4+5x^2+x+3} & 3x^3+2x^2+3x+6 \\ 2x^4+2x^3+4x+5 & 3x^2+4x+6 & \underline{x^4+5x^3+5} \\ \underline{x^5+x^3+6x^2} & 6x^2+3 & 2x^2+x+1 \end{bmatrix}$$

The pivot entries in each row have been underlined.

Definition 6.1.3 (Hermite normal form)**.** *A matrix $A \in \mathbb{F}[x]^{n\times m}$ is in Hermit normal form if*

(i) A is upper triangular,

(ii) The diagonal elements of A are monic,

(iii) $\deg(a_{ij}) < \deg(a_{jj})$ *for* $1 \leq i < j \leq n$.

The Hermit normal form is unique, and for every $A \in \mathbb{F}[x]^{n\times m}$ there exists a unimodular $U \in \mathbb{F}[x]^{n\times n}$ such that $UA = H$ in Hermit normal form (See [MS03] and [NRS18]).

Shifted Degrees

For a vector $p = [p_1, \ldots, p_n] \in \mathbb{F}[x]^n$ the *shift column degree* is defined as the maximum degree after shifting the degrees of p by a given integer vector that is known as a *shift*. If $s = [s_1, \ldots, s_n] \in \mathbb{Z}^n$, the shifted column degree of p with respect to the shift s is given as

$$\deg_s(p) = \max_{1 \le i \le n}[\deg(p_i) + s_i].$$

The shifted degrees are used for efficient multiplication of a pair of matrices with unbalanced degrees.

Kernel and Column Bases

The *kernel* of $A \in \mathbb{F}[x]^{m \times n}$ is the $\mathbb{F}[x]$-module $\ker(A)$

$$\ker(A) = \{p \in \mathbb{F}[x]^n \mid Ap = 0\}$$

with a *kernel basis* of A being a basis of this module. Similarly, a *column basis* of A is a basis for the $\mathbb{F}[x]$-module as

$$\{Ap \mid p \in \mathbb{F}[x]^n\}.$$

Efficients algorithms for computing kernel bases in [ZLS12] and column bases in [ZL13] take the use of shifted degrees.

Lemma 6.1.4. *Let s be* $\deg(A)$ *of the polynomial matrix $A \in \mathbb{F}[x]^{m \times n}$. There exists a deterministic algorithm (in [ZL13]) which can compute the column basis of A using $O^{\sim}(nm^{\omega-1}s)$ field operations, and kernel basis of A using $O^{\sim}(nm^{\omega-1}s)$ field operations.*

Here, ω is the exponent of matrix multiplication, which satisfies $2 \le \omega \le 3$ as explained in [GG13, Chapter 12].

6.1.2 Cost Model

For complexity analysis in polynomial rings over finite fields, we define a non-decreasing function $M : \mathbb{Z}_{\ge 0} \to \mathbb{R}_{>0} : d \mapsto M(d)$, which gives the number of operation in $\mathbb{F}$, when two polynomials in $\mathbb{F}[x]$ of degree at most d are multiplied. We will also have the following assumptions on M:

1. $M(d) \in O^{\sim}(d)$.
2. $M(d_1 d_2) \le M(d_1)M(d_2)$ for all $d_1, d_2 \in \mathbb{Z}$.

Here for polynomial multiplications, we use asymptotically fast methods such as FFT-based methods which has the complexity $O(d \log(d) \log(\log(d)))$.

The cost of polynomial divisions and gcd related computations has the complexity bound $M(d)\log(d)$. Since we use $O^{\sim}$ asymptotic notation in complexity analysis, we assume that for Euclidean algorithm related computations has the same complexity as $M(d)$ for multiplication.

In this section, for linear algebra such as matrix multiplication, inverse computation, normal form computations, we use asymptotically fast methods with complexity $O^{\sim}(n^{\omega}M(d))$, where d is the $\deg(A)$ (See [Sto03, Chapter 2]).

6.2 Determinant Computation Algorithms in the Literature

Most of the algorithms for determinant computation that we can find in the literature are based on computing a unimodular matrix $U \in \mathbb{F}^{n\times n}$ such that $AU = H$ is triangular. Moreover triangularizing a matrix is useful for solving linear system and computing matrix normal forms. In this regards, Strojohann in [Sto00, Algorithm 2.18] presents a fraction-free Gaussian elimination to triangularize a matrix, and then he uses this in a recursive deterministic algorithm to compute the determinant with a cost of $O^{\sim}(n^{\omega+1}d)$ operations where $d = \deg(A)$. A deterministic $O^{\sim}(n^3d^2)$ algorithm was later given by Mulders and Storjohann in [MS03] (see Section 6.2.1), modifying their algorithm for weak Popov form computation. Later in [Sto03], Storjohann reduces the complexity to $O(n^{\omega}\log^2 nd^{1+\epsilon})$ field operations using higher-order lifting and probabilistic approach. The deterministic algorithm `Determinant` (Algorithm 4), which is presented in the Section 2.2 by Abbott et al in [ABM99] can also be used over any arbitrary commutative ring. Abbott's algorithm has the complexity of $O^{\sim}(n^4)$ operations over $\mathbb{F}$.

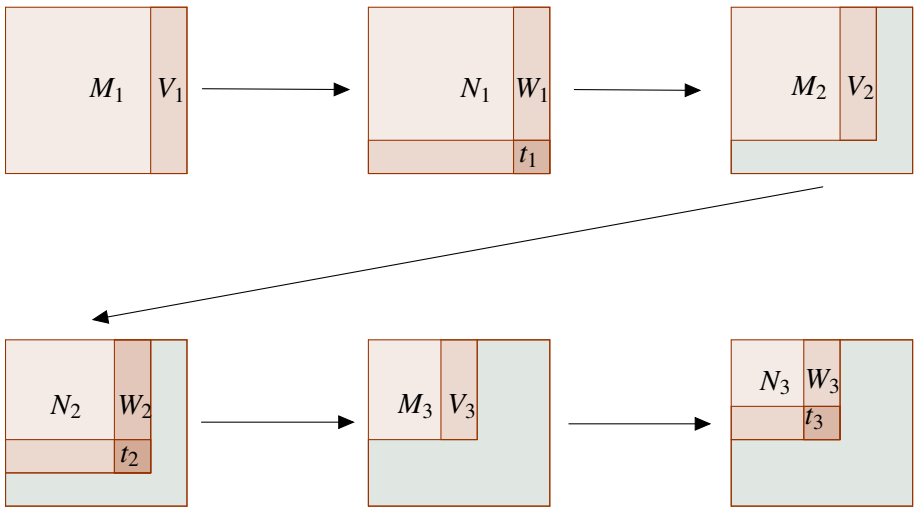
6.2.1 Mulders' and Storjohann's Method

Algorithm 24 Determinant Computation (`Hecke_det`)

Input: $A \in \mathbb{F}[x]^{n\times n}$ nonsingular.
Output: Determinant of A.

1: $C = A$ and $d_e = 1$.
2: **for** $i = n-1, n-2, \ldots, 1$ **do**
3: M_i = first i columns of C.
4: V_i = last columns fo C.
5: $\left[N_i \mid V_i\right] =$ `ExtendedWeakPopovForm`(M_i, V_i).
6: Let k such that k-th row of N_i is zero.
7: $C = N_i$ with row k deleted.
8: $t_i = k$-th entry of W_i.
9: $d_e = (-1)^{k+i+1}t_i d_e$
10: **end for**
11: **return** $C_{11}d_e$

Algorithm 24 computes the determinant of a non-singular matrix $A \in \mathbb{F}[x]^{n\times n}$ using the weak Popov form. Mulders and Storjohann gave this iterative algorithm which has the complexity $O^{\sim}(n^3\det(A)^2)$ over $\mathbb{F}$. It reduce the exponent of n compared to their previous algorithm in [Sto00], but at the cost of increasing the exponent of d. The algorithm uses a variant of the weak Popov form. That is: given two matrices M and V `ExtendedWeakPopovForm`(M, V) computes (N, W) such that N is a weak Popov form of M and W is obtained by applying the same transformation on V. The figure below describes the algorithm in detail. In Hecke, this is used for computing the determinant of polynomial matrices. At the end of this chapter, we give a performance analysis of this algorithm versus our new approach.



Algorithm 25 Determinant Computation (`det_poly`)

Input: $A \in \mathbb{F}[x]^{n \times n}$ nonsingular.
Output: Determinant of A.

1: **while** true **do**
2: **if** $n = 1$ **then**
3: **return** A
4: **end if**
5: $\begin{bmatrix} A_u \\ A_d \end{bmatrix} = A$, with $A_u = A_{[1:\lceil n/2 \rceil, 1:n]}$ and $A_d = A_{[\lfloor n/2 \rfloor : n, 1:n]}$.
6: $B_1, U_r, V_u := \texttt{ColumnBasis}(A_u)$
Note: `ColumnBasis`(A_u) returns the column basis B_1 and kernel basis U_r of A, and the right factor V_u such that $A_u = B_1 V_u$.
7: $B_2 = A_d U_r$
8: $u_r = U_r \pmod{x}$, $v_u = V_u \pmod{x}$.
9: Compute a matrix $u_l^* \in \mathbb{F}^{n \times \lceil n/2 \rceil}$, such that $u^* = \begin{bmatrix} u_l^* & u_r \end{bmatrix}$ is unimodular.
10: $d_v = \det(v_u u_l^*) / \det(u^*)$
11: $d_B = \texttt{det\_poly}(B_1) \cdot \texttt{det\_poly}(B_2)$
12: **return** $d_v d_B$
13: **end while**

6.2.2 Zhou's and Labahn's Method

Zhou and Labahn in [ZL14] present a fast and deterministic algorithm for finding the determinant of A, with the complexity $O^{\sim}(n^{\omega} d)$ operations over the field $\mathbb{F}$. They use a fast algorithm in [ZLS12] for computing minimal kernel basis. A kernel basis computation allows to partition the input matrix A into a block triangular form as:

$$A \cdot U = \begin{bmatrix} A_u \\ A_d \end{bmatrix} \begin{bmatrix} U_l & U_r \end{bmatrix} = \begin{bmatrix} B_1 & O \\ * & B_2 \end{bmatrix} = B \tag{6.2.1}$$

Now the problem is reduced to the computations of the determinants of B_1, B_2 and U as $\det(A) = \det(B_1)\det(B_2)/\det(U)$. The determinant of U is a non-zero element in $\mathbb{F}$, and it can be obtained without

computing the entire matrix U. The computation of $\det(B_1)$ and $\det(B_2)$ is done recursively, until the matrices reach smaller dimension that is easy to compute determinants, (as explained in Algorithm 25 from [LNZ17, Section 4]).

Computing Block-Diagonal Matrices

Consider (6.2.1), for $AU = B$:

$$\begin{bmatrix} A_u \\ A_d \end{bmatrix} \begin{bmatrix} U_l & U_r \end{bmatrix} = \begin{bmatrix} B_1 & O \\ * & B_2 \end{bmatrix}.$$

In each iteration of the algorithm, the input $n \times n$ matrix A is partitioned into two sub-matrices A_u and A_d consisting of the upper $\lceil n/2 \rceil$ and lower $\lfloor n/2 \rfloor$ rows of A, respectively. The column dimensions of the sub-matrices U_l and U_r of the transformation matrix U, match with the row dimension of A_u and A_d, respectively.

With the relation $A_u U_l = B_1$, we have that the block matrix B_1 is a column basis of A_u, therefore the Lemma 6.1.4 can be used to compute B_1.

Since $A_u U_r = O$, U_r is the right kernel basis of A_u, and it can be used to compute the block-matrix B_2 where, $B_2 = A_u U_r$. It is stated in [LNZ17, Lemma 3.1]) that this computation is independent of the chosen kernel basis U_r.

