Kurze Darstellung von Ordnungen

Diplomarbeit

von

Stefan Neis

nach einem Thema von

Prof. Dr. Buchmann

Fachbereich 14 – Informatik –

Universität des Saarlandes

14. Oktober 1994

Eidesstattliche Erklärung

„Hiermit versichere ich an Eides statt, daß ich diese Arbeit selbständig und nur unter Verwendung der angegebenen Quellen angefertigt habe.“

Saarbrücken, den 2. Mai 1994

Stefan Neis

An dieser Stelle möchte ich allen danken, die mich während der Entstehung dieser Diplomarbeit unterstützten.

Mein Dank gebührt zunächst Herrn Prof. Dr. Buchmann, der dieses interessante Thema vergeben und mich mit Ideen und Anregungen unterstützt hat.

Desweiteren danke ich Achim Müller für anregende Diskussionen, insbesondere über die in Kapitel 1 dargestellten Probleme der linearen Algebra.

Danken möchte ich auch Patrick Theobald für seine Unterstützung beim Korrekturlesen.

Schließlich gilt mein Dank insbesondere meinen Eltern, die mir das Studium ermöglichten und mich nach besten Kräften unterstützten.

Inhaltsverzeichnis

Bezeichnungen

Zu Beginn wollen wir wichtige Bezeichnungen, die in dieser Arbeit verwendet werden, zusammenfassen. Dabei untergliedern wir diese Aufstellung in folgende Themenbereiche:

- Zahlmengen und ihre Darstellung
- Vektoren, Matrizen, Gitter
- Polynome
- Spezielle Zahlen
- Normen
- Laufzeit– und Größenabschätzungen

Zahlmengen und ihre Darstellung

$\mathbb{N}/\mathbb{N}_0$	Menge der natürlichen Zahlen ohne/mit Null
$\mathbb{P}$	Menge der Primzahlen
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{Q}$	Menge der rationalen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}_{>0}/\mathbb{R}_{\geq 0}$	Menge der positiven reellen Zahlen ohne/mit Null
$\mathbb{C}$	Menge der komplexen Zahlen
$\mathbb{F}_p$	Körper mit p Elementen
$\mathbb{K} = \mathbb{Q}(\varrho)$	Von ϱ erzeugter algebraischer Zahlkörper
$\mathcal{O}$	Ordnung eines Zahlkörpers $\mathbb{K}$
$\mathcal{O}_{\mathbb{K}}$	Maximale Ordnung, Ganzheitsring von $\mathbb{K}$

$\Omega = (\omega_1, \ldots, \omega_n)$	Basis einer Ordnung $\mathcal{O}$ bzw. (Zeilen–)Vektor mit Zahlen der Basis als Komponenten
$\underline{\Omega} = (\underline{\omega}_1, \ldots, \underline{\omega}_n)$	Konjugiertenvektoren der Basis einer Ordnung $\mathcal{O}$ bzw. Matrix mit Konjugiertenvektoren als Spaltenvektoren
$\mathrm{MT}(\Omega) = (w_{ijk})_{1 \le i,j,k \le n}$	Multiplikationstafel einer Ordnung bzgl. der Basis Ω

Vektoren, Matrizen, Gitter

M^n	Menge der n–stelligen Vektoren mit Einträgen aus M. Vektoren werden meist durch unterstrichene Kleinbuchstaben notiert.
$\underline{e}_i$	i-ter Einheitsvektor
$\underline{a}^T$	Transponierter Vektor zu $\underline{a}$, macht aus einem Zeilen– einen Spaltenvektor.
$\mathrm{M}^{m \times n}$	Menge der $(m \times n)$–Matrizen mit Einträgen aus M. Matrizen werden meist durch Großbuchstaben notiert.
$\mathrm{GL}(n, \mathbb{K})$	Menge der regulären $(n \times n)$–Matrizen über einem Körper $\mathbb{K}$
$\mathrm{GL}(n, \mathbb{Z})$	Menge der über $\mathbb{Z}$ invertierbaren $(n \times n)$–Matrizen mit ganzzahligen Einträgen
I_n	$(n \times n)$–Einheitsmatrix
$\mathrm{ad}(A)$	adjungierte Matrix der $(n \times n)$–Matrix A
L	Gitter
$B = (\underline{b}_1, \ldots, \underline{b}_n)$	Basis eines Gitters L bzw. Matrix mit Basisvektoren als Spaltenvektoren
$\mathrm{X}\{q\}$	Approximation der Präzision q an eine Struktur (Zahl, Vektor, Gitter, ...) X

Polynome

$\deg(f)$	Grad des Polynoms f
$\mathrm{lc}(f)$	Leitkoeffizient des Polynoms f
$\mathrm{cont}(f)$	Inhalt des Polynoms f
$\mathrm{pp}(f)$	Primitiver Anteil des Polynoms f

Spezielle Zahlen

$\mathrm{ggT}(m,n)$	größter gemeinsamer Teiler von m und n
$\lceil x \rceil$	nächste obere ganze Zahl zu $x \in \mathbb{R}$
$\lfloor x \rfloor$	nächste untere ganze Zahl zu $x \in \mathbb{R}$
$\lfloor x \rceil$	nächste ganze Zahl zu $x \in \mathbb{R}$
$\log(b)$	Logarithmus von b zur Basis 2
$\Re(c)$	Realteil einer komplexen Zahl c
$\Im(c)$	Imaginärteil einer komplexen Zahl c
σ_i	Einbettungen eines Zahlkörpers $\mathbb{K}$ in $\mathbb{C}$
r_1	Zahl der reellwertigen Einbettungen von $\mathbb{K}$ in $\mathbb{C}$
$2r_2$	Zahl der echt komplexwertigen Einbettungen von $\mathbb{K}$ in $\mathbb{C}$
$\Delta_{\mathcal{O}}$	Diskriminante der Ordnung $\mathcal{O}$
$\lambda_i(A)$	Länge des i.–kleinsten Spaltenvektors einer Matrix A
$\lambda_i(L)$	i–tes sukzessives Minimum des Gitters L
$\mathrm{dft}(A)$	Defekt einer Matrix A
$\det(A)$	Determinante einer quadratischen Matrix A
$\det(L)$	Gitterdeterminante von L
$\lvert\mathrm{X}\rvert$	Betrag des betragsgrößten Eintrags in der Matrix oder Multiplikationstafel X

Normen

$\Vert\underline{a}\Vert, \Vert\underline{a}\Vert_2$	euklidische Norm eines Vektors $\underline{a}$
$\lvert\underline{a}\rvert, \Vert\underline{a}\Vert_\infty$	Maximumnorm eines Vektors $\underline{a}$
$\Vert A\Vert, \Vert A\Vert_2$	Frobeniusnorm einer Matrix A
$\Vert A\Vert_\infty$	Zeilensummennorm einer Matrix A

Laufzeit– und Größenabschätzungen

$\mathcal{F}$	$\{f \mid f$ Funktion von $\mathbb{N}$ nach $\mathbb{R}_{\geq 0}\}$
$O(f), f \in \mathcal{F}$	$\{g \in \mathcal{F} \mid \exists c \in \mathbb{R}_{>0} \exists n_0 \forall n \geq n_0 : g(n) \leq cf(n)\}$
$\Omega(f), f \in \mathcal{F}$	$\{g \in \mathcal{F} \mid \exists c \in \mathbb{R}_{>0} \exists n_0 \forall n \geq n_0 : g(n) \geq cf(n)\}$
$f = O(g)$	$f \in O(g)$

Vorwort

Laufzeiten für Algorithmen zur Berechnung zahlentheoretischer Invarianten von algebraischen Zahlkörpern werden häufig nicht in Abhängigkeit von der Eingabe angegeben, sondern nur in Abhängigkeit von der Diskriminanten. Üblicherweise sind die dabei auftretenden Laufzeiten polynomielle Ausdrücke im Grad des Zahlkörpers und im Logarithmus der Diskriminanten. Daher darf für solche Algorithmen die Eingabe auch nicht länger sein, wir müssen also beliebige Eingaben zu kurzen Darstellungen reduzieren. Im folgenden werden wir nun zeigen, wie sich diese Aufgabe lösen läßt, wobei wir uns insbesondere auf die beiden Arbeiten von J. Buchmann und V. Kessler ([BuKe90]) bzw. J. Buchmann und O. van Sprang ([BuSp92]) stützen, die sich mit diesem Thema beschäftigen.

Ziel dieser Arbeit ist es, bessere Schranken für die Größe der reduzierten Darstellung zu beweisen und den Einfluß der Rechengenauigkeit auf die Qualität des Ergebnisses zu untersuchen. Außerdem stellen wir erstmals auch eine präzise Schranke für die Laufzeit vor, nachdem die vorangehenden Arbeiten nur zeigten, daß die Probleme in polynomieller Zeit gelöst werden können. Auch diese Laufzeiten wollen wir wieder in Abhängigkeit von der Rechengenauigkeit angeben.

Zunächst wollen wir nun in Kapitel 1 die Grundlagen für die weiteren Kapitel legen, indem wir fundamentale Begriffe und Algorithmen darstellen. In Kapitel 2 werden wir dann nähere Untersuchungen über das Rechnen mit Näherungen durchführen, die wir in Kapitel 3 nutzen werden, um einen Reduktionsalgorithmus vorzustellen und zu untersuchen, der mit Näherungen arbeitet. In Kapitel 4 stellen wir dann die Algorithmen vor, mit deren Hilfe es möglich ist, die Eingabe auf kürzere Darstellungen zu reduzieren bzw. so zu transformieren, daß die Zahl 1 als Basiselement auftritt. In Kapitel 5 werden wir schließlich alle unsere Teilergebnisse zusammenfügen und das Gesamtergebnis präsentieren.

Dort werden wir z.B. beweisen, daß wir aus einer Eingabe in Form der Multiplikationstafel $\mathrm{MT}(\Omega)$ einer Ordnung $\mathcal{O}$ in Zeit $O(n^9 \log^3(n|\mathrm{MT}(\Omega)|))$ eine Multiplikationstafel $\mathrm{MT}(\Omega')$ mit Einträgen

$$|\mathrm{MT}(\Omega')| < n^{\frac{1}{2}-n} 2^{\frac{(3n+4)(n-1)+2}{4} - 2r_2 + 3n + 6} |\Delta_{\mathcal{O}}|$$

berechnen können, wenn wir mit Präzision

$$(\frac{3}{2} + \frac{5}{2}n) \log n + n \log |\mathrm{MT}(\Omega)| - \frac{1}{2} \log(|\Delta_{\mathcal{O}}|) + \frac{n+1}{2}$$

rechnen. Wir beweisen also zum einen eine exakte Schranke für die Laufzeit, zum anderen verbessern wir auch die bisher bekannte Schranke, durch die die Einträge in der reduzierten Tafel beschränkt sind. Der Artikel von J. Buchmann und V. Kessler zeigt nämlich nur, daß für die reduzierten Einträge gilt:

$$|\mathrm{MT}(\Omega')| \leq n^{(n+1)^2/2} 2^{(n+1)(n^2+1)/2 + n^2} |\Delta_{\mathcal{O}}|^{\frac{n}{2}},$$

was also in etwa der n–ten Potenz unserer Abschätzung entspricht. Um dieses schwächere Ergebnis zu erhalten, wird aber sogar noch eine größere Präzision bei den Berechnungen vorausgesetzt.

Ermöglicht werden diese Verbesserungen durch präzisere mathematische Abschätzungen einerseits, die auch das Herzstück der Betrachtungen über die Laufzeit und über die benötigte Rechengenauigkeit bilden, andererseits durch einen effizienteren Reduktionsalgorithmus, der besser mit Approximationen an reelle Zahlen umgehen kann.

Zusammen mit einer besseren Abschätzung für die Größe einer Transformationsmatrix in Kapitel 4 ergibt sich dann auch eine Verbesserung für das Hauptergebnis des Artikels von J. Buchmann und O. van Sprang. Wenn wir als Ausgabe eine Multiplikationstafel $\mathrm{MT}(\Omega'')$ erhalten wollen, so daß die Zahl 1 in der Basis Ω'' auftritt, dann können wir nun zeigen:

$$\begin{aligned} |\mathrm{MT}(\Omega'')| \;&<\; n^{-3n^3+3n^2+\frac{5n-1}{2}} \\ &\quad 2^{\left(\frac{(3n+4)(n-1)+2}{4} - 2r_2\right)(3n^2-2) + 9n^3 + 18n^2 - 6n - 12} |\Delta_{\mathcal{O}}|^{3n^2-2}, \end{aligned}$$

während bisher nur bewiesen war:

$$|\mathrm{MT}(\Omega'')| < n^{\frac{n^5+2n^4+n^3-2n^2+n+9}{2}} 2^{\frac{n^6+3n^5-n^3+2n^2+1}{2}} |\Delta_{\mathcal{O}}|^{\frac{n^4-n^2+n}{2}}.$$

Kapitel 1

Grundlagen

1.1 Komplexität von elementaren Operationen

Um sinnvoll den Begriff der *Komplexität* eines Algorithmus betrachten zu können, muß man sich zunächst darüber einigen, was man darunter versteht: Wir werden stets die Laufzeit im schlechtesten Fall untersuchen. Eine andere Möglichkeit wäre, aus Annahmen über die Verteilung der Eingaben durchschnittliche Laufzeiten der Algorithmen abzuleiten, die häufig niedriger sind. Zu dieser Unterscheidung finden sich nähere Betrachtungen in [Me84], S. 1f. Der Platzbedarf unserer Algorithmen ergibt sich stets in trivialer Weise aus den Größenabschätzungen für die beteiligten Zahlen, daher werden wir darauf im folgenden nicht mehr eingehen.

Als nächstes müssen wir entscheiden, welches *Maschinenmodell* unseren Analysen zugrunde liegt. Je komplizierter die Operationen sind, die ein Rechner in einem Schritt bewältigen kann, desto weniger Schritte wird der Rechner im allgemeinen brauchen, um aus einer Eingabe das gewünschte Ergebnis zu berechnen. Dem trägt man im allgemeinen durch Verwendung der O–Notation (vgl. [Me84], S. 37) Rechnung, in der konstante Faktoren in den Laufzeitanalysen keine Rolle spielen. Dadurch gelten die angegebenen Laufzeiten — mit jeweils unterschiedlichen Vorfaktoren — für sehr viele Rechner. Ein bekanntes Rechnermodell ist die *k–Band–Turingmaschine*, wie sie etwa in [LePa81], S. 198–204 beschrieben ist. Dieses Modell hat jedoch den Nachteil, daß der Speicher nur linear zugreifbar ist und so ein vermeidbarer Overhead durch das Suchen nach der gerade benötigten Speicherzelle entsteht. Daher wählen wir ein realitätsnäheres Modell mit der Möglichkeit des wahlfreien Speicherzugriffs. Eine formale Beschreibung eines solchen Rechnermodells findet sich etwa in [Me84], S. 1–12 (RAM, RASP) oder in [LoMeWi86], S. 236–241 (RESA). Allerdings werden wir in Abwandlung dieser Modelle nicht voraussetzen, daß beliebig lange Zahlen in eine Speicherzelle passen, wodurch wir einerseits natürlich die Speicherverwaltung komplizieren (allerdings ändert sich die Laufzeit höchstens um einen konstanten Faktor), andererseits aber realitätsnähere Voraussetzungen für das Rechnen mit langen Zahlen schaffen: Wir werden große Zahlen in „Ziffern“ aufteilen müssen, wobei eine „Ziffer“ gerade in eine Speicherzelle paßt, und dann ziffernweise die Zahlen bearbeiten. Dann können wir für elementare Operationen auf solchen Ziffern das *Einheitskostenmaß* verwenden, d.h. für solchen Ziffern, insbesondere also für kleine Zahlen wie z.B. Schleifenvariablen, kostet eine elementare Operation stets konstante Zeit. In realen Rechnern ist eine solche Ziffer derzeit meist eine Zahl, die kleiner ist als 2^{32}. Da unterschiedliche Zifferngrößen jedoch die Laufzeit nur um Konstanten ändern, können wir für unsere Betrachtungen bei langen Zahlen statt von Ziffern $< 2^{32}$ auch von Ziffern < 2 ausgehen, d.h. wir kodieren Zahlen binär, eine positive ganze Zahl u wird also dargestellt als $(u_{n-1} \ldots u_0) \in \{0,1\}^n$ mit

$$u = \sum_{i=0}^{n-1} u_i 2^i.$$

Somit hat eine positive Zahl u eine Darstellung der (binären) Länge $\lfloor \log_2 u \rfloor + 1$. Eine Zahl der binären Länge n bezeichnen wir auch als n–Bit–Zahl.

Wir untersuchen nun den Zeitbedarf einiger elementarer Operationen wie Addition und Multiplikation. Später zählen wir dann nur noch, wie viele solcher elementaren Operationen wir ausführen müssen und erhalten dann aus Abschätzungen für die Länge der beteiligten Zahlen Angaben über den Zeitbedarf unserer Algorithmen.

Bevor wir jedoch diese elementaren Algorithmen darstellen, wollen wir noch darauf hinweisen, daß wir nicht alle benutzten Algorithmen explizit formulieren werden. Häufig wird ein Algorithmus implizit in einem konstruktiven Beweis einer Laufzeitabschätzung enthalten sein, wie etwa im folgenden die Algorithmen für die elementaren Operationen. Wenn wir jedoch Algorithmen explizit formulieren, lehnen wir uns an die Syntax der Programmiersprache PROSA (vgl. [LoMeWi86], S. 149–164 und S. 226–232) an, ohne allerdings vollständige Programme zu formulieren. Dabei werden wir jedoch auch kompliziertere mathematische Operationen, wie etwa die Matrizenmultiplikation (Algorithmus 1.4), und Elemente der Programmiersprache C (Kommentarklammern und die Operatoren „++" bzw. „−−" für erhöhe bzw. erniedrige um 1) verwenden. Für Feldzugriffe werden wir nicht die übliche Notation mit eckigen Klammern verwenden, sondern die häufig intuitivere Indizierung von Variablen. Schließlich benutzen wir auch informalen Text in Algorithmen, etwa um zu erklären, daß an einer bestimmten Stelle ein früher dargestellter Algorithmus benutzt werden soll und in welchen Variablen er sein Ergebnis hinterlassen soll (z.B. in Algorithmus 1.9) oder wenn wir einen Algorithmus nicht in allen Details zitieren wollen (z.B. Algorithmus 3.10). Implizit stecken solche Aufrufe zuvor erklärter Algorithmen auch in den arithmetischen Operationszeichen, die wir in Algorithmen benutzen.

Nach [Bu92] ergibt sich folgender Zeitbedarf für die elementaren Operationen:

1.1. Satz (Grundrechenarten)

1. *Die Addition oder Subtraktion zweier Zahlen mit höchstens je n Bit kann in Zeit $O(n)$ ausgeführt werden.*
2. *Um eine n–Bit–Zahl mit einer m–Bit–Zahl zu multiplizieren, benötigen wir Zeit $O(mn)$.*
3. *Die Division (mit Rest) einer $(n+m)$–Bit–Zahl durch eine n–Bit–Zahl benötigt Zeit $O(mn)$.*
4. *Hilfreich ist auch folgende Abschätzung: Die Division (mit Rest) einer n–Bit–Zahl durch eine andere Zahl, so daß das Ergebnis eine m–Bit–Zahl ist, kann in Zeit $O(mn)$ ausgeführt werden.*

Beweis:
Indem man einfach gemäß der Schulmethode entsprechende Ziffern von rechts nach links addiert bzw. subtrahiert (unter Berücksichtigung eventueller Überträge), erhält man die angegebene Laufzeit für die Addition bzw. Subtraktion. Dabei kann man zu kurze Zahlen natürlich links mit Nullen auffüllen.

Auch für die Multiplikation benutzen wir die Schulmethode: Wir berechnen in Zeit $O(mn)$ alle Summanden, die in der Schulmethode üblicherweise untereinander geschrieben und am Ende aufaddiert werden. Etwas geschickter addieren wir aber gleich jeden Summanden zum bisherigen Teilergebnis, wobei der Multiplikator von rechts nach links abgearbeitet wird, so daß es ausreicht die n Stellen am linken Ende aufzuaddieren und die Stellen weiter rechts einfach zu übernehmen. Dann kostet jede der m Additionen nach (1) Zeit $O(n)$ und wir erhalten somit die angegebene Laufzeit. (Wenn wir den Multiplikator von links nach rechts abarbeiten, müssen wir stets ganz rechts mit den Additionen anfangen und können möglicherweise wegen Überträgen erst ganz links aufhören, so daß sich eine höhere Laufzeit ergibt!)

Schließlich ist auch die Division wie üblich möglich. Dabei beachten wir allerdings, daß das Dividieren im Binärsystem besonders einfach ist, da der Divisor entweder in den bearbeiteten Teil des Dividenden paßt oder nicht, wir müssen also nicht wie im Dezimalsystem geschickt raten, *wie oft* der Divisor hineinpaßt. Um (4) zu zeigen beachten wir, daß wir um ein Ergebnis der Länge m zu erhalten offenbar m Divisionsschritte ausführen müssen, denn jeder Schritt berechnet ein Bit des Ergebnisses. In jedem Schritt müssen wir schlimmstenfalls den Divisor von einem ebenso langen Teil des Dividenden abziehen, dies benötigt nach (1) Zeit $O(\text{Länge des Divisors})$. Da der Divisor höchstens so lang sein kann wie der Dividend, also nach Voraussetzung höchstens n Bit, erhalten wir Laufzeit $O(mn)$.

Für (3) beachten wir, daß bei dieser Division das Ergebnis höchstens m Bit hat und der Divisor gerade n Bit, also erhalten wir wie eben die angegebene Laufzeit.

Schließlich sei noch bemerkt, daß bei dieser „schriftlichen Division“ als Nebenprodukt auch der Rest der Division berechnet wird, man kann das Verfahren also auch benutzen um für zwei Zahlen u und v die Zahl $u \bmod v$ zu berechnen. □

1.2. Bemerkung

Es gibt Verfahren, mit denen man wesentlich schneller multiplizieren und dividieren kann, jedoch sind diese Verfahren relativ kompliziert und lohnen sich nur für sehr große Zahlen. Insbesondere kann man nach der *Methode von Schönhage und Strassen* zwei n–Bit–Zahlen in Zeit $O(n \log n \log\log n) = O(n^{1+\epsilon})$ multiplizieren. Zudem kann man auch ebenso schnell dividieren.

Will man zusätzlich auch negative Zahlen behandeln können, so muß man ein zusätzliches Bit spendieren, um das Vorzeichen zu speichern. Beim Rechnen muß man dann zusätzlich das Vorzeichen beachten. Dies führt allerdings nicht zu einer Änderung der Laufzeit, da man etwa beim Multiplizieren einfach die Beträge der Zahlen miteinander multipliziert und die Vorzeichenbits der beiden Zahlen vergleicht, um das Vorzeichenbit des Ergebnisses entsprechend zu setzen. Dabei benötigt diese Betrachtung der Vorzeichen offensichtlich konstante Zeit, d.h. ohne eine Änderung in der Laufzeit zu erhalten, können wir auch Zahlen aus $\mathbb{Z}$ betrachten.

Bekanntlich ist die Multiplikation assoziativ und kommutativ, d.h. bei Berechnung eines Produktes ist es für das Ergebnis unwesentlich, in welcher Reihenfolge die Faktoren multipliziert werden. Nun wollen wir eine entsprechende Aussage für die Laufzeit beweisen.

1.3. Satz

Seien $a_1, \ldots, a_k$ Zahlen aus $\mathbb{Z}$, $k \geq 2$. Dabei sei für $1 \leq i \leq k$ a_i eine n_i–Bit–Zahl. Dann benötigt die Berechnung des Produkts $\prod_{i=1}^{k} a_i$ unabhängig von der Reihenfolge, in der die Faktoren miteinander multipliziert werden, stets Zeit

$$O\Big(\sum_{1 \leq i < j \leq k} n_i n_j\Big).$$

Beweis:

Der Beweis erfolgt durch Induktion.

Für k=2 ist die Aussage mit Satz 1.1, Teil (2) klar.

Im Induktionsschritt zerlegen wir die Berechnung in 3 Schritte: Zunächst berechnen wir in beliebiger Reihenfolge ein Produkt aus k_1 Faktoren $a_{i_1}, \ldots, a_{i_{k_1}}$, $k_1 < k$. Nach Induktionsannahme benötigen wir dazu Zeit $O(\sum_{1 \leq x < y \leq k_1} n_{i_x} n_{i_y})$. Dabei erhalten wir als Ergebnis eine Zahl mit $\sum_{1 \leq x \leq k_1} n_{i_x}$ Bits. Dann berechnen wir das Produkt aus den restlichen Faktoren $a_{j_1}, \ldots, a_{j_{k-k_1}}$, wobei $\{i_1, \ldots, i_{k_1}, j_1, \ldots, j_{k-k_1}\} = \{1, \ldots, k\}$. Dies geschieht nach Induktionsvoraussetzung in Zeit $O(\sum_{1 \leq x < y \leq k-k_1} n_{j_x} n_{j_y})$ und liefert eine Zahl mit $\sum_{1 \leq x \leq k-k_1} n_{j_x}$ Bits. Um schließlich die beiden Teilergebnisse miteinander zu multiplizieren, benötigt man nach Satz 1.1, Teil (2) Zeit

$$O\Big(\sum_{1 \leq x \leq k_1} n_{i_x} \sum_{1 \leq x \leq k-k_1} n_{j_x}\Big).$$

Also ergibt sich insgesamt ein Zeitbedarf von

$$O\Big(\sum_{1 \leq x < y \leq k_1} n_{i_x} n_{i_y} + \sum_{1 \leq x < y \leq k-k_1} n_{j_x} n_{j_y} + \sum_{\substack{1 \leq x \leq k_1 \\ 1 \leq y \leq k-k_1}} n_{i_x} n_{j_x}\Big).$$

Nach Umsortieren ergibt sich also die Behauptung. □

1.2 Der euklidische Algorithmus

Der „euklidische Algorithmus“ ist ein Verfahren, um den größten gemeinsamen Teiler, den ggT, zweier Zahlen m und n zu berechnen. Dabei kann dieser Algorithmus leicht so erweitert werden, daß er zugleich auch eine Darstellung des ggT berechnet, d.h. er liefert zusätzlich Zahlen a und b mit $\mathrm{ggT}(m,n) = am + bn$. Mit dieser Erweiterung erhält man den sogenannten „erweiterten euklidischen Algorithmus“, den man z.B. dazu benutzt, in den Körpern $\mathbb{F}_p$ (p Primzahl) die Inverse zu einer gegebenen Zahl auszurechnen. Im folgenden Abschnitt werden wir weitere wichtige Anwendungen des Algorithmus sehen.