Computing the Determinant of the Unimodular Matrix

Let $V = \begin{bmatrix} V_u & V_d \end{bmatrix}^t$ be the inverse matrix of U such that the sub-matrices V_u and V_d are compatible with the dimensions of A_u and A_d as shown below.

$$BV = \begin{bmatrix} B_1 & O \\ * & B_2 \end{bmatrix} \begin{bmatrix} V_u \\ V_d \end{bmatrix} = \begin{bmatrix} A_u \\ A_d \end{bmatrix} = A.$$

We compute a sub-matrix U_l^* such that $U^* = \begin{bmatrix} U_l^* & U_r \end{bmatrix}$ is unimodular. Then, it holds that:

$$\det(V)\det(U^*) = \det(VU^*) = \det\left(\begin{bmatrix} V_u \\ V_d \end{bmatrix} \begin{bmatrix} U_l^* & U_r \end{bmatrix} \right) = \det\left(\begin{bmatrix} V_u U_l^* & O \\ * & I \end{bmatrix} \right) = \det(V_u U_l^*).$$

Therefore it holds that $\det(V) = \det(V_u U_l^*)/\det(U^*)$, and the determinant of A can be computed as:

$$\det(A) = \frac{\det(B)\det(V_u U_l^*)}{\det(U^*)}.$$

Finally we need the following lemma for the completion of the proof of Algorithm 25.

Lemma 6.2.1. *If $U \in \mathbb{F}[x]^{n \times n}$ is unimodular, then* $\det(U) = \det(U \pmod{x})$.

More details with the complexity analysis of Algorithm 25 can be found in [ZL14] and [LNZ17].

6.2.3 The State of the Art

Neiger and Pernet in [NP20] give the first description of an algorithm achieving the complexity of $O(n^\omega M(D/n))$ operations in $\mathbb{F}$, where $D = \deg(\det(A))$ is equal to the sum of the degrees of the rows of A.

Their approach basically follows the algorithm of Zhou and Labahn in [ZL14], and it uses the properties of reducing and normalizing matrices. Let us rewrite the (6.2.1) as

$$U \cdot A = \begin{bmatrix} * & * \\ U_1 & U_2 \end{bmatrix} \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} B_1 & * \\ 0 & B_2 \end{bmatrix} = B \tag{6.2.2}$$

where the entries $*$ are not computed, $B_2 = U_1A_2 + U_2A_4$, B_1 and $[U_1\ U_2]$ are computed from $[A_1\ A_3]^t$ as a row basis and a kernel basis. Here, the kernel basis computation and the matrix multiplication giving B_2 (the product $U_1A_2 + U_2A_4$) involves matrices with possibly unbalanced degrees. Therefore, they use the current fastest method for this, i.e. they split the computation into $O(\log(n))$ multiplications of smaller matrices which have balanced degrees [ZLS12, Section 3.6]). However, the computation of the row basis B_1 remains an obstacle which prevents the determinant algorithm of [ZL14] from having complexity $O(n^\omega M'(D/n)$. This computation has a loop over $\log(n)$ iterations, each of them calling [ZL12, Algorithm 2] for minimal approximate bases with unbalanced input. Therefore Neiger and Pernet, in their algorithm in [NP20, Section 3] follow an approach which is more direct at first: keep the first block row of A as in (6.2.3)

$$U \cdot A = \begin{bmatrix} I_{n/2} & * \\ U_1 & U_2 \end{bmatrix} \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ 0 & B \end{bmatrix} \tag{6.2.3}$$

and rely on the identity $\det(A) = \det(A_1)\det(B)/\det(U_2)$. Algorithm has a total cost of $O(n^\omega M'(D/n))$ for the three recursive calls, where the first two are $(n/2)\times(n/2)$ matrices (A_1 and U_2) whose determinant has degree at most $D/2$, and at most D for the third.

Although this approach removes the expensive row basis computation which arises in [ZL14], it adds a requirement that all recursive calls must be carried out with reduced matrices. The reducedness of A_1 and B is achieved using [SS11, Section 3]. The fastest known algorithm in [ZLS12] for computing kernel bases outputs a matrix in shifted reduced form (s -weak popov form in [NP20, Section 2.4]). Since, the fastest algorithms in [SS11] are not given for shifted forms, they extend normalizing methods for shifted forms in [NP20, Section 4].

6.3 Rational Function and Vector Reconstruction

In this section, we present a fast algorithm to find a rational function, that is congruent to some polynomial modulo another polynomial. The standard strategy is to use the Extended Euclidean Algorithm (EEA), which is also used to compute the gcd of two polynomials. Here, we take the use of Half-gcd algorithm to optimize the implementation for rational function reconstruction. At the end of the section we address the vector rational reconstruction problem with a literature review.

Given input a nonzero modulus $f \in \mathbb{F}[x]$, a single image polynomial $g \in \mathbb{F}[x]$ with $\deg(g) < \deg(f)$, and degree bounds $0 \le N < \deg(f)$ and $0 \le D < \deg(f)$, the rational function reconstruction problem asks for a pair of polynomials (r, t) such that $tg \equiv r \pmod f$. If $\gcd(t, g) = 1$, then we have $r/t \equiv g \pmod f$ as a valid solution (See [GG13, 5.7] and [KM06]).

The generalization to vector rational function reconstruction problem is defined similarly except with g replaced by $[g_1, \ldots, g_n] \in F[x]^{1\times n}$. The problem asks for a pair $(t, [r_1, \ldots, r_n])$ such that $tr_i \equiv g_i \pmod f$, $\deg(t) \le D$ and $\deg(r_i) \le N$ for $i = 1, \ldots, n$. Similarly, if $\gcd(t, f) = 1$, we have a valid solution as $r_i/t \equiv g_i \pmod{f_i}$ for $i = \ldots, n$.

6.3.1 Extended Euclidean Algorithm

In this section we define three functions `quot`, `rem` and `divrem` which are uniquely associated to any two elements a and b in a Euclidean domain R with $b \neq 0$ as: $q = \texttt{quot}(a, b)$, $r = \texttt{rem}(a, b)$ and $(q, r) = \texttt{divrem}(a, b)$ where $q, r \in R$ such that $a = qb + r$ and $\deg(r) < \deg(b)$. Before we look at the

EEA, we will see the traditional Euclidean Algorithm (EA) which computes greatest common divisors of any given two elements f and g in an arbitrary Euclidean domain as follows.

$r_0 = f$, $r_1 = g$ and $i = 1$.
while $r_i \neq 0$ **do**
  $r_{i+1} = \texttt{rem}(r_{i-1}, r_i)$, and $i = i + 1$
end while
return r_{i-1}

The EEA computes not only the gcd but also a representation of it as a linear combination of the inputs. Here, we mention only the steps of the EEA in (6.3.1), and the algorithm can be found in [GG13, Algorithm 3.6].

Let R be a Euclidean domain, $r_0, r_1 \in R\backslash\{0\}$ with $\deg(r_0) \geq \deg(r_1)$. The EEA starts with $s_0 = t_1 = 1$, $s_1 = t_0 = 0$, and then proceed as:

$$\begin{array}{ccc} r_2 = r_0 - q_1 r_1, & s_2 = s_0 - q_1 s_1, & t_2 = t_0 - q_1 t_1, \\ \vdots & \vdots & \vdots \\ r_{i+1} = r_{i-1} - q_i r_i, & s_{i+1} = s_{i-1} - q_i s_i, & t_{i+1} = t_{i-1} - q_i t_i, \\ \vdots & \vdots & \vdots \\ 0 = r_{\ell-1} - q_\ell r_\ell, & s_{\ell+1} = s_{\ell-1} - q_\ell s_\ell, & t_{\ell+1} = t_{\ell-1} - q_\ell t_\ell. \end{array} \tag{6.3.1}$$

Here, $\deg(r_{i+1}) < \deg(r_i)$ for $1 \leq i \leq \ell$, as in the EA. We assume that $r_{\ell+1} = 0$ and $\deg(r_{\ell+1}) = -\infty$.

Lemma 6.3.1. *Let* $Q_i = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \in R^{2\times 2}$ *and* $R_i = Q_i \cdots Q_1 = \begin{pmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{pmatrix}$ *for* $0 \leq i \leq \ell$. *Then it holds that,*

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} = Q_i \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} = Q_i \cdots Q_1 \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = R_i \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}.$$

Let $m_i = \deg(r_i)$ for $0 \leq i \leq \ell$, then $\deg(q_i) = m_{i-1} - m_i$ for $1 \leq i \leq \ell$. The sequence $(m_0, m_1, \ldots, m_\ell)$ is called the *degree sequence* in the Extended Euclidean Algorithm for r_0 and r_1.

6.3.2 Extended Euclidean Algorithm for Rational Function Reconstruction

Consider two polynomials $f \in \mathbb{F}[x], \deg(f) = m > 0$, and $g \in \mathbb{F}[x]$ of $\deg(g) < m$. Then we can use the EEA to find a rational function $r/t \in \mathbb{F}(x)$, with $r, t \in \mathbb{F}[x]$, satisfying

$$\gcd(t, f) = 1 \text{ and } rt^{-1} \equiv g \pmod{f}, \quad \deg(r) < k, \ \deg(t) \leq m - k, \tag{6.3.2}$$

for a given $k \in \{0, \ldots, m\}$. Here t^{-1} is the inverse of t modulo f. Note that, if $k = m$, then clearly $r = g$ and $t = 1$ is a solution, but for other values of k, it is not obvious. Since, the degree of f is m, the constrains on degrees of r and t as given in (6.3.2) are sensible.

In order to take the use of the EEA, we obtain a weaker condition of the (6.3.2) as follows. Since t is a unit modulo f, we can multiply the congruence in (6.3.2) by t and obtain an equivalent condition as:

$$r \equiv tg \pmod{f}, \quad \deg(r) < k, \ \deg(t) \leq m - k. \tag{6.3.3}$$

Despite of some exceptional cases where (6.3.2) has no solution, we can show that (6.3.3) can always be satisfied using EEA, as mentioned in the Theorem 6.3.2 (see [GG13, Section 5.7]).

A rational function $r/t \in \mathbb{F}(x)$, with $r, t \in \mathbb{F}[x]$ is said to be in *canonical form* if t is monic and $\gcd(r, t) = 1$. Every rational function has a unique canonical form.

Theorem 6.3.2. *Let $f \in \mathbb{F}[x]$ of degree $m > 0$ and $g \in \mathbb{F}[x]$ of degree less than m. By applying the EEA for f and g, consider the j-th row $r_j, s_j, t_j \in \mathbb{F}[x]$, where j is the minimal such that $\deg(r_j) < k$ (for a given k).*

(i) There exist polynomials $r, t \in \mathbb{F}[x]$ satisfying (6.3.3)*, namely $r = r_j$ and $t = t_j$. If in addition $\gcd(r_j, t_j) = 1$, then r and t also solve* (6.3.2)*.*

Moreover it holds that $r_j = s_j f + t_j g$.

(ii) If $r/t \in \mathbb{F}(x)$ is a canonical form solution to (6.3.2)*, then $r = \tau^{-1} r_j$ and $t = \tau^{-1} t_j$, where $\tau \in \mathbb{F}\backslash\{0\}$ is the leading coefficient of t_j. In particular,* (6.3.2) *is solvable if and only if $\gcd(r_j, t_j) = 1$.*

(See [GG13, Theorem 5.16])

6.3.3 Half-GCD Algorithm

The basic idea leading to a fast gcd algorithm for two polynomials $r_0, r_1 \in \mathbb{F}[x]$ is that the first quotients q_i's of EA only depend on the coefficients of higher degrees of r_0 and r_1. In this section, we elaborate on this idea which is used in the Half-GCD (HGCD) algorithm.