Der Algorithmus beruht auf sukzessiver Division mit Rest, man berechnet also Relationen der Form

$$m_{i-1} = q_i m_i + m_{i+1} \qquad \text{mit } 0 \leq m_{i+1} < m_i.$$

Irgendwann erhält man dann $m_{i+1} = 0$ und kann induktiv zeigen, daß m_i der größte gemeinsame Teiler der Startwerte $m = m_0$ und $n = m_1$ ist (vgl. etwa [Bu92]).

Daraus ergibt sich der folgende Algorithmus:

1.4. Algorithmus (Erweiterter euklidischer Algorithmus)
Eingabe: $m, n \in \mathbb{N}_0, m > n$.
Ausgabe: $\mathrm{ggT}(m,n), a, b$ mit $\mathrm{ggT}(m,n) = am + bn$.

1. $\begin{pmatrix} a & b \\ x & y \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. WHILE $n \neq 0$ DO

$$\begin{pmatrix} a & b \\ x & y \end{pmatrix} := \begin{pmatrix} 0 & 1 \\ 1 & -\left\lfloor \frac{m}{n} \right\rfloor \end{pmatrix} \begin{pmatrix} a & b \\ x & y \end{pmatrix};$$

$$\begin{pmatrix} m \\ n \end{pmatrix} := \begin{pmatrix} n \\ m \bmod n \end{pmatrix}$$

3. OD;
4. $\mathrm{ggT} := m$.

1.5. Satz

Der erweiterte euklidische Algorithmus berechnet den ggT *zweier positiver Zahlen* m, n *mit* $m \geq n$ *und seine Darstellung* $\mathrm{ggT}(m,n) = am + bn$. *Dabei gilt für das Endergebnis von Algorithmus 1.4:* $|x| \leq \frac{n}{\mathrm{ggT}(m,n)}$ *und* $|y| \leq \frac{m}{\mathrm{ggT}(m,n)}$.

Beweis:

Wir bezeichnen mit $(m_i)_i$ die Folge der Werte für m, die der Algorithmus berechnet. Dabei sei m_0 der Startwert für m, m_1 ist dann der Startwert für n. Mit $m_2, m_3, \ldots$ bezeichnen wir die sukzessiven Werte von m beim Durchlaufen der Schleife. Dann bilden die m_i eine streng monoton fallende Folge natürlicher Zahlen, also existiert ein minimales j, so daß $m_j = 0$. Außerdem bezeichnen wir mit q_i den Quotienten $\left\lfloor \frac{m_{i-1}}{m_i} \right\rfloor$. Dann ist durch Induktion leicht zu zeigen, daß m_{j-1} der ggT der beiden Zahlen m und n ist und daß gilt: $\mathrm{ggT}(m,n) = am + bn$ (vgl. [Bu92]).

Das Ergebnis für x und y ist die zweite Zeile des Produkt vieler Matrizen, genauer gesagt gilt:

$$\begin{pmatrix} a & b \\ x & y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_{j-1} \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix}.$$

Dabei wird dieses Produkt rechtsgeklammert ausgewertet. Um unsere Abschätzung zu beweisen, werten wir dieses Produkt nun linksgeklammert aus, was wegen der Assoziativität der Matrizenmultiplikation zum selben Ergebnis führt, und erhalten:

$$\begin{pmatrix} x & y \end{pmatrix} = \begin{pmatrix} 1 & -q_{j-1} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_{j-2} \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix}.$$

Mit x_i und y_i bezeichnen wir die sukzessiven Werte für x und y in absteigender Reihenfolge in dieser linksgeklammerten Auswertung, d.h.

$$\begin{pmatrix} x_i & y_i \end{pmatrix} = \begin{pmatrix} 1 & -q_{j-1} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -q_{j-2} \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix},$$

x_1 bzw. y_1 stellen also das Endergebnis dar. Dann gilt:

$$|x_i| \leq \frac{m_i}{\mathrm{ggT}(m,n)}, \qquad |y_i| \leq \frac{m_{i-1}}{\mathrm{ggT}(m,n)}.$$

Dies beweisen wir mittels vollständiger Induktion auf $1, \ldots, j-1$:

Induktionsanfang ((x_{j-1}, y_{j-1}) erfüllt die Behauptung):

Wegen $m_{j-2} = q_{j-1}m_{j-1} + m_j$ gilt:

$$0 = m_j = m_{j-2} - q_{j-1}m_{j-1},$$

also $(x_{j-1}, y_{j-1}) = (1, -q_{j-1})$ und

$$|x_{j-1}| = 1 = \frac{m_{j-1}}{\mathrm{ggT}(m,n)}, \text{ da } \mathrm{ggT}(m,n) = m_{j-1}$$

sowie

$$|y_{j-1}| = q_{j-1} = \frac{m_{j-2}}{m_{j-1}} = \frac{m_{j-2}}{\mathrm{ggT}(m,n)}.$$

Somit ergibt sich in diesem Fall die Behauptung.

Induktionsschluß $((x_i, y_i) \longrightarrow (x_{i-1}, y_{i-1}), i > 1)$:
Es ist $x_{i-1} = y_i$ und somit nach Induktionsvoraussetzung die Behauptung erfüllt. Weiter ist $y_{i-1} = x_i - q_{i-1}y_i$ und somit aufgrund der Dreiecksungleichung $|y_{i-1}| \leq |x_i| + q_{i-1}|y_i|$ (Wie man leicht zeigt, gilt sogar immer die Gleichheit!). Unter Benutzung der Induktionsvoraussetzung ergibt sich also:

$$\begin{aligned} |y_{i-1}| &\leq \frac{m_i}{\mathrm{ggT}(m,n)} + q_{i-1}\frac{m_{i-1}}{\mathrm{ggT}(m,n)} \\ \Longleftrightarrow |y_{i-1}|\,\mathrm{ggT}(m,n) &\leq m_i + q_{i-1}m_{i-1}. \end{aligned}$$

Wegen $m_{i-2} = q_{i-1}m_{i-1} + m_i$ ergibt sich also nunmehr die Behauptung.

Insbesondere erhalten wir also für (x_1, y_1) unseren Satz. □

1.6. Korollar
Die angegebenen Schranken gelten auch für alle Zwischenergebnisse, d.h. es gilt für alle Werte, die man im Laufe des Algorithmus für a und b bzw. für x und y erhält:

$$|x|, |a| \leq \frac{n}{\mathrm{ggT}(m,n)} \qquad \textit{und} \qquad |y|, |b| \leq \frac{m}{\mathrm{ggT}(m,n)}.$$

Beweis:
Diesmal bezeichnen wir mit x_i den Wert von x vor dem i–ten Schleifendurchlauf und analog verwenden wir y_i, a_i und b_i. Nach Satz 1.5 wissen wir schon, daß das Endergebnis für x und y die Behauptung erfüllt. Wir werden jetzt zeigen, daß die Folgen der Beträge der x_i bzw. y_i monoton wachsend sind. Damit folgt dann, da die Werte von a_i und b_i gerade die Werte von x_{i-1} und y_{i-1} sind, daß die Behauptung für alle a_i bzw. b_i sowie für alle x_i und y_i, $1 \leq i \leq j$ gilt.

Um die Monotonie zu zeigen, beweisen wir induktiv folgende erweiterte Behauptung: Es ist $x_i = (-1)^i|x_i|$, $y_i = (-1)^{i-1}|y_i|$ und $|x_{i-1}| \leq |x_i|$, $|y_{i-1}| \leq |y_i|$ für $i \geq 2$, wobei wir der Einfachheit halber noch definieren: $x_0 := 1$, $y_0 := 0$, wodurch die oben angeführte Eigenschaft, daß a_i und b_i gerade die Werte von x_{i-1} und y_{i-1} sind, konsistent auf a_1 und b_1 erweitert wird.

Induktionsanfang ($i = 0, 1$):
Es ist $x_0 = 1$, $y_0 = 0$ und damit $x_0 = (-1)^0|x_0|$ und $y_0 = (-1)^{-1}|y_0|$. Ebenso ist $x_1 = 0$, $y_1 = 1$ und damit $x_1 = (-1)^1|x_1|$ und $y_1 = (-1)^0|y_1|$.

Induktionsschritt ($1 \leq i \longrightarrow i + 1$):
Es ist

$$\begin{aligned} x_{i+1} = a_i - q_{i+1}x_i &= x_{i-1} - q_{i+1}x_i = (-1)^{i-1}|x_{i-1}| - q_{i+1}(-1)^i|x_i| \\ &= (-1)^{i-1}(\underbrace{|x_{i-1}|}_{\geq 0} + \underbrace{q_{i+1}}_{\geq 1}|x_i|) \\ \text{und also} \quad |x_{i+1}| &\geq |x_i| \\ \text{und} \quad x_{i+1} &= (-1)^{i+1}|x_{i+1}|. \end{aligned}$$

Somit ergibt sich also für die Werte von x die Behauptung. Für die Werte von y zeigt man sie durch eine analoge Rechnung. □

1.7. Satz
Algorithmus 1.4 benötigt Zeit $O(\log^2 m)$, um den ggT *zweier positiver Zahlen m und n mit $m \geq n$ und seine Darstellung* $\mathrm{ggT}(m, n) = am + bn$ *zu berechnen.*

Beweis:
(Wir ergänzen den entsprechenden Beweis in [Bu92] um eine Abschätzung für die Laufzeit der Berechnung der Darstellung des ggT.)

Wieder bezeichnen wir, wie im Beweis von Satz 1.5 mit $(m_i)_i$ die Folge der Werte für m, die der Algorithmus berechnet, wieder sei j der kleinste Index, so daß $m_j = 0$. Nun zeigen wir, daß die WHILE–Schleife maximal $O(\log m)$–mal durchlaufen wird, d.h. $j = O(\log m)$. Dazu genügt es, $2m_{i+2} \leq m_i$ zu zeigen, denn dann wird offensichtlich m in mindestens jedem zweiten Durchlauf durch die Schleife um ein Bit kürzer. Es gilt nun aber $m_i = \left\lfloor \frac{m_i}{m_{i+1}} \right\rfloor m_{i+1} + m_{i+2}$ und $m_{i+1} = \left\lfloor \frac{m_{i+1}}{m_{i+2}} \right\rfloor m_{i+2} + m_{i+3}$. Damit ergibt sich:

$$m_i = \underbrace{\left\lfloor \frac{m_i}{m_{i+1}} \right\rfloor}_{\geq 1} m_{i+1} + m_{i+2} \geq m_{i+2} + \underbrace{\left\lfloor \frac{m_{i+1}}{m_{i+2}} \right\rfloor}_{\geq 1} m_{i+2} + \underbrace{m_{i+3}}_{\geq 0} \geq 2m_{i+2}.$$

Indem man zeigt, daß bei Berechnung des ggT zweier aufeinanderfolgender Fibonaccizahlen die maximale Zahl von Iterationen ausgeführt wird, kann man sogar genauer zeigen:

$$j \leq \frac{\log(\sqrt{5}m)}{\log \frac{1+\sqrt{5}}{2}}\ .$$

Ein Beweis dieser Aussage findet sich etwa in [Mi92], S. 31.

In jedem Schleifendurchlauf wird eine Division mit Rest ausgeführt sowie eine Matrizenmultiplikation. Offensichtlich ist der Dividend der Division stets $\leq m$ und das Ergebnis der Division ist q_i, also braucht die Division Zeit $O((\lfloor \log m \rfloor + 1)(\lfloor \log q_{i+1} \rfloor + 1))$. Da alle auftretenden Matrizen den betragsgrößten Eintrag unten rechts haben, wie man im Beweis von Korollar 1.6 sehen kann, liegt die Laufzeit der Matrizenmultiplikation in der selben Größenordnung wie die Zeit für die Multiplikation der aktuellen Werte von q und y. Die Werte für y sind nach Korollar 1.6 durch m beschränkt. Somit ergibt sich für diese Multiplikation der gleiche Zeitbedarf wie für die Division. Damit erhalten wir für alle Schleifendurchläufe zusammen eine Laufzeit von

$$O(\sum_{i=1}^{j-1}(1+\log m)(1+\log q_i)) = O(\log m \log \prod_{i=1}^{j-1} q_i + j \log m).$$

Da $j = O(\log m)$ ergibt sich also eine Laufzeit von $O(\log^2 m + \log m \log \prod_{i=1}^{j-1} q_i)$. Da nun $m_{j-1} \geq q_j m_j$ gilt, folgt also:

$$\begin{aligned}
m = m_0 &\geq q_1 m_1 \\
&\ \vdots \\
&\geq q_1 \cdots q_i m_i \\
&\ \vdots \\
&\geq q_1 \cdots q_{j-1} \underbrace{m_{j-1}}_{=\mathrm{ggT}(m,n) \geq 1} \\
&\geq q_1 \cdots q_{j-1}. \\
\text{Also: } \log \prod_{i=1}^{j-1} q_i &\leq \log m.
\end{aligned}$$

Somit ergibt sich also eine Gesamtlaufzeit von $O(\log^2 m)$. □

1.8. Bemerkung

Satz 1.7 läßt sich auch auf andere Weise beweisen, eine interessante, aber technisch aufwendigere Alternative findet sich etwa in der Diplomarbeit von Achim Müller ([Mü94]).

1.3 Der chinesische Restsatz

Der chinesische Restsatz erlaubt es, aus Ergebnissen einer Operation, die man in Körpern $\mathbb{F}_p$, $p \in \mathbb{P}$ ($\mathbb{P}$ bezeichnet die Menge der Primzahlen), berechnet hat, das Ergebnis der entsprechenden Operation im Ring $\mathbb{Z}$ zu rekonstruieren, falls ein solches existiert. Viele Operationen der linearen Algebra sind über einem Körper leichter ausführbar, als über einem Ring, etwa die Berechnung der Determinanten einer Matrix. Hier würde man gerne den Gaußschen Algorithmus anwenden, was aber über $\mathbb{Z}$ nicht möglich ist. Will man die Berechnung über $\mathbb{Q}$ durchführen, so muß man damit rechnen, daß die einzelnen Einträge in Zähler und Nenner sehr viele Stellen haben. Aus dieser Situation bietet nun der chinesische Restsatz einen Ausweg.

Präziser gesagt liefert der chinesische Restsatz eine Methode, wie man aus Ergebnissen $x_i \bmod p_i$, $1 \leq i \leq k$, p_i paarweise verschiedene Primzahlen, ein Ergebnis $x \bmod \prod_{i=1}^k p_i$ rekonstruiert. Wenn man gleichzeitig weiß, daß auch eine Lösung $y \in \mathbb{Z}$ existiert, und daß diese Lösung $|y| < \frac{1}{2}\prod_{i=1}^k p_i$ erfüllt, so kann man y aus x bestimmen, da gelten soll: $y \equiv x \pmod{\prod_{i=1}^k p_i}$ und zugleich $y \in]-\frac{1}{2}\prod_{i=1}^k p_i, \frac{1}{2}\prod_{i=1}^k p_i[$.

Dabei kann x gemäß folgendem Algorithmus konstruiert werden:

1.9. Algorithmus (Chinesischer Restsatz)
Eingabe: $x_i \in \mathbb{N}_0$, $p_i \in \mathbb{P}$ mit $0 \leq x_i \leq p_i$, $1 \leq i \leq k$.
Ausgabe: $x \in [0, \prod_{i=1}^{k} p_i[$ mit $x \equiv x_i \pmod{p_i}$ für $1 \leq i \leq k$.

1. $M := \prod_{i=1}^{k} p_i$; $x := 0$;
2. FOR $i := 1$ TO k DO
   (a) $M_i := \frac{M}{p_i}$.
   (b) $r_i := (M_i \bmod p_i)$.
   (c) Berechne den ggT mit Darstellung von p_i und r_i. Dies ergibt Zahlen G_i und g_i mit $g_i p_i + G_i r_i = \mathrm{ggT}(p_i, r_i)$
   (d) $x := (x + G_i M_i x_i \bmod \prod_{i=1}^{k} p_i)$.
3. OD.

Wie funktioniert der Algorithmus? In den Schritten (2b) und (2c) wird der Euklidische Algorithmus angewandt, um den ggT von M_i und p_i zu berechnen. Dieser ist aber sicherlich 1, da M_i ja gerade ein Produkt von Primzahlen $\neq p_i$ ist. Dabei erhalten wir in Schritt (2c) auch eine Darstellung $g_i p_i + G_i r_i = 1$. Wenn wir dies rückwärts einsetzen, erhalten wir wegen $r_i = M_i - q p_i$:

$$g_i p_i + G_i(M_i - q p_i) = 1.$$

Somit ist

$$G_i M_i \equiv \begin{cases} 1 & \pmod{p_i} \\ 0 & \pmod{p_j}, j \neq i. \end{cases}$$

Am Ende des Algorithmus gilt nun:

$$x \equiv \sum_{i=1}^{k} G_i M_i x_i \pmod{\prod_{i=1}^{k} p_i}.$$

Also erhält man gerade eine Zahl mit den verlangten Eigenschaften.

Da wir uns auch für die Laufzeit des Algorithmus interessieren, beweisen wir folgendes Resultat, wie es u.a. auch in [Bu92] zu finden ist:

1.10. Satz

Seien $p_1, \ldots, p_k \leq z$. Dann benötigt die Ausführung von Algorithmus 1.9 Zeit

$$O(k^2 \log^2 z).$$

Beweis:

Das Produkt in Schritt (1) können wir nach Satz 1.3 in Zeit $O(\sum_{1 \leq i < j \leq k} \log p_i \log p_j)$ berechnen. Da nach Voraussetzung $p_i \leq z$ ist, ergibt sich also, daß Schritt (1) Zeit $O(k^2 \log^2 z)$ benötigt.

Die Schritte (2a) und (2b) benötigen nach Satz 1.1(3) Zeit $O(k \log z \log z)$, da gilt:

$$\prod_{i=1}^{k} p_i \leq z^k.$$

Für Schritt (2c) benötigen wir laut Satz 1.7 Zeit $O(\log^2 z)$.

Dank Korollar 1.6 wissen wir, daß $G_i \leq p_i$ und somit kann $G_i M_i < \prod_{i=1}^{k} p_i$ nach Satz 1.1(2) in Zeit $O(k \log z \log z)$ ausgerechnet werden. Zur Berechnung von $G_i M_i x_i$ in Schritt (2d) benötigen wir ebenso lange. Für die Reduktion $\bmod \prod_{i=1}^{k} p_i$ benötigen wir schließlich nach Satz 1.1(4) Zeit $O(k \log z \log z)$, da das Ergebnis der Division höchstens $\log z$ Bit haben kann. Die Addition zum bisherigen Wert und die anschließende Reduktion, die durch eine einfache Subtraktion realisiert werden kann, dauert nach Satz 1.1(1) höchstens $O(k \log z)$.

Wenn man berücksichtigt, daß die Schritte (2a), (2b), (2c) und (2d) jeweils k-mal ausgeführt werden, ergibt sich die angegebene Gesamtlaufzeit. □

1.4 Anwendungen des chinesischen Restsatzes

Bei naiver Betrachtung des Problems, wie man z.B. Gleichungssysteme über $\mathbb{Z}$ lösen kann, denkt man möglicherweise zunächst daran, mit Hilfe des Gaußschen Algorithmus das Gleichungssystem über $\mathbb{Q}$ zu lösen. Dieses Vorgehen hat jedoch den Nachteil, daß die Zahlen, mit denen man rechnen muß, auch für kleine Eingabematrizen sehr groß werden können. Im wesentlichen sind die Zahlen nämlich $O(n)$ Bit lang. Wenn man also ein Gleichungssystem mit dem Gaußschem Algorithmus lösen will, muß man $O(n^3)$ Operationen auf Zahlen der Länge $O(n)$ durchführen, so daß sich Kosten $O(n^5)$ ergeben, wenn man für die Multiplikation zweier n–Bit–Zahlen Zeit $O(n^2)$ veranschlagt. Weil dies zu teuer ist, suchen wir eine bessere Methode.

Rechnet man nun mit Floating–Point–Zahlen, um das Problem der großen Zahlen zu vermeiden, so muß man aber eine aufwendige Rundungsfehlerkontrolle betreiben. Zudem erhält man am Ende nicht direkt einen Vektor über $\mathbb{Z}$, man muß hier also versuchen, mit Hilfe der näherungsweise berechneten Lösung die richtige Lösung zu „erraten“.

Aufgrund dieser Probleme benutzen wir hierfür den Chinesischen Restsatz. Wie wir mit seiner Hilfe effiziente Algorithmen für einige Aufgaben der linearen Algebra erhalten, die wir in späteren Kapiteln lösen müssen, wollen wir im folgenden zeigen.

1.4.1 Definitionen

1.11. Definition

Wir werden folgende Bezeichnungen für Mengen von Matrizen benutzen:

- *Mit* $\mathbb{Z}^{n\times n}$ *bezeichnen wir den Ring aller* $n \times n$*–Matrizen über* $\mathbb{Z}$*.*
- *Mit* $\mathrm{GL}(n, \mathbb{Z})$ *bezeichnen wir die Gruppe der über* $\mathbb{Z}$ *invertierbaren Matrizen aus* $\mathbb{Z}^{n\times n}$*, d.h. also alle Matrizen* $A \in \mathbb{Z}^{n\times n}$ *mit* $|\det(A)| = 1$*.*

Analoge Bezeichnungen benutzen wir auch für Matrizen über Körpern, wobei $\mathrm{GL}(n, \cdot)$ *dann alle Matrizen mit von 0 verschiedener Determinante bezeichnet.*

1.12. Definition (Vektornormen)

Sei $\underline{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$*. Dann definieren wir*

- *die* euklidische Norm*:*
$$\|\underline{a}\| := \|\underline{a}\|_2 := \sqrt{\sum_{i=1}^{n} a_i^2}$$
- *die* Maximumnorm*:*
$$|\underline{a}| := \|\underline{a}\|_\infty := \max_{1\le i\le n} |a_i|.$$

1.13. Definition (Matrixnormen)

Sei $A = (\underline{a}_1, \ldots, \underline{a}_n) = (a_{ij})_{1\le i,j\le n} \in \mathbb{R}^{n\times n}$*. Dann definieren wir*

- *die* Frobeniusnorm*:*
$$\|A\| := \|A\|_2 := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$
- *die* Zeilensummennorm*:*
$$\|A\|_\infty := \max_{1\le i\le n} \sqrt{\sum_{j=1}^{n} a_{ij}^2}.$$

Außerdem bezeichnen wir den Betrag des betragsgrößten Elements einer Matrix A *mit* $|A|$*, d.h.*
$$|A| := \max_{1\le i,j\le n} |a_{ij}|.$$

Dann gilt bekanntlich für $A \in \mathbb{R}^{n\times n}$ und $t \in \mathbb{R}^n$:

$$\|At\| \leq \|A\|\|t\|$$

sowie

$$\|At\|_\infty \leq \|A\|_\infty \|t\|_\infty.$$

Im folgenden verwenden wir häufig die Hadamardsche Ungleichung, die für alle Matrizen $A = (\underline{a}_1, \ldots, \underline{a}_n) \in \mathbb{R}^{n\times n}$ gilt:

$$|\det(A)| \leq \prod_{j=1}^{n} \|\underline{a}_j\| \leq (\sqrt{n}|A|)^n, \tag{1.1}$$

wobei in der linken Ungleichung die Gleichheit genau dann gilt, wenn die Spaltenvektoren paarweise orthogonal sind. Ein Beweis dieser Ungleichung findet sich etwa in [Mi92], S. 302f.