Let $f = f_m x^m + f_{m-1} x^{m-1} + \cdots + f_0 \in \mathbb{F}[x]$ with leading coefficient $f_n \neq 0$, and $k \in \mathbb{Z}$. Then we define the truncated polynomial

$$f \upharpoonright k = \mathtt{quot}(f, x^{m-k}) = f_m x^k + f_{m-1} x^{k-1} + \cdots + f_{m-k}.$$

If $f = 0$, then $f \upharpoonright k = 0$. Also, if $k < 0$ or $k = -\infty$ we define $f \upharpoonright k = 0$.

Definition 6.3.3. *Let $f, g, f^*, g^* \in \mathbb{F}[x]$ with f, f^* both non-zero, $\deg(f) \geq \deg(g)$ and $\deg(f^*) \geq \deg(g^*)$, and $k \in \mathbb{Z}$. Then, we say that (f, g) and (f^*, g^*) coincide up to k, if*

$$\begin{aligned} f \upharpoonright k &= f^* \upharpoonright k, \\ g \upharpoonright (k - (\deg(f) - \deg(g))) &= g^* \upharpoonright (k - (\deg(f^*) - \deg(g^*))). \end{aligned}$$

Consider the steps of the EA for two pairs r_0, r_1 and r_0^*, r_1^* of polynomials with $\deg(r_0) > \deg(r_1)$ and $\deg(r_0^*) > \deg(r_1^*)$ of lenght ℓ and ℓ^*:

$$\begin{array}{ll} r_0 = q_1 r_1 + r_2, & r_0^* = q_1^* r_1^* + r_2^*, \\ \vdots & \vdots \\ r_{i-1} = q_i r_i + r_{i+1}, & r_{i-1}^* = q_i^* r_i^* + r_{i+1}^*, \\ \vdots & \vdots \\ \vdots & r_{\ell^*-1}^* = q_{\ell^*}^* r_{\ell^*}^*. \\ r_{\ell-1} = q_\ell r_\ell, & \end{array} \tag{6.3.4}$$

Let $n_i = \deg(q_i)$ for $1 \leq i \leq \ell$ and $n_i^* = \deg(q_i^*)$ for $1 \leq i \leq \ell^*$. Then, $m_i = \deg(r_i) = m_0 - n_1 - \cdots - n_i$ for $0 \leq i \leq \ell$ and $m_{\ell+1} = -\infty$, similarly m_i^* are defined for $0 \leq i \leq \ell^* + 1$. Given $k \in \mathbb{N}$, we define the number $\eta(k) \in \mathbb{N}$ by

$$\eta(k) = \max\{0 \leq j \leq \ell : \sum_{i=1}^{j} n_i \leq k\},$$

so that

$$m_0 - m_{\eta(k)} = \sum_{i=1}^{\eta(k)} n_i \leq k < \sum_{i=1}^{\eta(k)+1} n_i = m_0 - m_{\eta(k)+1}. \tag{6.3.5}$$

Here, the second inequality only holds if $\eta(k) < \ell$, and $\eta(k)$ is uniquely determined by (6.3.5). We define $\eta^*(k)$ analogously.

Now the following lemma [GG13, Lemma 11.3] gives the required relation (relation between quotients and input polynomials of EA,) for the HGCD algorithm.

Lemma 6.3.4. *Let $k \in \mathbb{N}$, $h = \eta(k)$, and $h^* = \eta^*(k)$. If (r_0, r_1) and (r_0^*, r_1^*) coincide up to $2k$, then $h = h^*$ and $q_i = q_i^*$ for $1 \le i \le h$.*

In our implementation for the rational polynomial reconstruction, we only need the Lemma 6.3.4. The complete HGCD algorithm can be found in [GG13, Algorithm 11.6].

6.3.4 Half-GCD Algorithm for Rational Polynomial Reconstruction

In this section, we introduce a fast algorithm for rational polynomial reconstruction using HGCD algorithm. In order to make the description and the proofs simpler, we have directly taken the first few steps from [GG13, Algorithm 11.6]. The algorithm will not create the monic remainders in intermediate steps, but it is possible to obtain monic output at the end, if desired. Given two polynomials $r_0, r_1 \in \mathbb{F}[x]$, with degree $m = \deg(r_0) \ge \deg(r_1)$, and a bound on the degree of the denominator $k \in \mathbb{N}$ with $0 \le k \le n$, Algorithm 26 computes the polynomials $r, t \in \mathbb{F}[x]$, such that $r/t \cong r_1 \pmod{r_0}$. We denote $\mathrm{lc}(f)$ be the leading coefficient of any polynomial f.

Algorithm 26 Rational Polynomial Reconstruction (`RatPolyRcon`)

Input: $r_0, r_1 \in \mathbb{F}[x]$, $m = m_0 = \deg(r_0) \ge \deg(r_1) = m_1$, and $k \in \mathbb{N}$ with $0 \le k \le m$.
Output: r and t such that $r \cong t r_1 \pmod{r_0}$.

1: **if** $r_1 = 0$ or $k < m - m_1$ **then**
2: **return** $r_1, 1$
3: **end if**
4: **if** $k = 0$ and $m - m_1 = 0$ **then**
5: $q = \mathrm{lc}(r_0)/\mathrm{lc}(r_1)$
6: **return** $r_0 - q r_1, -q$.
7: **end if**
8: $R = I_2$
9: $d = \lceil k/2 \rceil$ and $s = 0$.
10: $r = r_0 \upharpoonright (2d - 2)$
11: $\tilde{r} = r_1 \upharpoonright (2d - 2 - (m - m_1))$
12: **while** $s \le d - 1$ and $\tilde{r} \ne 0$ **do**
13: $q, r_e = \texttt{divrem}(r, \tilde{r})$
14: $s = s + \deg(q)$
15: $Q = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$
16: **if** $s \le d - 1$ **then**
17: $R = QR$
18: **end if**
19: $r, \tilde{r} = \tilde{r}, r_e$
20: **end while**
21: $r = R_{2,1} r_0 + R_{2,2} r_1$, $t = R_{2,2}$
22: **return** r, t

Theorem 6.3.5. *Algorithm 26 is correct.*

Proof. If $r_1 = 0$ or $k < m - m_1$, then $\eta(k) = 0$. Therefore, the while loop does not have to be involved, as s should stay 0 (the variable s keeps track on the value of $\eta(d)$). Since, R stays as the identity matrix the algorithm returns numerator r_1 and denominator 1. (Suppose that we compute denominators regardless of considering the case $k < m - m_1$. Then the degrees of the denominators will exceed the given bound k, as the degree of the first quotient $\deg(q_1) > k$).

If $k = 0$ and $m - m_1 = 0$, then the first quotient is $q_1 = \mathrm{lc}(r_0)/\mathrm{lc}(r_1)$. Therefore, the only possible denominator is $\pm q_1$ and the correct result is returned.

Otherwise, $k \geq 1$ and $k \geq m - m_1$, and we have defined $(r, \tilde{r})$ from (r_0, r_1), such that they coincide up to $2(d-1)$. Therefore by the Lemma 6.3.4, considering the EEA outputs of two pairs of polynomials $(r, \tilde{r})$ and (r_0, r_1), we have that the corresponding quotient polynomials in both cases are equal up to the row $\eta(d-1)$. As described in the Lemma 6.3.1, the matrix R give the outputs of EEA. Hence, the correct numerator and denominator is returned by the algorithm as explained in the Theorem 6.3.2(i). Furthermore, according to the (6.3.5) we have that $n_1 + n_2 + \cdots + n_{\eta(d-1)} \leq d - 1 < k$ where $n_i = \deg(q_i)$ for quotient polynomials q_i.

□

Example 6.3.6. *Consider two polynomials r_0 and r_1 over $\mathbb{F}_7[x]$: $r_0 = 6x^8 + x^7 + 3x^6 + 5x^5 + 6x^4 + 4x^2 + 2x + 2$, $r_1 = 3x^7 + x^6 + x^3 + x^2 + 4x + 1$. We compute r and t such that $r \equiv t r_1 \pmod{r_0}$ using Algorithm 26* `RatPolyRecon` *as follows:*

$d = \deg(r_0)/2 = 4$.
$r = r_0 \upharpoonright (2d-2) = 6x^6 + x^5 + 3x^4 + 5x^3 + 6x^2 + 4$.
$\bar{r} = r_1 \upharpoonright ((2d-2) - (\deg(r_0) - \deg(r_1))) = 3x^5 + x^4 + x + 1$.
Round 1:
$q_1, r_{e_1} =$ `divrem`$(r, \bar{r}) = (2x + 2,\ x^4 + 5x^3 + 4x^2 + 3x + 2)$.
Then, $s = \deg(q_1) = 1$ *and* $Q_1 = \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 5x+5 \end{pmatrix}$.
Round 2:
$r, \bar{r} = \bar{r}, r_{e_1} = (3x^5 + x^4 + x + 1,\ x^4 + 5x^3 + 4x^2 + 3x + 2)$.
$q_2, r_{e_2} =$ `divrem`$(r, \bar{r}) = (3x,\ 2x^3 + 5x^2 + 2x + 1)$.
Then, $s = \deg(q_1) + \deg(q_2) = 2$, *and* $Q_2 = \begin{pmatrix} 0 & 1 \\ 1 & -q_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 4x \end{pmatrix}$.
Round 3:
$r, \bar{r} = \bar{r}, r_{e_1} = (x^4 + 5x^3 + 4x^2 + 3x + 2,\ 2x^3 + 5x^2 + 2x + 1)$.
$q_3, r_{e_3} =$ `divrem`$(r, \bar{r}) = (4x + 3,\ 2x^2 + 6)$.
Then $s = \deg(q_1) + \deg(q_2) + \deg(q_3) = 3$, *and* $Q_3 = \begin{pmatrix} 0 & 1 \\ 1 & -q_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 3x+4 \end{pmatrix}$.
Round 4:
$r, \bar{r} = \bar{r}, r_{e_1} = (2x^3 + 5x^2 + 2x + 1,\ 2x^2 + 6)$.
$q_4, r_{e_4} =$ `divrem`$(r, \bar{r}) = (x + 6, 3x)$.
$s = \deg(q_1) + \deg(q_2) + \deg(q_3) + \deg(q_4) = 4$
Since $s > d - 1$, while loop terminates with:
$R = Q_3 Q_2 Q_1 = \begin{pmatrix} 4x & 6x^2 + 6x + 1 \\ 5x^2 + 2x + 1 & 4x^3 + 4x + 2 \end{pmatrix}$.
Now a numerator and a denominator can be computed as: $r = R_{2,1} r_0 + R_{2,2} r_1$, and $t = R_{2,2}$, and algorithm returns $(r, t) = (4x^4 + x^2 + 4x + 4,\ 4x^3 + 4x + 2)$.
The solution can be normalized by clearing denominators as $(x^4 + 2x^2 + x + 1,\ x^3 + x + 4)$.

Algorithm 26 provides only the outputs suggested in the weaker condition (6.3.3) of the rational polynomial reconstruction. In order to obtain an output which satisfies the (6.3.2), we extend the Algorithm 26 as:

if $\gcd(r,t) \neq 1$ **then**
  return false
else
  return true, $(r/\operatorname{lc}(t), t/\operatorname{lc}(t))$
end if

Algorithm 26 outputs the rational polynomial r/t, with the degree of the denominator polynomial satisfying $\deg(t) \leq \lceil k/2 \rceil - 1$. We have $r = ur_0 + tr_1 \cong tr_1 \pmod{r_0}$, for some $u \in F[x]$. According to the (6.3.2), if $\gcd(r_0, t) = 1$ then $(r/\operatorname{lc}(t), t/\operatorname{lc}(t))$ is a canonical form solution to the rational reconstruction problem. Here, we make the test $\gcd(r,t) = 1$ instead of $\gcd(r_0,t) = 1$, since we have that $\gcd(r,t) = \gcd(r_0,t)$ by Lemma 6.3.7 below. It is more efficient to check the coprimality of the final numerator and denominator, instead of checking the invertibility of denominator ($t \pmod{r_0}$), since the size of $\deg(r) < \deg(r_0)$.

Lemma 6.3.7. *Let r_0, r and t be as defined above, then we have that* $\gcd(r,t) = \gcd(r_0,t)$.

Proof. Suppose that $p \in F[x]$ be a divisor of t. If $p|r_0$, then clearly $p|ur_0 + tr_1 = r$. On the other hand, if $p|r$, then $p|ur_0 = r - tr_1$, where u and t are coprime (by the properties of EEA, see [GG13, Lemma 3.8]). Hence the lemma holds. □

6.3.5 Extension to Vector Rational Function Reconstruction

Here, we present a vector rational function reconstruction algorithm, using the same approach that we used in Section 1.3.6. First, we use a variant of the Algorithm 26 to find the common denominator of the solution vector. Then, we use the common denominator to recover numerators of the solution vector.

Note that for our application, we choose $k = \deg(r_0)$ to be the degree bound which is mentioned in Algorithm 26. First we introduce the sub-algorithm for denominator reconstruction as `DenomRecon`. This is obtained by replacing the Steps 15 and 17 of the Algorithm 26 by the following equation:

$$R_{1,2}, R_{2,2} = R_{2,2}, R_{1,2} - qR_{2,2}. \tag{6.3.6}$$

The (6.3.6) gives the necessary computations, to recover a denominator. When the conditions of the while loop (Step 12) are satisfied, the algorithm `DenomRecon` returns a suitable denominator $t = R_{2,2}$ such that $r \cong tr_1 \pmod{r_0}$ for some $r \in \mathbb{F}[x]$; (we use this as a code $\texttt{DenomRecon}(r_1, r_0) = t$).

Computing Common Denominator and Rational Solution Vector

Let s be the solution S modulo the irreducible polynomial p, of some linear system over $\mathbb{F}[x]$ (i.e. $s = S \pmod{p}$). Algorithm 27 computes the solution vector $S \in \mathbb{F}(x)^{n\times 1}$ and the common denominator d_s of S. The correctedness of Algorithm 27 follows from the Theorem 1.3.20 as we use the same idea of Algorithm 2. The algorithm is efficient in practice.

6.3.6 Vector Rational Function Reconstruction by Storjohann and Olesh

Similar to the vector reconstruction methods over integers (Section 1.2.4), we have algorithms for vector rational function reconstruction, which requires half of the required image computation than it is required for element wise rational function reconstruction.

Algorithm 27 Vector Reconstruction (`VecPolyRecon`)

Input: $s \in \mathbb{F}[x]^{n\times 1}$ and $p \in \mathbb{F}[x]$
Output: $S \in \mathbb{F}(x)^{n\times 1}$ such that $S = s \pmod p$ and common denominator d_s of S.

```
B = ⌈deg(p)/2⌉
S = O_{n×n} and d_s = 1.
for i = 1, ..., n do
    a = d_s s_i (mod p)
    if deg(a) ≤ B then
        S_i = a/d_s
    else
        t = DenomRecon(a, p)
        d_s = t d_s
        if deg(d_s) > B then
            return false, S, d_s
        end if
        a = d_s s_i (mod p)
        S_i = a/d_s
    end if
end for
return true, S, d_s
```

In order to find a solution via element wise reconstruction, it requires $\deg(f) > N + D$ to ensure the uniqueness of the solution space. Storjohann and Olesh in [OS07] presents a fast algorithm for vector rational function reconstruction problem using $O(nk^{\omega-1}M(\deg(f)))$ operations in $\mathbb{F}$, with the requirement that $\deg(f) > N + D/k$ for a small constant k. Their algorithm is based on simultaneous Padé approximation and minimal approximants bases computations.

Minimal Approximant Bases

We need the following definitions following [BL94] and [OS07] for minimal approximant bases computation.

Definition 6.3.8 (defect)**.** *The defect of a vector $g = (g_1, g_2, \ldots, g_n) \in \mathbb{F}[x]^n$ with respect to the fixed multi-index $w = (w_1, w_2, \ldots, w_n)$ is defined by*

$$\text{defect}(g) = \text{defect}(g, w) = \min_i\{w_i + 1 - \deg(g_i)\}.$$

Definition 6.3.9. *A matrix $B = [B_1^t|B_2^t|\ldots|B_r^t]^t \in \mathbb{F}[x]^{r\times m}$ is a reduced basis of type w for $A \in \mathbb{F}[x]^{n\times m}$ if the following conditions are satisfied.*

(i) B has a full row rank and $\Delta(B) = \Delta(A)$.

(ii) Each $b \in \Delta(B)$ admits a unique decomposition $b = \sum_{i=1}^{r} c_i B_i$ with $c_i \in \mathbb{F}[x]$, $\deg(c_i) \le \text{defect}(B_i) - \text{defect}(b)$, $1 \le i \le r$.

The sub-matrix of the reduced basis B comprised of the rows with positive defect is called the positive part of B.

Now we define the *minimal approximant bases* as follows. Let $G \in \mathbb{F}[x]^{n\times m}$, $w \in \mathbb{Z}^n$, and $d \in \mathbb{Z}_{\ge 0}$.

Definition 6.3.10. *An order d minimal approximant of type w for G is a reduced basis $M \in K[x]^{n\times n}$ of type w for the lattice $\{y \in \mathbb{F}[x]^{1\times n} \mid yG \equiv 0 \pmod{x^d}\}$. Here, M is nonsingular, and satisfies $MG \equiv 0 \pmod{x^d}$.*

In [GJV03, Section 2], Giorgi et al. provide an algorithm to compute M using matrix multiplications, with the cost of $O(n^\omega M(d))$ operations in $\mathbb{F}$. A similar definition for M can be found in [BLV99, Definition 5.1], which uses a kernel basis of a shifted popov form.

Given G, d and w as above, `MinBasis` algorithm computes the output $(M, \delta) \in (\mathbb{F}[x]^{n\times n}, \mathbb{Z}^n)$, an order d minimal approximant M of type w for G together with a tuple $\delta = (\delta_1, \ldots, \delta_n)$ of the defects of rows of M. The algorithm `PosMinBasis`(G, d, w) computes the output of `MinBasis`(G, d, w) restricted to the rows with positive defect; M could also be a $0 \times n$ matrix.

The next two lemmas are the basic tools of the algorithm for simultaneous Padé approximation.

$$\left[\begin{array}{c|c} M & \\ \hline & I_k \end{array}\right] \overset{H}{\left[\begin{array}{c} * \\ \hline 0 \end{array}\right]} \equiv 0 \pmod{x^d} \tag{6.3.7}$$

Lemma 6.3.11. *Let $H \in \mathbb{F}[x]^{n\times m}$ have its last k rows zero and let $w = (w_1, \ldots, w_n)$. If $M \in \mathbb{F}[x]^{(n-k)\times(n-k)}$ is an order d minimal approximant of type $(w_1, \ldots, w_{n-k})$ for the first $n-k$ rows of H, then* $\mathrm{diag}(M, I_k)$ *is an order d minimal approximant of type w for H as in* (6.3.7)*.*

The next lemma allows us to compute the positive part of the minimal approximant bases using recursive approach. Here, 1 denote the vector $(1, 1, \ldots, 1)$.

Lemma 6.3.12. *Let $H \in \mathbb{F}[x]^{n\times m}$ and $H' \in \mathbb{F}[x]^{n\times m'}$. If $(M, \delta) =$ `PosMinBasis`(H, d, w) and $(M', \delta') =$ `PosMinBasis`$(MH', d, \delta - 1)$, then $(M'M, \delta')$ solves the minimal approximant problem with the input $([H|H'], d, w)$.*

Padé Approximation

An algorithm to compute the minimal approximant of an input matrix of the form (6.3.8) has been explained in [OS07, Section 3.1].

$$G = \begin{bmatrix} G_1 & G_2 & \cdots & G_n \\ E & & & \\ & E & & \\ & & \ddots & \\ & & & E \end{bmatrix} \in \mathbb{F}[x]^{(m+tn)\times nk}, \tag{6.3.8}$$

Here, $G_i \in \mathbb{F}[x]^{m\times k}$ and $E \in \mathbb{F}[x]^{t\times k}$. Let $w \in \mathbb{Z}^{m+tn}$ such that $n = (n_1, n_2, \ldots, n_n)$ with $n_1 \in \mathbb{Z}^m_{\geq 0}$ and $n_2 = n_3 = \cdots = n_n \in \mathbb{Z}^t_{\geq 0}$. The approach is to compute the positive part of an order d approximant for each non-zero block of matrices, starting from the upper-left part of (6.3.8).

Let $[\bar{M}|*] \in \mathbb{F}[x]^{s\times(m+t)}$ for $\bar{M} \in \mathbb{F}[x]^{s\times m}$ be the positive part of an order d minimal approximant of type (n_1, n_2) for $[G_1|E]^t \in \mathbb{F}[x]^{(m+t)\times k}$ with a vector $\delta \in \mathbb{Z}^s_{>0}$ of defects of rows. That is

$$\left[\bar{M} \;\middle|\; *\right] \left[\begin{array}{c} G_1 \\ \hline E \end{array}\right] \equiv 0 \pmod{x^d}.$$

Now consider the two block of k column matrices H and H' as

$$\left[\begin{array}{c|c} H & H' \end{array}\right] = \left[\begin{array}{c|c} G_1 & G_2 \\ E & \\ \hline & E \end{array}\right] \in \mathbb{F}[x]^{(m+2t)\times 2k}.$$

By Lemma 6.3.11 we have that $\operatorname{diag}([\bar{M}|*], I_t)$ with defect vector $(\delta, n_2 + 1)$, as the positive part of an order d minimal approximant of type (n_1, n_2, n_2) of H.

$$\left[\begin{array}{c|c|c} \bar{M} & * & \\ \hline & & I_t \end{array}\right] \begin{matrix} H \\ \left[\begin{array}{c} G_1 \\ E \\ \hline 0 \end{array}\right] \end{matrix} \equiv 0 \pmod{x^d}$$

Let $(M', \delta') = \texttt{PosMinBasis}(\operatorname{diag}([\bar{M}|*], I_t)H', d, (\delta, n_2 + 1) - 1)$, then $M' \operatorname{diag}([\bar{M}|*], I_t)$ is the positive part of an order d minimal approximant of type (n_1, n_2, n_2) for $[H|H']$ by the Lemma 6.3.12.

The key observation is that, we can proceed without computing the unknown block $*$ of $[\bar{M}|*]$ due to the following result:

$$\left[\begin{array}{c|c|c} \bar{M} & * & \\ \hline & & I_t \end{array}\right] \left[\begin{array}{c} G_2 \\ 0 \\ \hline E \end{array}\right] = \left[\begin{array}{c} \bar{M}G_2 \\ \hline E \end{array}\right],$$

For the rest of the block matrices in G, this can be applied recursively.

Therefore, the output of the $\texttt{PosMinBasis}(G, d, (n_1, n_2, \ldots, n_2))$ is given by the following steps, which is also known as the Padé approximation.

$(\bar{M}, \delta) = (I_m, n_1 + 1)$;
for $i = 1, \ldots, n$ **do**
  $\delta = (\delta, n_2 + 1)$
  $(M', \mu) = \texttt{PosMinBasis}\left(\left[\begin{array}{c} \bar{M}G_i \\ \hline E \end{array}\right], d, \delta - 1\right)$
  $\bar{M} = M'\bar{M}$
end for
return $(\bar{M}, \delta)$

The dimension of $\bar{M}$ remains bounded by k throughout the computation, therefore the cost of the algorithm is $O((nk + m)k^{\omega-1}M(d))$ operations in $\mathbb{F}$ as explained in [OS07, Section 3.1]. In [RS21] Rosenkilde and Storjohann give a new deterministic algorithm for fast computations of minimal approximant bases and Hermite -Padé approximations using the same complexity.