1.4.2 Lineare Gleichungssysteme über $\mathbb{Z}$ und verwandte Probleme

Sei $A = (\underline{a}_1, \ldots, \underline{a}_n) \in \mathrm{GL}(n, \mathbb{Z})$ und $\underline{b} \in \mathbb{Z}^n$. Dann ist $A^{-1} \in \mathrm{GL}(n, \mathbb{Z})$ und also $A^{-1}\underline{b} \in \mathbb{Z}^n$. Also hat das Gleichungssystem $A\underline{x} = \underline{b}$ mit $\underline{x} = (x_1, \ldots, x_n)$ eine eindeutige Lösung, die in $\mathbb{Z}^n$ liegt, nämlich $A^{-1}\underline{b}$. Etwas allgemeiner können wir auch Gleichungssysteme $A\underline{x} = \underline{b}$ betrachten, für die nur gilt $A \in \mathbb{Z}^{n\times n}$ mit $\det(A) \neq 0$. Dann wissen wir im allgemeinen jedoch nicht, ob die Lösung des Gleichungssystems wieder in $\mathbb{Z}^n$ liegt. Allerdings wissen wir, daß die Lösung in $\mathbb{Q}$ eindeutig bestimmt ist, da $A \in \mathrm{GL}(n, \mathbb{Q})$. Wenn es also eine Lösung in $\mathbb{Z}$ gibt, so ist dies die eindeutige Lösung. Die aus der Algebra bekannte Kramersche Regel besagt, daß wir die Komponenten des Lösungsvektors wie folgt berechnen können:

$$x_i = \frac{\det(\underline{a}_1, \ldots, \underline{a}_{i-1}, \underline{b}, \underline{a}_{i+1}, \ldots, \underline{a}_n)}{\det(A)}.$$

Einen Beweis kann man etwa in [Jä91] auf Seite 160 nachlesen.

Mit der Hadamardschen Ungleichung (1.1) erhalten wir also folgende Abschätzung:

$$|x_i| \leq \prod_{\substack{j=1\\ j\neq i}}^{n} \|\underline{a}_j\| \|\underline{b}\| \frac{1}{|\det(A)|}. \tag{1.2}$$

Diese Abschätzung können wir abschwächen zu:

$$|x_i| \leq \frac{(\sqrt{n}|A|)^{n-1}}{|\det(A)|}\|\underline{b}\|. \tag{1.3}$$

Wir zeigen nun folgenden Satz:

1.14. Satz

Seien $A \in \mathbb{Z}^{n\times n}$ mit $\det(A) \neq 0$ und $\underline{b} \in \mathbb{Z}^n$ gegeben, so daß das Gleichungssystem $A\underline{x} = \underline{b}$ eine Lösung in $\mathbb{Z}^n$ hat. Dann kann die Lösung in Zeit

$$O(\left[n^3 \log\log((n|A|)^{n-1}\|\underline{b}\|) + n\log((n|A|)^{n-1}\|\underline{b}\|)\right] \log((n|A|)^{n-1}\|\underline{b}\|))$$

berechnet werden.

Beweis:

In diesem Beweis werden die folgenden Ergebnisse der analytischen Zahlentheorie benutzt:

$$\pi(z) \;:=\; \#\{p|p \leq z, p \text{ prim}\} = O(\frac{z}{\log z}) \text{ (vgl. [CoLeRi90], Seite 837).} \tag{1.4}$$

$$\theta(z) \;:=\; \sum_{\substack{p\leq z\\ p \text{ prim}}} \log p = \Omega(z) \text{ nach [RoSc85], Korollar 3.16.} \tag{1.5}$$

Mit (1.3) erhalten wir zunächst $|x_i| \leq c' := \frac{(\sqrt{n}|A|)^{n-1}}{|\det(A)|}\|\underline{b}\|$. Wir werden nun die Lösung des reduzierten Gleichungssystems in Körpern $\mathbb{F}_p$ berechnen und die modularen Ergebnisse mit dem chinesischen Restsatz zusammensetzen. Um nun das eindeutige Ergebnis in $\mathbb{Z}$ zu sichern, müssen wir verlangen, daß für alle betrachteten Primzahlen p das Gleichungssystem über $\mathbb{F}_p$ eine eindeutige Lösung hat, d.h. A muß über $\mathbb{F}_p$ invertierbar sein oder äquivalent: p ist kein Teiler von $\det(A)$. Wenn also $A \in \mathrm{GL}(n, \mathbb{Z})$, so können wir alle Primzahlen p benutzen, anderenfalls müssen wir alle Teiler von $\det(A)$ verbieten. Das Produkt aller *verschiedenen* Primzahlen, die $\det(A)$ teilen, ist aber höchstens $\det(A)$. Daher genügt es statt $\prod_{\substack{p\leq z\\ p \text{ prim}}} p \geq 2c'$, jetzt

$$\prod_{\substack{p\leq z\\ p \text{ prim}}} p \geq c := 2(\sqrt{n}|A|)^{n-1}\|\underline{b}\| \tag{1.6}$$

zu fordern, um zu gewährleisten, daß wir genügend modulare Teilergebnisse erhalten, um das Endergebnis zu rekonstruieren. Dieses c ist auch leicht (insbesondere weit unter der im Satz erlaubten Zeit!) zu berechnen.

Wir berechnen also die Lösung des Gleichungssystems modulo aller Primzahlen $\leq z$, wobei es wegen (1.5) ausreicht, $z \in O(\log c)$ zu wählen, denn dann ist $\prod_{\substack{p \leq z \\ p \text{ prim}}} p = \Omega(2^z) = \Omega(c)$ und bei richtiger Wahl von z ist also (1.6) erfüllt. Nachdem wir nun die Ergebnisse modulo ungeeigneter Primzahlen entfernt haben, bleiben also garantiert genügend Ergebnisse übrig, um mit dem chinesischen Restsatz ein modulo $2c'$ eindeutiges Ergebnis zusammmensetzen zu können.

Die Lösungen in den Körpern $\mathbb{F}_p$ berechnen wir jeweils mit dem Gaußschen Algorithmus, also werden jeweils $O(n^3)$ arithmetische Operationen benötigt. Wenn wir dabei zum Invertieren die ggT–Berechnung mittels des erweiterten euklidischen Algorithmus verwenden und Division durch Invertieren und anschließende Multiplikation realisieren, benötigt jede arithmetische Operation Zeit $O(\log^2 z)$. Nach (1.4) gibt es $O(\frac{z}{\log z})$ Primzahlen $\leq z$. Somit ergibt sich eine Rechenzeit von $O(n^3 \log^2 z \frac{z}{\log z}) = O(n^3 z \log z)$. Dabei können wir leicht auch die Determinante modulo p mitberechnen und so die Ergebnisse, die wir modulo Primzahlen erhalten, die die Determinante teilen, gleich als ungeeignet aussondern.

Mit dem chinesischen Restsatz müssen wir zum Schluß $O(\frac{z}{\log z})$ Vektoren zum Ergebnis zusammensetzen. Da diese Vektoren jeweils n Komponenten haben, braucht man n-mal so lange wie in Satz 1.10 für einfache Zahlen angegeben. Somit ergibt sich eine Laufzeit von

$$O\left(n\left(\frac{z}{\log z}\right)^2 \log^2 z\right) = O(nz^2).$$

Insgesamt ergibt sich also eine Laufzeit von $O(n^3 z \log z + nz^2)$, d.h. die Behauptung. $\square$

Später wird sich auch noch eine evidente Folgerung als hilfreich erweisen:

1.15. Korollar

Seien $A \in \mathbb{Z}^{n\times n}$ mit $\det(A) \neq 0$ und $\underline{b} \in \mathbb{Z}^n$ mit $\|\underline{b}\| = (n|A|)^{O(n)}$ gegeben, so daß das Gleichungssystem $A\underline{x} = \underline{b}$ eine Lösung in $\mathbb{Z}^n$ hat. Dann kann das Gleichungssystem in Zeit

$$O(\left[n^4 \log(n \log(n|A|)) + n^3 \log(n|A|)\right] \log(n|A|))$$

gelöst werden.

Beweis:
Aus Satz 1.14 ergibt sich zunächst eine Laufzeit von

$$O(\Big[n^3 \log\log((n|A|)^{n-1}(n|A|)^{O(n)}) + n\log((n|A|)^{n-1}(n|A|)^{O(n)})\Big] \\ \log((n|A|)^{n-1}(n|A|)^{O(n)})).$$

Durch einige Umformungen erhalten wir:

$$\begin{aligned} & O(\Big[n^3 \log\log((n|A|)^{O(n)}) + n\log((n|A|)^{O(n)})\Big] \log((n|A|)^{O(n)})) \\ = \; & O(\Big[n^3 \log(O(n)\log(n|A|)) + O(n^2)\log(n|A|)\Big] O(n)\log(n|A|)) \\ = \; & O(\Big[n^4 \log(n\log(n|A|)) + n^3\log(n|A|)\Big] \log(n|A|)). \end{aligned}$$

□

Dieses Resultat läßt sich sehr leicht auf die Berechnung der Determinanten übertragen, wobei wir folgendes Ergebnis erhalten:

1.16. Korollar
Sei $A \in \mathbb{Z}^{n\times n}$. Dann kann die Determinante von A in Zeit

$$O(\Big[n^4 \log(n\log(n|A|)) + n^2\log(n|A|)\Big] \log(n|A|))$$

berechnet werden.

Beweis:
Für die Determinante gilt dank der Hadamardschen Ungleichung (1.1) im wesentlichen die gleiche Schranke c, die wir für das Verfahren zur Lösung eines Gleichungssystems benutzt haben. Zudem benutzt man zur Berechnung der Determinanten den gleichen Algorithmus, wie zur Lösung eines Gleichungssystems, nämlich die Gaußelimination. Daher ergibt sich durch die gleiche Rechnung wie im Beweis von Satz 1.14 das behauptete Resultat. Noch einfacher erhalten wir hier über $\mathbb{F}_p$ sogar immer ein eindeutiges Ergebnis! Dabei müssen wir jetzt allerdings den chinesichen Restsatz nicht mehr auf Vektoren mit n Komponenten anwenden, sondern nur noch auf einzelne Zahlen, daher wird der zweite Summand um einen Faktor n kleiner. □

Damit läßt sich dieses Resultat auch auf die Berechnung der adjungierten Matrix übertragen:

1.17. Korollar

Sei $A \in \mathbb{Z}^{n\times n}$ *mit* $\det(A) \neq 0$. *Dann kann die adjungierte Matrix* $\mathrm{ad}(A)$ *bzw. die inverse Matrix* $A^{-1} \in \mathrm{GL}(n,\mathbb{Q})$ *von* A *in Zeit*

$$O(\left[n^4 \log(n \log(n|A|)) + n^3 \log(n|A|)\right] \log(n|A|))$$

berechnet werden. Dabei gilt:

$$|\,\mathrm{ad}(A)| \leq (\sqrt{n}|A|)^{n-1}.$$

Beweis:

Mann berechnet zunächst die Determinante von A. Nach Korollar 1.16 ist dies in der angegebenen Zeit möglich. Um dann die adjungierte Matrix zu erhalten, löst man über $\mathbb{Z}$ simultan die n Gleichungssysteme $A\,\mathrm{ad}(A) = \det(A)I_n$. Dabei ist zu beachten, daß das simultane Lösen von n Gleichungssystemen nicht teurer ist, als das Lösen eines Gleichungssystems, da sich lediglich die Kosten für eine Zeilenoperation von $n+1$ auf $2n$ arithmetische Operationen erhöhen, was beides $O(n)$ ergibt.

Durch die spezielle Wahl der rechten Seiten der n Gleichungssysteme ist sichergestellt, daß Lösungen in $\mathbb{Z}$ existieren. Da zudem die Vektoren auf der rechten Seite gemäß der Hadamardschen Ungleichung (1.1) höchstens Länge $(n|A|)^{O(n)}$ haben, können wir Korollar 1.15 anwenden und erhalten somit die behauptete Laufzeit. Damit haben wir die Determinante und die adjungierte Matrix von A berechnet. Wegen $A^{-1} = \frac{1}{\det(A)}\,\mathrm{ad}(A)$ kennen wir damit auch die inverse Matrix mit Koeffizienten in $\mathbb{Q}$. Die behauptete Schranke für $|\,\mathrm{ad}(A)|$ folgt wegen $A\,\mathrm{ad}(A) = \det(A)I_n$ unmittelbar aus (1.3). $\square$

1.18. Bemerkung

Die Berechnung der adjungierten Matrix kann sogar effizienter realisiert werden, indem man die beiden konzeptionellen Teilschritte (Berechnung der Determinante, dann Berechnung der Adjungierten) integriert (vgl. [NeRo92]). Dann muß man für jeden der Moduln die Gauß-Elimination nur einmal statt zweimal durchführen und kann so die Laufzeit in etwa um ein Drittel senken.

Wenn man nun Gleichungssysteme mit sehr großen Einträgen auf der rechten Seite (schätzungsweise mindesetens in der Größenordnung $(n|A|)^{2n}$) zu lösen hat, dann zeigt eine exakte Analyse, daß es vorteilhafter sein kann, das Gleichungssystem auf einem Umweg zu lösen: Man berechnet zunächst die adjungierte Matrix zu A, berechnet dann

$\mathrm{ad}(A)b$ und teilt das Ergebnis dann durch $\det(A)$ und erhält somit die Lösung. Insbesondere erhält man auf diese Weise auch die korrekte Lösung über $\mathbb{Q}$, falls das Gleichungssystem über $\mathbb{Z}$ nicht lösbar ist. Dieses Verfahren wird in der Diplomarbeit von Achim Müller ([Mü94]) ausführlich beschrieben. Wir benötigen jedoch im weiteren nur den Spezialfall aus Korollar 1.15, für den dieses Verfahren dieselbe Laufzeit liefert wie das zuvor beschriebene Verfahren.

1.4.3 Berechnung des charakteristischen Polynoms

Nun wollen wir noch eine weitere Anwendung des chinesischen Restsatzes betrachten, nämlich die Berechnung des charakteristischen Polynoms einer Matrix $A \in \mathbb{Z}^{n\times n}$, wobei wir das charakteristische Polynom χ_A einer Matrix A als $\det(x \cdot I_n - A)$ und nicht als $\det(A - x \cdot I_n)$ definieren. Letztere Definition ist in der Algebra gebräuchlicher, ein Unterschied besteht jedoch höchstens im Vorzeichen des Polynoms, durch Multiplikation mit $(-1)^n$ geht die eine Form aus der anderen hervor.

Es erweist sich dabei als günstig, die Matrix zunächst auf *Hessenberg–Form* zu transformieren, das heißt eine zu A ähnliche Matrix $\tilde{A}$ der folgenden Form zu berechnen:

$$\tilde{A} = \begin{pmatrix} h_{11} & h_{12} & \cdots & \cdots & h_{1,n-1} & h_{1n} \\ k_2 & h_{22} & \cdots & \cdots & h_{2,n-1} & h_{2n} \\ 0 & k_3 & \cdots & \cdots & h_{3,n-1} & h_{3n} \\ \vdots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & k_n & h_{nn} \end{pmatrix}$$

Dies gelingt, indem man von links nach rechts gehend die Spalten unterhalb der Nebendiagonalen mittels geeigneter Zeilentransformationen der Reihe nach zu Null macht und durch passende Spaltentransformationen dafür sorgt, daß man eine zu A ähnliche Matrix erhält. Da $\tilde{A}$ ähnlich zu A ist, gilt dabei: Es gibt eine invertierbare Matrix B mit $\tilde{A} = BAB^{-1}$. Wegen $\chi_{\tilde{A}} = \det(xI_n - BAB^{-1}) = \det(B)\det(xI_n - A)\det(B^{-1}) = \chi_A$ stimmen dann die charakteristischen Polynome von A und $\tilde{A}$ überein. Nun stellen wir einen Algorithmus zur Berechnung der Hessenberg–Form aus [Co93] vor:

Man sucht in der aktuellen Spalte, die wir als m–te bezeichnen wollen, das erste von Null verschiedene Element unterhalb der Nebendiagonalen, wir nennen es a_{im}. Dann vertauschen wir die i–te Zeile mit der (m+1)–ten Zeile, da höchstens in der (m+1)–ten Zeile ein von Null verschiedenes Element stehen soll, sowie die i–te Spalte mit der (m+1)–ten Spalte, damit wir eine Ähnlichkeitstransformation erhalten. Dann machen wir die Elemente unterhalb des neuen a_{im} zu Null, wie dies auch im Gaußschen Algorithmus geschieht; damit wir allerdings eine Ähnlichkeitstransformation erhalten, müssen wir zu jeder Zeilentransformation noch eine analoge Spaltentransformation ausführen.

Im oben erwähnten Buch findet man auch die folgende Rekursionsformel zur Berechnung des charakteristischen Polynoms der Hessenbergmatrix:

$$p_0(x) = 1$$
$$p_m(x) = (x - h_{mm})p_{m-1}(x) - \sum_{i=1}^{m-1} \left(h_{im} \left(\prod_{j=i+1}^{m} k_j \right) p_{i-1}(x) \right).$$

Dabei bezeichnet p_m das charakteristische Polynom der oberen linken $m \times m$–Untermatrix. Von der Richtigkeit dieser Formel kann man sich leicht durch Entwicklung nach der jeweils letzten Spalte der Untermatrix überzeugen.

Dabei gilt für die Größe der Koeffizienten:

1.19. Proposition
Die Koeffizienten des charakteristischen Polynoms einer Matrix A sind durch $(n|A|)^n$ beschränkt.

Beweis:
Der Satz von Gershgorin besagt, daß die Menge aller Nullstellen des charakteristischen Polynoms einer Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ in der Vereinigung der Kreise $K_i := \{x \in \mathbb{C} \mid |c - a_{ii}| \leq \sum_{\substack{k=1 \\ k \neq i}}^{n} |a_{ik}|\}$ enthalten ist (vgl. etwa [StBu78], S. 77). Daraus ergibt sich, daß die Nullstellen des charakteristischen Polynoms betragsmäßig durch $n|A|$ beschränkt sind. Aus der Darstellung der Koeffizienten mittels symmetrischer Funktionen der Nullstellen ergibt sich dann, daß der $(n-i)$-te Koeffizient beschränkt ist durch

$$\begin{aligned} \binom{n}{i}(n|A|)^i &= \frac{n^{i+1}(n-1)\cdots(n-i+1)}{i!}|A|^i \\ &\leq n^n|A|^i \\ &\leq (n|A|)^n \end{aligned}$$

□

Ferner gilt der folgende Satz:

1.20. Satz

Zur Berechnung des charakteristischen Polynoms einer Matrix $A \in \mathbb{F}_p^{n\times n}$ benötigen wir Zeit

$$O(n^3 \log^2 p)$$

Beweis:

Zunächst betrachten wir die Berechnung der Hessenbergform. Dazu untersuchen wir das beschriebene Verfahren nochmals Schritt für Schritt:

- Die Suche des ersten von Null verschiedenen Elements unterhalb der Nebendiagonalen benötigt $O(n)$ Vergleiche
- Für die Zeilen– und Spaltenvertauschungen benötigen wir $O(n)$ Vertauschungen von einzelnen Elementen.
- Das annullieren der Elemente unterhalb von a_{im} benötigt $O(n)$ Zeilen– und Spaltenoperationen, also $O(n^2)$ Operationen mit Körperelementen.

Da wir diese Operationen für $O(n)$ Spalten ausführen und eine einzelne Operation im Körper $\mathbb{F}_p$ Zeit $O(\log^2 p)$ benötigt (insbesondere reicht das auch aus, um zu dividieren!), ergibt sich ein Zeitaufwand von $O(n^3 \log^2 p)$ zur Berechnung der Hessenbergform. Für die rekursive Berechnung des charakteristischen Polynoms ergibt sich ebenfalls ein Zeitaufwand von $O(n^3 \log^2 p)$, da wir $O(n)$ Rekursionsschritte ausführen, und in jedem Schritt werden für jeden der $O(n)$ Koeffizienten $O(n)$ Körperoperationen ausgeführt. $\square$

Aus den beiden vorangehenden Resultaten ergibt sich, indem wir den chinesischen Restsatz anwenden:

1.21. Satz

Zur Berechnung des charakteristischen Polynoms einer Matrix $A \in \mathbb{Z}^{n\times n}$ benötigen wir Zeit

$$O(n^4 \log^2(n|A|)).$$

Beweis:

Der Beweis ist dem Beweis von Satz 1.14 sehr ähnlich. Die Größe der Koeffizienten des charakteristischen Polynoms ist nach Proposition 1.19 durch $c := (n|A|)^n$ beschränkt. Wiederum können wir dieses c innerhalb der vorgegebenen Zeit leicht berechnen.

Daher berechnen wir das charakteristische Polynom modulo aller Primzahlen $\leq z$, wobei $z = O(\log c)$, was stets zu einem eindeutigen Ergebnis führt. Dazu benötigt man nach Satz 1.20 jeweils Zeit $O(n^3 \log^2 z)$. Nach (1.4) gibt es $O(\frac{z}{\log z})$ Primzahlen $\leq z$. Somit ergibt sich eine Rechenzeit von $O(n^3 z \log z)$.

Wie im Beweis von Satz 1.14 erhalten wir mit Hilfe des chinesichen Restsatzes ein Ergebnis, das eindeutig modulo $2c$ ist, wenn z geeignet gewählt wurde, und daraus wiederum das Ergebnis in $\mathbb{Z}$. Mit dem chinesischen Restsatz müssen wir nun $k = O(\frac{z}{\log z})$ Koeffiziententupel zum Ergebnis zusammensetzen. Dazu benötigen wir wie oben gesehen Zeit $O(nz^2)$. Insgesamt ergibt sich also wiederum eine Laufzeit von $O(n^3 z \log z + nz^2)$, d.h. also eine Laufzeit von

$$\begin{aligned} &O(n^3 \log(n|A|)^n \log\log(n|A|)^n + n \log^2(n|A|)^n) \\ &= O(n^4 \log(n|A|) \log(n \log(n|A|)) + n^3 \log^2(n|A|)). \\ &= O(n^4 \log^2(n|A|)). \end{aligned}$$

□

1.4.4 Berechnung des ggT zweier Polynome

Jetzt wollen wir den größten gemeinsamen Teiler zweier Polynome aus $\mathbb{Z}[x]$ berechnen. Wenn man hierzu eine naive Übertragung des euklidischen Algorithmus benutzt, hat man ähnliche Probleme mit der Größe der Zwischenergebnisse wie beim Lösen von Gleichungssystemen. Daher wählen wir wiederum einen Zugang über die Berechnung modularer Teilergebnisse, wobei wir für die modularen Berechnungen eine einfache Übertragung des euklidischen Algorithmus benutzen können. Diese Teilergebnisse setzten wir wieder mit dem chinesischen Restsatz zum Ergebnis über $\mathbb{Z}$ zusammen. Dabei tritt allerdings ein ähnliches Problem bei der Wahl der Primzahlen auf, wie bei der Lösung von Gleichungssystemen. Zudem tritt eine zusätzliche Schwierigkeit auf, die wir bisher nicht hatten.

Zunächst müssen wir einige Begriffe für die Handhabung von Polynomen einführen:

1.22. Definition

Es sei R ein Ring mit 1. Dann heißt eine Zahl, zu der in R ein Inverses existiert, eine Einheit *in R. Sei $f \in R[x]$ ein Polynom mit Koeffizienten in R, $f(x) = \sum_{i=0}^{n} a_i x^i$, $a_n \neq 0$. Dann heißt n der* Grad *des Polynoms, bezeichnet mit $\deg(f)$ und a_n heißt* Leitkoeffizient *des Polynoms, bezeichnet mit $\mathrm{lc}(f)$. Falls $\mathrm{lc}(f) = 1$, so heißt f* normiert. *Wenn in R zusätzlich ein ggT existiert, dann bezeichnet man mit $\mathrm{cont}(f)$ einen* Inhalt *von f, d.h. einen ggT der Koeffizienten. Wenn $\mathrm{cont}(f)$ eine Einheit ist, so heißt f* primitiv. *Allgemeiner heißt $\frac{f(x)}{\mathrm{cont}(f)}$* primitiver Teil *von f, in Zeichen $\mathrm{pp}(f)$. Schließlich identifizieren wir häufig Polynome mit ihren Koeffizientenvektoren, so daß z.B. mit $|f|$ der Betrag des betragsgrößten Koeffizienten bezeichnet wird.*

1.23. Bemerkung

Der ggT und somit der Inhalt sind im allgemeinen nur bis auf Einheiten eindeutig bestimmt! Häufig kann man aber eine gewisse Normierung einführen, die es erlaubt von *dem* Inhalt eines Polynoms zu sprechen. Zum Beispiel legt man für $R = \mathbb{Z}$ fest, daß der ggT stets eine positive Zahl sein soll und erhält damit einen eindeutig definierten ggT, da 1 und -1 die einzigen Einheiten in $\mathbb{Z}$ sind.

Nun wollen wir sehen, wie man den ggT zweier Polynome modulo einer Primzahl p berechnet. Da in $\mathbb{F}_p$ alle von 0 verschiedenen Zahlen Einheiten sind und der ggT nur bis auf Einheiten eindeutig bestimmt ist, können wir vereinbaren, daß wir als ggT stets ein normiertes Polynom erhalten. Dabei kann man eine einfache Übertragung des euklidischen Algorithmus benutzen, um den ggT zu berechnen, wobei wir jetzt auf die Berechnung einer Darstellung verzichten. Der wichtigste Bestandteil des euklidischen Algorithmus ist die Division mit Rest. Daher wollen wir zunächst sehen, wie wir diese über $\mathbb{F}_p[x]$ ausführen können:

1.24. Algorithmus (Polynomdivision)
Eingabe: Polynome $f_1, f_2 \in \mathbb{F}_p[x]$.
Ausgabe: Polynome $q, r \in \mathbb{F}_p[x]$ mit $f_1 = qf_2 + r$ und $\deg(r) < \deg(f_2)$.