Vector Rational Function Reconstruction

Consider the vector rational function reconstruction problem for the input, a non-zero modulus $f \in \mathbb{F}[x]$, a vector $g \in \mathbb{F}[x]^{1\times n}$ with $\deg(g) \leq \deg(f)$, and degree bounds N and D for numerators and denominator of solutions. Let

$$S = \{[t \,|\, r] \in \mathbb{F}[x]^{1\times(n+1)} \mid tg \equiv r \pmod{f},\ \deg(t) \leq D,\ \deg(r) \leq N\}.$$

Consider the nonsingular matrix,

$$G = \left[\begin{array}{c|c} 1 & g \\ \hline 0 & fI_n \end{array}\right] \in \mathbb{F}[x]^{(n+1)\times(n+1)}.$$

If we set the degree constrains $(D, N, \ldots, N)$, then $[t\,|r] \in S$ if and only if $[t\,|r] \in \Delta(A)$ with defect$([t\,|r]) > 0$. Therefore, S is generated by the positive part of a reduced basis of type $(D, N, \ldots, N)$ for G. By extending this result, an algorithm for solving the vector rational reconstruction problem is given in [OS07, Section 4].

6.4 Linear System Solving and Bounds on Solutions

One of the major step in Storjohann's determinant computation algorithm is linear system solving. In this section, we present an algorithm for linear system solving based on the Dixon algorithm in [Dix82]. The algorithm computes the common denominator, so that we can use it directly in the determinant computation. We first compute bounds on the solutions. By Cramers rule we know that the solution of the linear system can be represented as determinants of matrices. Therefore, we compute the degree bound for determinant of the matrix $A \in \mathbb{F}[x]^{n\times n}$.

6.4.1 Degree Bound on the Determinant

In the cases of integers and number fields, we can use Hadamard's bound to get a size bound on the determinant. For matrices over polynomial rings over finite fields, we can use a similar method to get a degree bound on the determinant of A. Given $\mathbb{F}[x]^{n\times n}$, let `DetBound` be the algorithm, which computes $D \in \mathbb{Z}$ such that $\deg(\det(A)) \leq D$. We can take D as:

$$D = \sum_{i=1}^{n} \deg(A_i) \tag{6.4.1}$$

Gupta et al in [GSSV12, Corollary 2] gives a similar bound on the determinant using permutation as in [Lan02, Proposition XIII 4-4.6]. However, the degree bound given in Algorithm 6.4.1 is easier to compute. It is even easier to get a bound on $\det(A)$ using the Corollary 6.4.1, which immediately follows from (6.4.1).

Corollary 6.4.1. *It holds that* $\deg(\det(A)) \leq n \deg(A)$ *where* $\deg(A)$ *is the maximum degree of the entries of* A.

Algorithm 28 Solving Linear Systems (`SolverPolyMat`)

Input: $A \in \mathbb{F}[x]^{n\times n}$ and $b \in \mathbb{F}[x]^{n\times 1}$.
Output: $S \in \mathbb{F}[x]^{n\times 1}$ such that $AS = b$ and the common denominator d_s of S.

1: Choose a suitable irreducible polynomial p_0 of degree d.
2: $A_p = (A \pmod p)^{-1}$
3: $B =$ `DetBound`(A)
4: $s = O_{n\times 1}$ and $p = 1$
5: **for** $i = 1, 2, \ldots, \lceil B/d \rceil$ **do**
6: $\quad y = A_p b \pmod{p_0}$
7: $\quad s = s + py$
8: $\quad p = pp_0$
9: $\quad b = (b - Ay)/p_0$
10: **end for**
11: $\_, S, d_s =$ `VecPolyRecon`(s, p)
12: **return** S, d_s

6.4.2 Linear System Solving

Algorithm 28 presents the Dixon method for solving the linear system $Ax = b$ over $\mathbb{F}[x]$. Here we have assumed that the size of the degrees of matrix entries in b are also same as in A. In order to get a more precise bound on denominators and numerators of the solution, we can also apply (6.4.1) for the matrix $[A|b]^t$. The algorithm will give the correct solution, as we have considered enough precision.

6.5 Unimodular Certification

In order to check the unimodularity of the matrix A, we can use an integrality test for A^{-1}, using a similar method as Storjohann explains in [PS12]. In the case of integers, he uses higher order lifting to test whether the p-adic expansion of A^{-1} is finite. This is done by computing the error term of the p-adic expansion for a given precision, without completely computing the expansion. In our case also, we can use the double-plus-one lift which they have used to stay within small coefficients and reach the required precision within a few steps. Over polynomial rings over finite fields, one of the useful properties of double-plus-one lift is that we need few iterations to obtain the result and all the computations are carried out with small degrees. The following lemma captures the essential idea of linear-lifting and quadratic-lifting which will be used in the construction of double-plus-one in our case. Proof of the lemma follows from theorems 3.2.3 and 3.2.4.

Lemma 6.5.1. *Let $A \in \mathbb{F}[x]^{n\times n}$ be non-singular and $p \in \mathbb{F}[x]$ is an irreducible polynomial which is relatively prime to* $\det(A)$.

1. *If $A^{-1} = B + A^{-1}Rp^k$ for some $B, R \in \mathbb{F}[x]^{n\times n}$ and $k \in \mathbb{Z}_{>0}$, then for $M \in \mathbb{F}[x]^{n\times n}$ such that $M = A^{-1}R \pmod p$ it holds that $A^{-1} = B + Mp^k + A^{-1}R'p^{k+1}$ where $R' = (1/p)(R - AM)$.*

2. *For any B, R such that $A^{-1} = B + A^{-1}Rp$, we have $A^{-1} = B(I + Rp) + A^{-1}R^2p^2$.*

Algorithm 29 Unimodular Certification (`UniCertPolyMat`)

Input: $A \in \mathbb{F}[x]^{n\times n}$.
Output: `true` if A is unimodular, `false` otherwise.

Choose an irreducible polynomial $p \in \mathbb{F}[x]$ with $p \nmid \det(A)$ and $\deg(p) \geq \deg(A)$.
$k = \lceil \log_2(n/2) \rceil$
$B_0 = A^{-1} \pmod p$
$M_0 = B_0$, $R_0 = I$
for $i = 0 : k - 1$ **do**
  $T_p = R_i^2$
  $R_{i+1} = \frac{1}{p}(T_p - AM_i)$
  if $R_{i+1} = O$ **then**
    break
  end if
  $M_{i+1} = B_0T_p \pmod p$
  if $i = k - 1$ **then**
    return `false`
  end if
end for
return `true`

The construction of the double-plus-one lift over $\mathbb{F}[x]$ is similar to the case of integers in [PS12, Section 3], (or the construction of A^{-1} expansion in Section 3.2.2). Now we can use this in the unimodular certification of matrices over $\mathbb{F}[x]$, as explained in Algorithm 29.

We can also use Algorithm 29 to get the p-adic expansion of A^{-1}. Define $p_i = p^{2^{i+1}-1}$, then $p_{i+1} = p_i^2 p$. We start with $A^{-1} \pmod{p_0} = B_0$. Then for $i = 0, 1, 2, ..$ in succession we have:

$$\begin{aligned} A^{-1} \pmod{p_1} &= B_0(I + R_0 p_0) + M_0 p_0^2 \\ A^{-1} \pmod{p_2} &= B_1(I + R_1 p_1) + M_1 p_1^2 \\ &\vdots \end{aligned}$$

Continuing this we obtain the following sparse inverse expansion:
$A^{-1} \pmod{p_i} = (\cdots((B_0(I + R_0 p_0) + M_0 p^2)(I + R_1 p_1) + M_0 p_1^2) + \cdots + M_{i-1} p_{i-1}^2)$

Before proving the correctedness of Algorithm 29, we will show that in each iteration of the algorithm, matrix entries stay within a bounded degree.

Theorem 6.5.2. *Given* $\deg(p) \geq \deg(A)$*, it holds that* $\deg(B_i) < \deg(p_i)$*,* $\deg(R_i) < \deg(A)$ *and* $A^{-1} = B_i + A^{-1} R_i p_i$ *for all i,* $0 \leq i \leq k$*.*

Proof. Using induction on i, we will prove that the following identities and bounds hold.

$$A^{-1} = B_i + A^{-1} R_i p_i \tag{6.5.1}$$

$$\deg(B_i) < \deg(p_i) \tag{6.5.2}$$

where $B_i = B_{i-1}(I + R_{i-1} p_{i-1}) + M_{i-1} p_{i-1}^2$.

For $i = 0$: the identity (6.5.1) directly follows from Lemma 6.5.1(1). The inequality (6.5.2) holds for $i = 0$ from the construction that $B_0 = \pmod{(A^{-1}, p)}$.

Now we suppose that (6.5.1) and (6.5.2) hold for some i, $i \geq 0$. Applying a division free quadratic lifting step and a linear lifting step, from Lemma 6.5.1, we have

$$\begin{aligned} A^{-1} &= B_i(I + R_i p_i) + A^{-1} {R_i}^2 p_i^2 \\ A^{-1} &= B_i(I + R_i p_i) + M_i p_i^2 + A^{-1} R_{i+1} p_{i+1} \end{aligned} \tag{6.5.3}$$

which shows that (6.5.1) is satisfied for $i + 1$.

Given bounds on $\deg(B_i)$, we can compute bounds on $\deg(R_i)$ using (6.5.1).

$$\begin{aligned} R_i &= (1/p_i)(I - AB_i) \\ \deg(R_i) &\leq \max\{\deg(I), \deg(A) + \deg(B_i)\} - \deg(p_i) \\ &< \deg(A) + \deg(p_i) - \deg(p_i) \\ &= \deg(A). \end{aligned}$$

The computed bound on R_i can be used to prove (6.5.2) for $i + 1$. We have that

$$\begin{aligned} \deg(B_i(I + R_i p_i)) &= \deg(B_i) + \max\{\deg(I), \deg(R_i) + \deg(p_i)\} \\ &< 2\deg(p_i) + \deg(A) \end{aligned} \tag{6.5.4}$$

Then, we apply computed bounds on expansions of B_{i+1} as follows:

$$\begin{aligned} B_{i+1}/p_{i+1} &= (B_i(I + R_i p_i) + M_i p_i^2)/p_i^2 p \\ \deg\left(B_{i+1}/p_{i+1}\right) &< \max\{2\deg(p_i) + \deg(A), 2\deg(p_i) + \deg(p)\} - 2\deg(p_i) + \deg(p) \\ &\leq 0 \end{aligned}$$

The last two inequalities follows due to (6.5.4) and $\deg(p) \geq \deg(A)$. This shows that (6.5.2) holds for $i + 1$, and similar to the case i the bound on R_{i+1} can be shown by induction.

□

Theorem 6.5.3. *Algorithm 29 is correct. If k is chosen large enough to satisfy $2^{k+1} \geq n$, then A^{-1} is integral if and only if R_k is the zero matrix.*

Proof. Algorithm 29 tests the integrality of A^{-1} by checking whether it has a finite p-adic expansion. Assume A is unimodular, from Corollary 6.4.1, we have a bound on the degree of the determinant in terms of the maximum degree of the entries of A. Therefore, considering A^{-1} as a solution matrix of the linear system $Ax = I$, we can obtain a bound on the degrees of matrix entries of A^{-1} as

$$\deg(A^{-1}) \leq (n-1)\deg(A) \tag{6.5.5}$$

Now, we will show how to get the bound on number of lifting steps k, to certify the integrality of A^{-1}. Suppose R_k is zero for the chosen k. Then from (6.5.1) we have $A^{-1} = B_k$, and it follows that A^{-1} is integral. In (6.5.7) we will show that the bound on $\deg(A^{-1})$, with respect to the chosen k satisfies (6.5.2) such that, $\deg(A^{-1}) = \deg(B_k) \leq \deg(p_k)$. In order to obtain this bound on A^{-1}, we choose the number of iterations k such that (6.5.6) holds.

$$(n-2)\deg(A) \leq 2\deg(p_{k-1}) \tag{6.5.6}$$

From (6.5.6) and (6.5.5) we have that:

$$\begin{aligned} \deg(A^{-1}) &\leq (n-1)\deg(A) \\ &\leq 2\deg(p_{k-1}) + \deg(A) \\ &\leq 2\deg(p_{k-1}) + \deg(p) \\ &= \deg(p_{k-1}^2 p) \\ &= \deg(p_k) \end{aligned} \tag{6.5.7}$$

Now, we rephase (6.5.6) to choose the number of steps k easily (as given in the Step 2 in Algorithm 29).

$$\begin{aligned} (n-2)\deg(A) &\leq 2\deg(p_{k-1}) \\ &= 2\deg(p^{2^k-1}) \\ &= 2(2^k-1)\deg(p) \\ (n-2)\deg(A) + 2\deg(p) &\leq 2^{k+1}\deg(p) \end{aligned}$$

Since $\deg(A) \leq \deg(p)$, we choose k such that $(n-2)\deg(A) + 2\deg(p) \leq n\deg(p) \leq 2^{k+1}\deg(p)$. That is $n/2 \leq 2^k$, and we can take $k = \lceil \log_2(n/2) \rceil$.

The quadratic lifting step at the $k-1$-th iteration of the double-plus-one gives that $B_{k-1}(I+R_{k-1}p_{k-1}) \cong A^{-1} \pmod{p_{k-1}^2}$. Equivalently, for some error term E we have

$$A^{-1} = B_{k-1}(I + R_{k-1}p_{k-1}) + Ep_{k-1}^2 \tag{6.5.8}$$

Now we can get a bound on the size of the error term using presumed size of p_{k-1}^2 and bounds on other terms.

$$\begin{aligned}
E &= (A^{-1} - (B_{k-1}(I + R_{k-1}p_{k-1}))/p_{k-1}^2 \\
\deg(E) &= \deg(A^{-1} - (B_{k-1}(I + R_{k-1}p_{k-1})) - \deg(p_{k-1}^2) \\
&= \max\{\deg(A^{-1}),\, \deg(B_{k-1}(I + R_{k-1}p_{k-1})\} - 2\deg(p_{k-1}) \\
&< \max\{(n-1)\deg(A) - 2\deg(p_{k-1}),\, \deg(A)\} && \text{from (6.5.4) and (6.5.5)} \\
&\leq \deg(A) && \text{from (6.5.6)} \\
&\leq \deg(p) && \text{by assumption}
\end{aligned}$$

Considering the quadratic lifting step (6.5.3) for $i = k-1$, we can relate the residue R_{k-1} of (6.5.3) to the error E of (6.5.8) and obtain:

$$A^{-1}R_{k-1}P_{k-1} = E. \tag{6.5.9}$$

Using linear p-adic lifting, the last step of the double-plus one lifting computes the correction $M_{k-1} = A^{-1}R_{k-1}^2$ which is also equal to $E \pmod{p}$ by (6.5.9) (since the size of the error satisfies $\deg(E) < \deg(p)$, we get $E \pmod{p} = E$). Hence M_{k-1} exactly captures the error as $M_{k-1} = E$.

Finally, we have that

$$\begin{aligned}
R_k &= (1/p)(R_{k-1}^2 - AM_{k-1}) && \text{by Lemma 6.5.1} \\
&= (1/p)(R_{k-1}^2 - AE) && \text{as } M_{k-1} = E \\
&= 0 && \text{by equation (6.5.9).}
\end{aligned}$$

□

6.6 Algorithms for Determinant Computation

6.6.1 Determinant Computation Using CRT

Algorithm 30 Determinant Computation (`DetCRT`)

Input: $A \in \mathbb{F}[x]^{n\times n}$.
Output: Determinant of A.

1: Computes a bound B on the degree of $\det(A)$ using Theorem 6.4.1.
2: Choose t matching irreducible relatively prime polynomials $p_1, p_2, \ldots, p_t \in \mathbb{F}[x]$ such that $\sum_{i=1}^{t} \deg(p_i) \geq B$.
3: **for** $i = 1, \ldots, t$ **do**
4: $\quad A_{p_i} = A \pmod{p_i}$
5: $\quad$ Compute $d_{p_i} = \det(A_{p_i})$.
6: **end for**
7: Apply CRT to compute d_A such that $d_{p_i} \cong d_A \pmod{p_i}$ for $i = 1, \ldots, t$.
8: **return** d_A

Here, we present the details of our implementation which uses modular methods for computing the determinant of a matrix $A \in \mathbb{F}[x]^{n\times n}$. First, the algorithm computes the bound on the degree of the determinant of A. Since the bound is quite sharp, we can use the CRT-tree method without using CRT-iterative approach, with stability or probabilistic conditions. Algorithm 30 explains the CRT-modular approach in details.

Theorem 6.6.1. *Algorithm 30 is correct.*

Proof. The correctness follows from the construction, as we have chosen, $\sum_{i=1}^{t} \deg(p_i) \geq B$, CRT gives the correct output. For $\det(A_{p_i})$ computation in the Step 5, we can choose any fast algorithm. □

Complexity and Performance Analysis

The complexity of one determinant computation modulo an irreducible polynomial p is approximately n^{ω}. The number of CRT steps required to achieve is $t \approx \deg(\det(A))/\deg(p) \approx n$, where $\deg(p)$ is the average degree of irreducible polynomials p_i. Therefore the determinant computation using CRT approach takes $O(n^{\omega+1})$ field operations, and this is not a good approach. Even though, the CRT-tree method is faster than the CRT-iterative method, the complexity gain from this step is negligible due to the expensive determinant modulo p_i computation, for many steps.

6.6.2 Extension to Storjohann's Determinant Algorithm

In this section, we presents a determinant computation algorithm for matrices over $\mathbb{F}[x]$, using Storjohann's determinant computation algorithm in [PS13]. Theoretically, and practically we can extend his method over $\mathbb{F}[x]$, and obtain similar complexities as the method given by Zhou and Lebahn.

Algorithm 31 Minimal Triangular Denominator (`PolyHcol`)

Input: $S \in \mathbb{F}[x]^{n\times n}$ and $d \in \mathbb{F}[x]$.
Output: Minimal triangular denominator H of S.

1: $s = dS$
2: $w = s$, $g = d$, $H = O_{n\times n}$ and $t = O_{n\times 1}$.
3: **for** $i = n, n-1, \ldots, 1$ **do**
4: Compute $(g_i, *, t_i)$ such that $g_i = \gcd(g, s_i)$ and $g_i = *g + t_i s_i$
5: $H_{ii} = g/g_i$ and $g = g_i$
6: **end for**
7: **for** $i = 1, 2, \ldots, n$ **do**
8: **if** $H_{ii} = 1$ **then**
9: **continue**
10: **else**
11: **for** $j = 1 : i-1$ **do**
12: $H_{ji} = -t_i s_j \pmod{H_{ii}}$
13: $s_j = s_j + H_{ji} s_i \pmod{d}$
14: **end for**
15: $s_i = H_{ii} s_i$
16: $d = d/H_{ii}$ and $s = s/H_{ii}$
17: **end if**
18: **end for**
19: **return** H

Minimal Triangular Denominator

In Storjohann's algorithm, he uses the solution of some linear system $Ax = b$ to obtain a minimal triangular denominator T such that AT^{-1} will be a equivalent to the matrix $[A \mid b]^t$ as a module generated by row vectors. In our case, the same theory can be applied as modules over $\mathbb{F}[x]$ are free, and matrix operation preserves module properties.

Definition 6.6.2. *Given $s \in \mathbb{F}(x)^{n\times 1}$ a minimal triangular denominator of s is a non-singular upper triangular matrix $T \in \mathbb{F}[x]^{n\times n}$ with minimal degree determinant such that Ts is integral.*

Since, $\mathbb{F}[x]$ is a Euclidean domain, the construction of a minimal triangular denominator, and its proof are exactly follows as in [PS13, Section 2] and [Pau13, Theorem 14] for the integer case. Let $s \in \mathbb{F}(x)^{n\times 1}$ and non-zero $d \in \mathbb{F}[x]$ be such that $v = ds$ is integral. We present Algorithm 31: `PolyHcol`(s, d) which computes a minimal triangular denominator of s. This will be used in the next section.

Determinant Computation

Algorithm 32 Determinant Computation (`DeterminantPoly`)

Input: $A \in \mathbb{F}[x]^{n\times n}$.
Output: Determinant of A.

1: $d = 1$ and $B = A$.
2: $f =$ `UniCertPolyMat`(B)
3: **while** $f =$ true **do**
4: Choose a matrix $b \in F[x]^{n\times 1}$ with degrees of the entries chosen uniformly randomly from a suitable interval $J \subset \mathbb{Z}_{\geq 0}$.
5: $S, d_s =$ `SolverPolyMat`(A, b)
6: $T =$ `PolyHcol`(S, d_s)
7: $B = BT^{-1}$
8: $d = d \cdot \det(T)$
9: $f =$ `UniCertPolyMat`(B)
10: **end while**
11: $B_x = B \pmod{x}$
12: **return** $d \cdot \det(B_x)$

The Algorithm 32 gives a determinant computation algorithm for a non-singular matrix A over $\mathbb{F}[x]$, extending the integer determinant computation algorithm given in [PS13, Figure 2]. The first phase computes $S = A^{-1}b$ for a vector $b \in \mathbb{F}[x]^n$ in which the entries chosen uniformly randomly from $J \subset \mathbb{Z}_{>0}$. If the degree range of the matrix A is known, we choose the same range for the interval J to obtain similar bounds for numerator and denominator of S. Then the minimal triangular denominator captures a large divisor of $\det(A)$, and the determinant of the updated AT^{-1} is reduced by a factor of $\det(A)$. In each succession minimal triangular denominators say $T_1, T_2, \ldots$ are computed, and unimodular certification of $B = AT_1^{-1}T_2^{-1}$ verifies the completeness of the determinant of A up to a unit. For generic matrices, we only need a single iteration to extract the determinant, similar to the cases of integers (in [PS13, Section 3]) and number fields. Finally, the determinant of A can be obtained from the product of determinants of T and $B \pmod{x}$ as explained in the Theorem 6.6.3. These det computations are efficient as $B \pmod{x}$ is a integer matrix and T is an upper triangular matrix. Also, we can use the fast inverse computation method of Algorithm 13 at the Step 7 to compute T^{-1}.

Theorem 6.6.3. *Algorithm 32 is correct.*

Proof. The correctedness of the sub-algorithm follows from previous sections and [Pau13]. Suppose that B is unimodular at the iteration k, with $B = AT^{-1}$ where $T^{-1} = T_1^{-1}T_2^{-1}\cdots T_k^{-1}$. Then $\det(A) = \det(B)\det(T)$, and $\det(B) = \det(B \pmod{x})$ by Lemma 6.2.1. □

Example 6.6.4. *Consider the matrix* $A = \begin{bmatrix} 3x^2+6x+5 & 5*x^3+x^2+x \\ 5x^2+4x & 3x \end{bmatrix}$ *over* $\mathbb{F}_7[x]$. *The entries of the matrix b are chosen randomly in* $\mathbb{F}_7[x]^2$ *as:* $b = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$. *Now we solve the linear system* $Ax = b$ *and compute the denominator of the solution using Dixon's algorithm and vector reconstruction.*

$S, \mathrm{den}(S) = \texttt{SolverPolyMat}(A, b)$ *where* $S = \begin{bmatrix} (5x^2+x+2)/(x^4+x^3+5) \\ (2x+2)/(x^5+x^4+5x \end{bmatrix}$ *and*

$\mathrm{den}(S) = 4x^5+4x^4+6x$.