1. $q := 0$; $r := f_1$;
2. WHILE $\deg(r) \geq \deg(f_2)$ DO
   - $q := q + \frac{\mathrm{lc}(r)}{\mathrm{lc}(f_2)} x^{\deg(r)-\deg(f_2)}$;

- $r := r - \frac{\mathrm{lc}(r)}{\mathrm{lc}(f_2)} x^{\deg(r)-\deg(f_2)} f_2$

3. OD.

Da klar ist, daß der Algorithmus das gewünschte Ergebnis liefert, wollen wir nur noch festhalten, in welcher Zeit dies geschieht:

1.25. Satz
Seien $f_1, f_2 \in \mathbb{Z}_p[x]$ *mit* $\deg(f_1) \geq \deg(f_2)$. *Dann benötigt Algorithmus 1.24 Zeit* $O((\deg(f_1) - \deg(f_2) + 1)\deg(f_2)\log^2 p)$ *zur Berechnung von* q *und* r *mit* $f_1 = qf_2 + r$ *und* $\deg(r) < \deg(f_2)$.

Beweis:
Da wir mit Koeffizienten aus $\mathbb{F}_p$ rechnen, benötigt jede Körperoperation mit solchen Koeffizienten Zeit $O(\log^2 p)$. Die While–Schleife wird höchstens $(\deg(f_1) - \deg(f_2) + 1)$–mal durchlaufen, da in jeder Iteration der Grad von r um mindestens 1 absinkt, und in jeder Iteration müssen wir höchstens $O(\deg(f_2))$ Körperoperationen ausführen. Daraus ergibt sich die angegebene Laufzeit. □

Damit erhalten wir die folgende Übertragung des euklidischen Algorithmus (Algorithmus 1.4):

1.26. Algorithmus (Euklidischer Algorithmus für Polynome über $\mathbb{F}_p$)
Eingabe: Polynome $f_1, f_2 \in \mathbb{F}_p[x]$.
Ausgabe: Normiertes Polynom $g = \mathrm{ggT}(f_1, f_2) \in \mathbb{F}_p[x]$.

1. WHILE $f_2 \neq 0$ DO
   - Berechne mit Algorithmus 1.24 q, r mit $f_1 = qf_2 + r$, $\deg(r) < \deg(f_2)$;
   - $f_1 := f_2;\ f_2 := r$
2. OD;
3. ggT $:= \frac{f_1}{\mathrm{lc}(f_1)}$.

Dann gilt:

1.27. Satz
Mit Hilfe von Algorithmus 1.26 kann man den ggT zweier Polynome f_1 und f_2 aus $\mathbb{F}_p[x]$ mit $\deg(f_1) \geq \deg(f_2)$ in Zeit $O(\deg(f_1)^2 \log^2 p)$ berechnen.

Beweis:
Wiederum kann man mittels vollständiger Induktion leicht zeigen, daß der Algorithmus einen gemeinsamen Teiler von f_1 und f_2 berechnet und daß jeder andere gemeinsame Teiler das Ergebnis teilt.

Für die Laufzeitbetrachtungen verwenden wir einen ähnlichen „Trick“ wie im Beweis von Satz 1.7. Wir bezeichnen mit $(f_i)_i$ die Folge der Polynome, die wir im Algorithmus für f_1 erhalten. Dabei sei j der kleinste Index, so daß $f_j = 0$. Da der Grad der Polynome in jedem Schritt um mindestens 1 kleiner wird, führen wir also höchstens $\deg(f_1)$ Iterationen aus, d.h. $j = O(\deg(f_1))$.

In jedem Schleifendurchlauf wird nun eine Polynomdivision ausgeführt. Diese Division benötigt nach Satz 1.25 Zeit $O((\deg(f_i) - \deg(f_{i+1}) + 1)\deg(f_{i+1})\log^2 p)$. Da der Grad der Polynome mit jedem Schritt abnimmt, können wir dies durch $O((\deg(f_i) - \deg(f_{i+1}) + 1)\deg(f_1)\log^2 p)$ abschätzen. Insgesamt ergibt sich also ein Zeitaufwand von:

$$O(\sum_{i=1}^{j-2} \deg(f_1)(\deg(f_i) - \deg(f_{i+1}) + 1)\log^2 p) = O(\deg(f_1)^2 \log^2 p).$$

□

Jetzt wollen wir untersuchen, wie wir solche modularen Teilergebnisse zum Ergebnis über $\mathbb{Z}$ zusammensetzen können. Dabei ergeben sich im wesentlichen zwei Probleme, die einer näheren Analyse bedürfen. Diese Probleme wurden von W. S. Brown in seinem Artikel über den euklidischen Algorithmus und den ggT von Polynomen (vgl. [Br71]) bereits gründlich untersucht, so daß wir den Algorithmus hier nicht in allen Einzelheiten darstellen werden, sondern teilweise nur die Resultate zitieren.

Das eine Problem der modularen ggT–Berechnung für Polynome liegt darin, daß der ggT der reduzierten Polynome eventuell einen höherem Grad hat als das Polynom, das man durch Reduktion des ggT in $\mathbb{Z}[x]$ erhält. Ist z.B. $F_1'(x) = (x-1)^3$ und $F_2'(x) = 5(x-1)^2(x-3)$, so gilt über $\mathbb{Z}$: $\mathrm{ggT}(F_1', F_2') = (x-1)^2$, über $\mathbb{F}_5$ ergibt sich hingegen

$\text{ggT}(F_1', F_2') = \text{ggT}(F_1', 0) = (x-1)^3$. Primzahlen für die dieses Problem eintritt, nennen wir *schlecht*. Man kann nun zeigen, daß für das Produkt P der schlechten Primzahlen gilt:

$$P \leq (2\deg(F_1')\max(\|F_1'\|_\infty, \|F_2'\|_\infty)^2)^{\deg(F_1')}, \tag{1.7}$$

falls $\deg(F_1') \geq \deg(F_2')$. Dies ergibt sich aus dem Beweis zu Theorem 2, S. 492 in [Br71], indem man das dortige Resultat für Polynome in einer Variablen spezialisiert.

Das andere Problem der modularen Berechnung besteht darin, daß man als ggT über $\mathbb{Z}$ im allgemeinen kein normiertes Polynom erhält, während wir über $\mathbb{F}_p$ stets normierte Polynome erzwingen und also auch nach dem Zusammensetzen mit dem chinesischen Restsatz stets ein normiertes Polynom erhalten würden, also im allgemeinen kein korrektes Ergebnis. Dieses Problem läßt sich lösen, indem man zunächst den ggT, den man modulo p erhalten hat, jeweils mit einer geeigneten Zahl multipliziert und am Ende des Algorithmus noch einige Multiplikationen vornimmt.

Wir berechnen nun nicht nur den ggT der beiden Polynome F_1' und F_2', sondern auch seine sogenannten *Kofaktoren*, d.h. die Polynome $\frac{F_1'}{\text{ggT}(F_1',F_2')}$ und $\frac{F_2'}{\text{ggT}(F_1',F_2')}$. Konzeptionell erhalten wir also für die Berechnung des ggT zweier Polynome aus $\mathbb{Z}[x]$ folgendes Verfahren:

1.28. Algorithmus (ggT von Polynomen über $\mathbb{Z}$)
Eingabe: Polynome $F_1', F_2' \in \mathbb{Z}[x]$.
Ausgabe: Polynome $G', H_1', H_2' \in \mathbb{Z}[x]$ mit $G' = \text{ggT}(F_1', F_2')$ und $F_i' = H_i'G'$ für $i = 1, 2$.

1. $c_i := \text{cont}(F_i')$; $F_i := \frac{F_i'}{c_i}$ für $i = 1, 2$; /* Bestimmung von Inhalt und primitivem Teil der Polynome */

2. $c := \text{ggT}(c_1, c_2)$; $e := min(\deg(F_1), \deg(F_2))$; /* Inhalt und maximaler Grad des ggT */

3. $f_i := \text{lc}(F_i)$ für $i = 1, 2$; $\bar{g} := \text{ggT}(f_1, f_2)$; /* Maximaler Leitkoeffizient des primitiven Teils des ggT */

4. Für hinreichend viele Primzahlen p DO

   - Berechne mittels Algorithmus 1.26:

     $$\tilde{G}_p = (\bar{g} \bmod p) \, \mathrm{ggT}(((\bar{g} \bmod p)F_1) \bmod p, ((\bar{g} \bmod p)F_2) \bmod p).$$

     /* Um ein Vielfaches des Leitkoeffizienten zu garantieren! */

   - $\tilde{H}_{ip} := \frac{((\bar{g} \bmod p)F_i) \bmod p}{\tilde{G}_p}$ für $i = 1, 2$;

   - IF $\deg(\tilde{G}_p) = 0$ THEN $G := 1$; $H_1 := F_1$; $H_2 := F_2$; GOTO (8) FI;

     /* Polynome haben höchstens gemeinsamen Teiler aus $\mathbb{Z}$ */

   - IF $\deg(\tilde{G}_p) > e$ THEN „wirf dieses modulare Ergebnis weg“ FI;

     /* ggT hat falschen Grad, Teilergebnis uninteressant */

   - IF $\deg(\tilde{G}_p) < e$ THEN $e := \deg(\tilde{G}_p)$; „wirf alle vorherigen Ergebnisse weg“ FI /* Bisher hatte der ggT immer falschen Grad, alle bisherigen Teilergebnisse sind uninteressant */

5. OD;

6. Setze Teilergebnisse mittels chinesischem Restsatz zusammen:

   - Aus den $\tilde{G}_p$ berechne G^*.

   - Aus den $\tilde{H}_{ip}$ berechne H_i^* für $i = 1, 2$.

7. $G := \mathrm{pp}(G^*)$; $g := \mathrm{lc}(G)$; $H_i := \frac{H_i^*}{g}$ für $i = 1, 2$. /* So erhalten wir den */

8. $G' := cG$; $H_i' := \frac{c_i'}{c} H_i$ für $i = 1, 2$. /* richtigen Leitkoeffizienten */

1.29. Bemerkung

Im allgemeinen ist es geschickt, die Zahl der Primzahlen, für die man modulare Berechnungen ausführt, dynamisch zu wählen, da man zu Beginn des Algorithmus nicht wissen kann, wie viele schlechte Primzahlen wirklich auftreten werden. Dabei kann man die berechneten Kofaktoren sinnvoll nutzen! Einen Algorithmus, der so vorgeht, findet man in [Br71], Teil 4.3, S. 489 – 491. Dort findet sich auch eine ausführliche Diskussion der Korrektheit unseres Verfahrens, daher gehen wir darauf hier nicht näher ein. Allerdings ist die dort unter sehr optimistischen Annahmen vorgenommene Laufzeitanalyse falsch. Sie berücksichtigt nämlich nicht, daß beim Zusammensetzen der Teilergebnisse mittels des chinesischen Restsatzes das Rechnen mit Zahlen nötig wird, deren Länge nicht mehr durch die Wortlänge des Rechners begrenzt ist, selbst wenn alle verwendeten Primzahlen sich in einem Wort des Rechners darstelllen lassen. Hier kann man also nicht konstante Laufzeit für eine elementare Operation erwarten. Wenn man hier den Zeitaufwand korrekt analysiert, zeigt sich insbesondere, daß es besser ist, erst alle Ergebnisse zu sammeln und diese dann gemäß Algorithmus 1.9 zusammenzusetzen, statt jedes Teilergebnis sofort mit dem bisherigen Ergebnis zusammenzusetzen. Insbesondere verschlechtert das Verfahren des sofortigen Zusammensetzens nicht nur die Laufzeit, es wird im allgemeinen auch ebensoviel Speicherplatz benötigen, wie das Speichern aller Zwischenergebnisse und das anschließende Zusammensetzen: Wenn man als Moduln Primzahlen wählt, die gerade noch in ein Wort des Rechners passen, wie man es in der Praxis im allgemeinen tut, dann ist die Zahl der von einem Zwischenergebnis belegten Rechnerworte i.a. gleich der Zahl der zu diesem Zwischenergebnis zusammengesetzten Zahlen, man spart also keinen Speicherplatz. (Wenn allerdings manche Zahlen in allen Zwischenergebnissen sehr klein sind, ergibt sich doch eine Ersparnis!) Aus dieser Beobachtung ergibt sich eine evidente Verbesserung des Algorithmus in [Br71].

Wir zeigen hier folgende Aussage über die Laufzeit, die im Gegensatz zu dem Resultat in [Br71] immer gilt, insbesondere auch dann, wenn der Algorithmus die maximal mögliche Zahl schlechter Primzahlen verwendet:

1.30. Satz
Seien F_1' und F_2' Polynome aus $\mathbb{Z}[x]$ vom Grad $\leq n$ und sei c so gewählt, daß gilt: $c \geq \max\left(n, |F_1'|, |F_2'|, |\operatorname{ggT}(F_1', F_2')|, \left|\frac{F_1'}{\operatorname{ggT}(F_1',F_2')}\right|, \left|\frac{F_2'}{\operatorname{ggT}(F_1',F_2')}\right|\right)$. Dann können wir den ggT bzw. die Kofaktoren von F_1' und F_2' in Zeit $O(n^3 \log c \log(n \log c) + n^2 \log^2 c)$ berechnen.