We compute the minimal triangular denominator as $T = \texttt{PolyHcol}(S, \mathrm{den}(S))$*:*

$T = \begin{bmatrix} 1 & 2x^4+x^2+2x \\ 0 & 4x^5+4x^4+6x \end{bmatrix}$

$T^{-1} = \frac{1}{4x^5+4x^4+6x}\begin{bmatrix} 4x^5+4x^4+6x & 5x^4+6x^2+5x \\ 0 & 1 \end{bmatrix}$

Step 7 of Algorithm 32 computes: $C = AT^{-1} = \begin{bmatrix} 3x^2+6x+5 & 2x+2 \\ 5x^2+4x & x+4 \end{bmatrix}$

Now we test for unimodularity of C using Algorithm 29:

First, we choose an irreducible polynomial $p \in \mathbb{F}_7[x]$ *such that* $\deg(p) \geq \deg(C)$. *Let* $p = x^2+4x+6$. *We compute* $C^{-1} \pmod{p}$ *over the field* $\mathbb{F}_7[x]/(p)$ *as:*

$B_0 = C^{-1} \pmod{p} = \begin{bmatrix} 6x+3 & 2x+2 \\ 5x+5 & 6x+6 \end{bmatrix}$.

Take $M_0 = B_0$, $R_0 = I$, *then* $T_0 = I^2$. *Compute* $R_1 = (T_0 - CM_0)/p = \begin{bmatrix} 3x+3 & x+1 \\ 5x+6 & 4x+2 \end{bmatrix}$.

$T_1 = R_1^2 = \begin{bmatrix} x+1 & 5x+5 \\ 4x+2 & 6x+3 \end{bmatrix}$.

$M_1 = B_0T_1 \pmod{p} = \begin{bmatrix} 0 & 0 \\ 4 & 6 \end{bmatrix}$. *Then, we compute* $R_2 = (T_1 - CM_1)/p = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. *Since* $R_2 = O_{2\times 2}$, *we have that* $\texttt{UniCertPolyMat}(C) = \text{true}$.

Now we compute the determinant of A as follows:

Let $C_x = C \pmod{x} = \begin{bmatrix} 5 & 2 \\ 0 & 4 \end{bmatrix}$. *It is easy to compute determinants of T and* C_x, *since T is an upper-triangular matrix and* C_x *is a matrix over* $\mathbb{F}_7$. *Therefore, we have:* $\det(A) = \det(C_x)\det(T) = 6(4x^5+4x^4+6x) = 3x^5+3x^4+x$.

Complexity Analysis

1. The complexity of solving step contains:
   (a) one inverse computation $A^{-1} \pmod{p}$ with complexity $O(n^\omega M(\deg(A)))$, assuming that $\deg(p) \leq \deg(A)$.
   (b) $\lceil (n\deg(A)/\deg(p)) \rceil$ lifting steps, with cost $O(n^2 M(\deg(A)))$.
2. `PolyHcol` takes $O(n\deg(A)^2)$ for n-times extended-gcd computations and product of polynomials.
3. `UniCertPolyMat` step has the complexity $O(n^\omega \log(n/2))$ as number of steps $\lceil \log_2(n/2) \rceil$.

Therefore, we can provide an upper bound for the complexity over $\mathbb{F}$ as $O(n^\omega \log(n) \deg(A))$, which is similar to the complexity of Zhou and Labahn approach in Section 6.2.2. The advantage here is that, we work with small degrees, not over the full field with full degrees. Neiger and Pernet use an unusual recursion scheme, their complexity is dropped by a factor of $\log(n)$, we expect better performance in Storjohann's approach, for generic matrices, as we get the required result at the first round, and we need `UniCertPolyMat` only for the verification of the completeness.

$\mathbb{F}_q$	d / n	10-15			100-150			500-550		
		`Det_P`	Hecke	Quot	`Det_P`	Hecke	Quot	`Det_P`	Hecke	Quot
$\mathbb{F}_7$	10	0.15	0.00	0.02	0.59	0.15	0.26	13.84	2.75	0.20
	30	0.45	0.07	0.16	3.62	4.23	1.17	22.90	61.09	2.67
	50	1.87	0.33	0.18	16.41	22.18	1.35	1.7 m	4.7 m	2.69
	100	15.57	2.91	0.19	1.9 m	3.3 m	1.64	11.7 m	37.9 m	3.23
	150	1 m	11.76	0.20	6.7 m	10.9 m	1.63	33.2 m	1.9 h	3.51
$\mathbb{F}_{3511}$	10	0.03	0.00	0.17	1.21	0.17	0.14	21.27	2.26	0.11
	30	0.55	0.08	0.15	6.17	4.91	0.80	51.45	1.2 m	1.35
	50	2.17	0.37	0.17	29.04	26.46	0.91	2.9 m	5.5 m	1.86
	100	16.80	3.28	0.20	3.4 m	4 m	1.14	20.5 m	43.8 m	2.13
	150	1.2 m	13.63	0.19	11.9 m	12.5 m	1.05	1.1 h	2.6 h	2.42
$\mathbb{F}_{q_3}$	10	0.05	0.00	0.06	5.45	0.12	0.02	1.1 m	1.48	0.02
	30	0.88	0.07	0.08	17.44	3.09	0.18	2.1 m	42.48	0.33
	50	4.04	0.33	0.08	1.4 m	17.60	0.20	8.0 m	3.3 m	0.41
	100	33.39	2.99	0.09	10.7 m	2.4 m	0.22	1.1 h	25.6 m	0.42
	150	2 m	13.71	0.11	36.4 m	8.2 m	0.22	3.5 h	2.8 h	0.81

Table 6.1: Timing for Polynomial Determinant Computation with Proof.

Performance Analysis Based on Timings

We have implemented Algorithm 32 using the Hecke, comparison have been made with the existing implementation in Hecke which is based on Mulders' and Storjohann's method as explained in Section 6.2.1. Table 6.1 shows the run-timings for Algorithm 32 `DeterminantPoly` (`Det_P`) with the proof for the determinant (using unimodular certification). However, for random matrices denominator is the determinant, and we do not require to test the result using `UniCert` step. Thus, Table 6.2 shows runtimes without the proof in columns `Det`.

To illustrate the efficiency of the new method, we have computed timings over finite fields: $\mathbb{F}_7$, $\mathbb{F}_{3511}$ and $\mathbb{F}_{q_3}$ for $q_3 = 351135113521$. Table 6.1 and 6.2 show times in seconds (and m -for minutes) for new algorithms `Det_P` and `Det`, (and quotient times in columns Quot). We have considered different choices of parameters: n- dimension of the matrix, d- degree range of the input matrix and $\mathbb{F}_q$- the finite field. Closer inspection of the timings show that the new algorithm works much better for big matrices having large degree entries, over fields of small characteristics: over $\mathbb{F}_7$ for 150×150 matrix having entries of degree $500 - 550$ the new implementation computes the determinant, nearly 3 times faster than the existing one with the unimodular certification, and nearly 10 times faster without the proof.

$\mathbb{F}_q$	d / n	10-15			100-150			500-550		
		Det	Hecke	Quot	Det	Hecke	Quot	Det	Hecke	Quot
$\mathbb{F}_7$	10	0.14	0.00	0.02	0.36	0.15	0.41	7.09	2.75	0.39
	30	0.22	0.07	0.33	2.12	4.23	2.00	14.26	61.09	4.28
	50	0.77	0.33	0.44	8.03	22.18	2.76	48.13	4.7 m	5.90
	100	5.76	2.99	0.52	47.55	3.3 m	4.12	4.7 m	37.9 m	7.91
	150	17.11	13.71	0.80	2.3 m	10.9 m	4.64	11.5 m	1.9 h	9.89
$\mathbb{F}_{3511}$	10	0.02	0.00	0.30	0.98	0.17	0.17	9.21	2.26	0.25
	30	0.28	0.08	0.30	3.29	4.91	1.49	24.44	1.2 m	2.85
	50	0.91	0.37	0.41	12.69	26.46	2.09	1.4 m	5.5 m	3.72
	100	6.21	3.28	0.53	1.3 m	4 m	2.99	8.8 m	43.8 m	5.17
	150	20.13	13.63	0.68	4 m	12.5 m	3.06	21.6 m	2.6 h	7.21
$\mathbb{F}_{q_3}$	10	0.03	0.00	0.10	1.78	0.12	0.07	54.58	1.48	0.03
	30	0.40	0.07	0.18	8.37	3.09	0.37	1.3 m	42.48	0.54
	50	1.60	0.33	0.21	29.97	17.60	0.59	3.6 m	3.3 m	0.91
	100	11.51	2.99	0.26	3.6 m	2.4 m	0.65	25.2 m	25.6 m	1.01
	150	36.70	13.71	0.37	11.5 m	8.2 m	0.71	1.3 h	2.8 h	2.13

Table 6.2: Timing for Polynomial Determinant Computation without Proof.

Conclusions and Further Work

The implemented determinant computation Algorithm 6 (`ModularDeterminant`), which uses the idea of Abbott, Bronstein and Mulders performs superior to all existing implementations. The new unimodular certification algorithm uses higher-order lifting and residue number systems. Also, the normal form computation of a one-dimensional module using `PseudoHcol` implementation is efficient and successful (generalizing to the `hcol` algorithm by Storjohann). The output of the `PseudoHcol` algorithm stays balanced with the size of entries similar to the input matrix, this phenomenon has been observed in the integer case as well, but we do not know the reason why it does not grow as suggested by the preliminary analysis in (pauderis2013computing).

We have been able to extend the techniques which are used in the state of the art method for determinant computation in [PS13] to number fields. We could achieve theoretically faster methods with better complexities. For further improvement we need the access to a highly optimized low-level libraries such as such as BLAS [BPP+02], ATLAS [WD98] or OpenBLAS.

Over number fields of small degrees, we could recover the determinant from the denominator ideal using the unit group. This approach for recovering the missing unit is similar to the final step of the determinant algorithm for integers.

We could optimize the determinant computation algorithm over polynomial ring over finite fields, by using residue number systems in the unimodular certification step.

The minimal polynomial can be computed in the time of matrix multiplications, using coprime basis computation and Hensel lifting.

Bibliography

[ABM99] John Abbott, Manuel Bronstein, and Thom Mulders. Fast deterministic computation of determinants of dense matrices. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 197–204, New York, NY, USA, 1999. Association for Computing Machinery.

[AM01] John Abbott and Thorn Mulders. How tight is Hadamard's bound? *Experimental Mathematics*, 10(3):331–336, 2001.

[BDFP15] Janko Böhm, Wolfram Decker, Claus Fieker, and Gerhard Pfister. The use of bad primes in rational reconstruction. *Mathematics of Computation*, 84(296):3013–3027, 2015.

[BDS90] Eric Bach, James R. Driscoll, and Jeffrey O. Shallit. Factor refinement. In David S. Johnson, editor, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA*, pages 201–211. SIAM, 1990.

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[Bel04] Karim Belabas. A relative van Hoeij algorithm over number fields. *Journal of Symbolic Computation*, 37(5):641–668, 2004.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–735, 1970.

[Ber84] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, March 1984.

[Ber05] Daniel J. Bernstein. Factoring into coprimes in essentially linear time. *Journal of Algorithms*, 54(1):1–30, 2005.

[BF12] Jean-François Biasse and Claus Fieker. A polynomial time algorithm for computing the hnf of a module over the integers of a number field. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 75–82, New York, NY, USA, 2012. Association for Computing Machinery.

[BFH17] Jean-François Biasse, Claus Fieker, and Tommy Hofmann. On the computation of the hnf of a module over the ring of integers of a number field. *Journal of Symbolic Computation*, 80:581–615, May 2017.

[BL94] Bernhard Beckermann and George Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, 15(3):804–823, 1994.

[BLV99] Bernhard Beckermann, George Labahn, and Gilles Villard. Shifted normal forms of polynomial matrices. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, ISSAC '99, pages 189–196, New York, NY, USA, 1999. Association for Computing Machinery.

[BP91] Wieb Bosma and Michael Pohst. Computations with finitely generated modules over dedekind rings. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, ISSAC '91, pages 151–156, New York, NY, USA, 1991. Association for Computing Machinery.

[BPP+02] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[BS11] Curtis Bright and Arne Storjohann. Vector rational number reconstruction. In *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, pages 51–58. ACM, 2011.

[CL77] S. Cabay and T. P. L. Lam. Congruence techniques for the exact solution of integer systems of linear equations. *ACM Trans. Math. Softw.*, 3(4):386–397, December 1977.

[Coh96] Henri Cohen. Hermite and Smith normal form algorithms over Dedekind domains. *Mathematics of computation*, 65(216):1681–1699, 1996.

[Coh12] Henri Cohen. *Advanced topics in computational number theory*, volume 193. Springer Science & Business Media, 2012.

[Coh13] Henri Cohen. *A course in computational algebraic number theory*, volume 138. Springer Science & Business Media, 2013.

[CZ81] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[Dan37] AM Danilevskii. On the numerical solution of the secular equation. *Matem. Sbornik*, 44(2):169–172, 1937.

[Dix82] John D Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40(1):137–141, 1982.

[Dum06] Jean-Guillaume Dumas. Bounds on the coefficients of the characteristic and minimal polynomials. working paper or preprint, 2006.

[FF63] DK Faddeev and VN Fadeeva. Computational methods of linear algebra Freeman. *San Francisco*, 1963.

[FF00] Claus Fieker and Carsten Friedrichs. On reconstruction of algebraic numbers. In *International Algorithmic Number Theory Symposium*, pages 285–296. Springer, 2000.

[FHHJ17] Claus Fieker, William Hart, Tommy Hofmann, and Fredrik Johansson. Nemo/Hecke: Computer algebra and number theory packages for the Julia programming language. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 157–164, New York, NY, USA, 2017. ACM.

[Gan60] Felix R Gantmacher. The theory of matrices. volume one. *Translated by KA Hirsch, Chelsea Publishing Company, Printed in USA, Card Nr. 59-11779, ISBN: 8284-0131-4*, 1960.

[Ger31] Semyon Aranovich Gershgorin. Uber die Abgrenzung der Eigenwerte einer Matrix. *Izvestiya Rossiyskoy akademii nauk. Seriya matematicheskaya*, (6):749–754, 1931.

[GG13] Joachim von zur Gathen and Jrgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, USA, 3rd edition, 2013.

[GJV03] Pascal Giorgi, Claude-Pierre Jeannerod, and Gilles Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 135–142, New York, NY, USA, 2003. Association for Computing Machinery.

[GS02] Mark Giesbrecht and Arne Storjohann. Computing rational forms of integer matrices. *Journal of Symbolic Computation*, 34(3):157–172, 2002.

[GSSV12] Somit Gupta, Soumojit Sarkar, Arne Storjohann, and Johnny Valeriote. Triangular x-basis decompositions and derandomization of linear algebra algorithms over k[x]. *Journal of Symbolic Computation*, 47(4):422 – 453, 2012. Special Issue for Joachim von zur Gathen at 60.

[Hes42] Karl Hessenberg. *Die Berechnung der Eigenwerte und Eigenlösungen linearer Gleichungssysteme*. PhD thesis, 1942.

[HK71] Kenneth Hoffman and Ray Kunze. Linear algebra, Prentice-Hall. *Englewood Cliffs, New Jersey*, 1971.

[Hop98] Andreas Hoppe. *Normal forms over Dedekind domains, efficient implementation in the computer algebra system KANT*. PhD thesis, Technischen Universität Berlin, 1998.

[JH85] Charles R Johnson and Roger A Horn. *Matrix analysis*. Cambridge university press Cambridge, 1985.

[Kal92] Erich Kaltofen. On computing determinants of matrices without divisions. In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '92, pages 342–349, New York, NY, USA, 1992. Association for Computing Machinery.

[KG85] Walter Keller-Gehrig. Fast algorithms for the characteristics polynomial. *Theoretical computer science*, 36:309–317, 1985.

[KM06] Sara Khodadad and Michael Monagan. Fast rational function reconstruction. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*, ISSAC '06, pages 184–190, New York, NY, USA, 2006. Association for Computing Machinery.

[Lan05] Edmund Landau. Sur quelques théorèmes de M. Petrovitch relatifs aux zéros des fonctions analytiques. *Bulletin de la Société Mathématique de France*, 33:251–261, 1905.

[Lan94] Serge Lang. *Algebraic Number Theory*. Springer New York, New York, NY, 2nd ed. 1994 edition, 1994.

[Lan02] Serge Lang. *Algebra*. Springer New York, New York, NY, 3rd ed. 2002 edition, 2002.

[Len82] Arjen Klaas Lenstra. Lattices and factorization of polynomials over algebraic number fields. In *European Computer Algebra Conference*, pages 32–39. Springer, 1982.

[Lev40] UJJ Leverrier. Sur les variations séculaire des élements des orbites pour les sept planétes principales. *J. de Math*, (s 1):220 – 254, 1840.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LMW05] Simon Lo, Michael Monagan, and Allan Wittkopf. A modular algorithm for computing the characteristic polynomial of an integer matrix in maple. In *Maple Summer Workshop*, 2005.

[LNZ17] George Labahn, Vincent Neiger, and Wei Zhou. Fast, deterministic computation of the hermite normal form and determinant of a polynomial matrix. *Journal of Complexity*, 42:44 – 71, 2017.

[Maz08] Marc Moreno Maza. Lecture notes advanced computer algebra: The resultant and gcd computation. *Available at* `https://www.csd.uwo.ca/~mmorenom/CS424/Lectures/ResultantGcd.html/ResultantGcd.html` Last visited on 2020/12/20", January 2008.

[Mig92] Maurice Mignotte. *Mathematics for Computer Algebra*. Springer-Verlag, Berlin, Heidelberg, 1992.

[MS03] T. Mulders and A. Storjohann. On lattice reduction for polynomial matrices. *Journal of Symbolic Computation*, 35(4):377 – 401, 2003.

[Neu99] Jürgen Neukirch. *Algebraic Number Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1st ed. 1999 edition, 1999.

[NP20] Vincent Neiger and Clément Pernet. Deterministic computation of the characteristic polynomial in the time of matrix multiplication. *CoRR*, abs/2010.04662, 2020.

[NRS18] Vincent Neiger, Johan Rosenkilde, and Grigory Solomatov. Computing Popov and Hermite forms of rectangular polynomial matrices. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, ISSAC '18, pages 295–302, New York, NY, USA, 2018. Association for Computing Machinery.

[NS09] Phong Q. Nguyen and Damien Stehlé. An lll algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[O'M00] Onorato Timothy O'Meara. *Introduction to quadratic forms*. Springer, 2000.

[OP07] Amos R Omondi and Benjamin Premkumar. *Residue number systems: theory and implementation*, volume 2. World Scientific, 2007.

[OS07] Zach Olesh and Arne Storjohann. The vector rational function reconstruction problem. In *Computer Algebra 2006: Latest Advances in Symbolic Algorithms*, pages 137–149. World Scientific, 2007.

[Pau13] Colton Pauderis. Deterministic unimodularity certification and applications for integer matrices. Master's thesis, University of Waterloo, 2013.

[Poh05] Michael E. Pohst. Factoring polynomials over global fields I. *Journal of Symbolic Computation*, 39(6):617 – 630, 2005.

[Pos] Sebastian Posur. personal communication.

[PP93] Karl C Posch and Reinhard Posch. Base extension using a convolution sum in residue number systems. *Computing*, 50(2):93–104, 1993.

[PS07] Clément Pernet and Arne Storjohann. Faster algorithms for the characteristic polynomial. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 307–314, New York, NY, USA, 2007. Association for Computing Machinery.

[PS12] Colton Pauderis and Arne Storjohann. Deterministic unimodularity certification. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 281–288. ACM, 2012.

[PS13] Colton Pauderis and Arne Storjohann. Computing the invariant structure of integer matrices: fast algorithms into practice. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 307–314. ACM, 2013.

[PZ97] Michael Pohst and Hans Zassenhaus. *Algorithmic algebraic number theory*, volume 30. Cambridge University Press, 1997.

[RS21] Johan Rosenkilde and Arne Storjohann. Algorithms for simultaneous Hermite-Padé approximations. *Journal of Symbolic Computation*, 102:279–303, 2021.

[SK89] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in rns. *IEEE Trans. Comput.*, 38(2):292–297, February 1989.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7(3-4):281–292, 1971.

[SS11] Soumojit Sarkar and Arne Storjohann. Normalization of row reduced matrices. In *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, ISSAC '11, pages 297–304, New York, NY, USA, 2011. Association for Computing Machinery.

[Ste97] Allan Steel. A new algorithm for the computation of canonical forms of matrices over fields. *Journal of Symbolic Computation*, 24(3):409 – 432, 1997.

[Sto00] Arne Storjohann. *Algorithms for matrix canonical forms*. PhD thesis, Department of Computer Science, Swiss Federal Institute of Technology – ETH, 2000.

[Sto03] Arne Storjohann. High-order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3-4):613–648, 2003.

[Sto07] Michael Stoll. Course no.100 221, lecture notes of courses taught at Jacobs University: Linear Algebra I. *Available at* `http://www.mathe2.uni-bayreuth.de/stoll/lecture-notes/LinearAlgebraI.pdf`. Last visited on 2019/10/24, December 2007.

[Sur21] Jayantha Suranimalee. Github: Linear-algebra-over-finitely-generated-fields-and-rings. *Available at* `https://github.com/suranimalee/Linear-Algebra-over-Finitely-Generated-Fields-and-Rings`, 2021.

[Var04] Richard S Varga. *Geršgorin and his circles*, volume 36 of Springer Series in Computational Mathematics. Springer, Berlin, Heidelberg, New York, 2004.

[vHN10] Mark van Hoeij and Andrew Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. In Alejandro López-Ortiz, editor, *LATIN 2010: Theoretical Informatics*, pages 539–553, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Vin03] Ėrnest Borisovich Vinberg. *A course in algebra*. Number 56. American Mathematical Soc., 2003.

[WD98] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–27, USA, 1998. IEEE Computer Society.

[Zho13] Zhou, Wei. *Fast Order Basis and Kernel Basis Computation and Related Problems*. PhD thesis, University of Waterloo, 2013.

[ZL12] Wei Zhou and George Labahn. Efficient algorithms for order basis computation. *Journal of Symbolic Computation*, 47(7):793 – 819, 2012. International Symposium on Symbolic and Algebraic Computation (ISSAC 2009).

[ZL13] Wei Zhou and George Labahn. Computing column bases of polynomial matrices. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 379–386, New York, NY, USA, 2013. Association for Computing Machinery.

[ZL14] Wei Zhou and George Labahn. Fast and deterministic computation of the determinant of a polynomial matrix. *CoRR*, abs/1409.5462, 2014.

[ZLS12] Wei Zhou, George Labahn, and Arne Storjohann. Computing minimal nullspace bases. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 366–373, New York, NY, USA, 2012. Association for Computing Machinery.

Wissenschaftlicher Werdegang

Ausbildung

Seit 11/2017	Doktorand am Fachbereich Mathematik der TUK, Advisor: Prof. Dr. Claus Fieker
10/2015 - 09/2017	Masterstudiengang Mathematics International am Fachbereich Mathematik, TUK
07/2008 - 01/2013	Bachelor of Science in Mathematik (Spezial) - University of Colombo, Sri Lanka

Beschäftigung

Seit 01/2014	Dozentin im Department of Mathematics, University of Colombo, Sri Lanka

Curriculum Vitae

Education

11/2017 - 05/2021	Doctoral student at the Department of Mathematics, TUK, Advisor: Prof. Dr. Claus Fieker
10/2015 - 09/2017	Master of Science - Mathematics International Program at the Department of Mathematics, TUK
07/2008 - 01/2013	Bachelor of Science in Mathematics (Special) - University of Colombo, Sri Lanka

Employment

Since 01/2014	Lecturer in the Department of Mathematics, University of Colombo, Sri Lanka