Beweis:
Zunächst ist klar, daß wir alle Berechnungen, die in der Initialisierungsphase und am Ende des Algorithmus notwendig sind, insbeondere die ggT–Berechnungen zur Berechnung der Inhalte, in der angegebenen Zeit ausführen können.

Wiederum läuft der Beweis in der bereits bekannten Art und Weise ab, allerdings müssen wir wegen der schlechten Primzahlen einige zusätzliche Anstrengungen unternehmen. Die Koeffizienten aller Polynome sind durch c beschränkt. Das Produkt der schlechten Primzahlen ist daher nach (1.7) durch $(2nc^2)^n$ beschränkt. Wenn wir also so viele Primzahlen wählen, daß ihr Produkt $\geq 2c(2nc^2)^n$ ist, dann werden nach Entfernen der Ergebnisse modulo schlechter Primzahlen noch genug Teilergebnisse übrigbleiben, um das Resultat eindeutig zu rekonstruieren.

Daher berechnen wir den ggT und seine Kofaktoren zunächst modulo aller Primzahlen $\leq z$, wobei $z = O(\log(2c(2nc^2)^n)) = O(n \log c)$. Dazu benötigt man nach Satz 1.27 und Satz 1.25 jeweils Zeit $O(n^2 \log^2 z)$. Nach (1.4) gibt es $O(\frac{z}{\log z})$ Primzahlen $\leq z$. Somit ergibt sich eine Rechenzeit von

$$O(n^2 z \log z) = O(n^3 \log c \log(n \log c)).$$

Mit Hilfe von Algorithmus 1.28 erhalten wir nun ein Ergebnis, das eindeutig modulo $2c$ ist, wenn z geeignet gewählt wurde, und daraus das Ergebnis in $\mathbb{Z}$. Mit dem chinesischen Restsatz müssen wir dabei wie bisher höchstens $k = O(\frac{\log c}{\log \log c})$ Koeffiziententupel zum Ergebnis zusammensetzen, da die Koeffizienten der Ergebnisse ja betragsmäßig durch c beschränkt sind. Da die Ergebnisse dabei wieder n–Tupel ganzer Zahlen sind, ergibt sich aus Satz 1.10, daß wir hierfür Zeit $O(nk^2 \log^2 z) = O\left(n \frac{\log^2 c}{\log^2 \log c} \log^2(n \log c)\right)$ benötigen.

Insgesamt ergibt sich also eine Laufzeit von

$$\begin{aligned} & O\left(n^3 \log c \log(n \log c) + n \log^2 c \left(\frac{\log n + \log \log c}{\log \log c}\right)^2\right) \\ &= O(n^3 \log c \log(n \log c) + n \log^2 c(1 + \log n)^2) \\ &= O(n^3 \log c \log(n \log c) + n^2 \log^2 c). \end{aligned}$$

$\square$

1.31. Bemerkung

Man beachte insbesondere, daß diese Abschätzung sehr pessimistisch ist. In praktischen Algorithmen wird man die ganz kleinen Primzahlen überhaupt nicht benutzen. Da aber die Wahrscheinlichkeit, daß eine Primzahl p schlecht ist, laut [Br71], Theorem 3, S. 492 durch $\frac{1}{p}$ beschränkt ist, wird man erwarten, daß sich nur ein geringer Bruchteil der verwendeten Primzahlen als schlecht erweist. Insbesondere wird man also erwarten, daß es ausreicht $O\left(\frac{\log c}{\log \log c}\right)$ Primzahlen zu benutzen. Damit erhält man eine erwartete mittlere Laufzeit, die ungefähr um den Faktor n kleiner ist als die soeben bewiesene Laufzeit. In unserer Laufzeitanalyse haben wir nämlich angenommen, daß wir eine maximale Anzahl schlechter Primzahlen treffen, und daher $O\left(n \frac{\log c}{\log(n \log c)}\right)$ Primzahlen benutzen müssen.

1.5 Algebraische Zahlkörper und Ordnungen

Zum Abschluß dieses Kapitels wollen wir noch einige Grundlagen der Theorie der algebraischen Zahlkörper und Ordnungen präsentieren, die wir später benötigen werden. Dazu erklären wir zunächst einige Begriffe und diskutieren dann das Problem der rechnerinternen Darstellung von Ordnungen.

1.5.1 Definitionen

1.32. Definition
Ein Polynom $A(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{Q}[x]$ *heißt* irreduzibel *über* $\mathbb{Q}$, *wenn es keine Polynome* $B(x), C(x) \in \mathbb{Q}[x]$ *von positivem Grad mit* $A(x) = B(x)C(x)$ *gibt.*

Für solch ein irreduzibles Polynom $A(x)$ *ist* $\mathbb{Q}[x]/(A(x)\mathbb{Q}[x])$ *ein Körper, der isomorph ist zu* $\mathbb{Q}(\varrho)$, *wobei* ϱ formale Nullstelle *des Polynoms* A *ist, d.h. es gilt* $\sum_{i=0}^{n} a_i \varrho^i = 0$. *Solch ein Körper* $\mathbb{Q}(\varrho)$ *heißt* algebraischer Zahlkörper, *die Zahlen aus* $\mathbb{Q}(\varrho)$ *heißen* algebraisch, $\frac{A}{\mathrm{lc}(A)}$ *heißt* Minimalpolynom zu ϱ *und* n *heißt* Körpergrad. *Der Körpergrad eines Körpers* $\mathbb{Q}(\varrho)$, *in Zeichen* $[\mathbb{Q}(\varrho) : Q]$ *gibt zugleich an, welche Dimension* $\mathbb{Q}(\varrho)$ *als* $\mathbb{Q}$*−Vektorraum hat.*

1.33. Definition
Eine Zahl, deren Minimalpolynom von der Form $x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ *mit* $a_{n-1}, \ldots, a_0 \in \mathbb{Z}$ *ist, heißt* (algebraisch) ganze Zahl.

Aus der algebraischen Zahlentheorie ist folgendes Resultat bekannt:

1.34. Satz
Die Menge der algebraisch ganzen Zahlen eines algebraischen Zahlkörpers $\mathbb{K}$ *bildet einen Ring, der* $\mathbb{Z}$ *enthält, d.h. Summe und Produkt ganzer Zahlen sind wieder ganz. Dieser Ring heißt* Ganzheitsring *und wird mit* $\mathcal{O}_{\mathbb{K}}$ *bezeichnet. Er ist zugleich freier* $\mathbb{Z}$*-Modul vom Rang des Körpergrads.*

Zum Beweis vgl. etwa [Ne92], Satz (2.2), S. 6. Interessant ist auch der Beweis in [Ba05], S. 4f, der diesen Satz mit elementaren Mitteln beweist und die Ganzheit von Summe und Produkt ganzer Zahlen explizit nachrechnet, statt auf Aussagen über die Struktur des Ganzheitsringes zurückzugreifen.

1.35. Definition
Ein 1*–Oberring* $\mathcal{O} \subset \mathbb{K}$ *von* $\mathbb{Z}$*, der zugleich* $\mathbb{Z}$*–Modul maximalen Rangs aus algebraisch ganzen Zahlen ist, heißt* Ordnung *des algebraischen Zahlkörpers* $\mathbb{K}$.

1.36. Beispiel
Wir betrachten den Körper $\mathbb{K} = \mathbb{Q}(\sqrt{-2}, \sqrt{5})$. Dann ist $\mathbb{Z}[\sqrt{-2}, \sqrt{5}] = \mathcal{O} = \{a+b\sqrt{-2}+c\sqrt{5} + d\sqrt{-10}$ mit $a, b, c, d \in \mathbb{Z}\}$ eine Ordnung von $\mathbb{K}$, denn $\sqrt{-2}, \sqrt{5}$ und $\sqrt{-10}$ sind ganze Zahlen, da ihre Minimalpolynome gerade $x^2 + 2,\ x^2 - 5$ bzw. $x^2 + 10$ sind, und $\mathcal{O}$ bildet offensichtlich einen Oberring von $\mathbb{Z}$, der als $\mathbb{Z}$-Modul den Rang 4 hat, wobei $\{1, \sqrt{-2}, \sqrt{5}, \sqrt{-10}\}$ eine Basis bildet.

Hingegen bildet $\mathcal{O}$ keinen Ganzheitsring, denn $\mathcal{O}$ enthält nicht alle ganzen Zahlen aus $\mathbb{Q}(\sqrt{-2}, \sqrt{5})$: $\frac{\sqrt{5}-1}{2}$ ist Nullstelle des irreduziblen Polynoms $x^2 + x - 1$ und somit ganz. Diese Zahl ist aber nicht in $\mathcal{O}$.

1.5.2 Darstellung von Ordnungen

Darstellung mittels der Basis

Das vorangehende Beispiel zeigt, daß man eine Ordnung durch ihre Basis repräsentieren kann, wobei jedes Basiselement zunächst durch eine komplexe Zahl dargestellt wird. Dabei werden wir im weiteren die Unterscheidung zwischen der Basis (einer Menge von Zahlen) und dem Vektor der Basiselemente unterdrücken, was die Notation erheblich vereinfacht.

Wir wollen nun die Vor– bzw. Nachteile dieser Darstellungsmethode betrachten:

- Die Repräsentation besteht aus einigen wenigen Zahlen.
- Diese sind aber komplexwertig, müssen also für eine interne Darstellung gerundet werden.
- Die Addition von zwei Zahlen ist einfach, allerdings genügt hierfür ohnehin ein Koeffizientenvektor, die Basis wird nicht wirklich benötigt.
- Bei der Multiplikation von Zahlen aus der Ordnung steht man vor unüberwindlichen Schwierigkeiten:
  - Man kann etwa für die beiden Zahlen mittels der Zahlenwerte für die Basis Näherungen in $\mathbb{C}$ berechnen und diese multiplizieren, was aufwendig ist.
  - Nun muß man versuchen, aus der erhaltenen komplexen Zahl eine Darstellung mit Hilfe der Basis zu rekonstruieren. Diese Darstellung kann man jedoch mit nur einer Gleichung nicht eindeutig bestimmen.

Diese Repräsentation ist für theoretische Betrachtungen sehr gut geeignet, da sie leicht handhabbar ist; aufgrund der Schwierigkeiten bei der Multiplikation ist sie jedoch für die Realisierung auf dem Rechner nicht geeignet, wie wir an einem Beispiel illustrieren wollen:

1.37. Beispiel (Fortsetzung von Beispiel 1.36)
Wir müssen die komplexen Zahlen runden, z.B. auf zwei Nachkommastellen im Binärsystem. Damit wird unsere Basis $(1, \sqrt{-2}, \sqrt{5}, \sqrt{-10})$ repräsentiert durch den Vektor $(1, 1.5i, 2.25, 3.25i)$. Wenn wir jetzt eine Multiplikation ausführen, z.B. $\sqrt{-2}\sqrt{-2} = -2$, so berechnen wir im Rechner $1.5i \cdot 1.5i = -2.25$. Dies ist in unserer Repräsentation gerade $-\sqrt{5}$, wir erhalten also wegen der Rundungsprobleme ein falsches Ergebnis. Aber selbst wenn wir richtig runden und als Ergebnis -2 erhalten, wissen wir nicht, ob hiermit beispielsweise $-2 \cdot 1$, $7 \cdot 1 - 4 \cdot 2.25$ oder $-11 \cdot 1 + 4 \cdot 2.25$ gemeint ist.

Daher wählen wir eine etwas andere Darstellung für die Repräsentation der Basis im Rechner: Wir stellen eine algebraische Zahl nicht einfach durch eine approximierte komplexe Zahl dar, sondern durch einen Vektor aus Approximationen an alle Nullstellen ihres Minimalpolynoms.

Darstellung mittels Konjugiertenvektoren

Im weiteren benutzen wir die folgenden Bezeichnungen:

1.38. Definition

Sei ϱ eine algebraische Zahl und $A(x) = \sum_{i=0}^{n} a_i x^i$ das zugehörige Minimalpolynom. Dann heißen die n Nullstellen $\varrho_1, \dots \varrho_n$ des Polynoms zueinander konjugiert. *Die Abbildungen, die $\mathbb{Q}$ auf $\mathbb{Q}$ und ϱ auf eine Nullstelle ϱ_i des Polynoms abbilden, können jeweils zu homomorphen* Einbettungen *σ_i von $\mathbb{Q}(\varrho)$ in $\mathbb{C}$ fortgesetzt werden.*

Für den späteren Gebrauch definieren wir noch einige weitere Begriffe:

1.39. Definition

Sei ϱ eine algebraische Zahl mit $[\mathbb{Q}(\varrho) : \mathbb{Q}] = n$ und seien $\sigma_1, \dots, \sigma_n$ die n Einbettungen von $\mathbb{Q}(\varrho)$ in $\mathbb{C}$. Für ein Element $\alpha \in \mathbb{Q}(\varrho)$ bezeichnen wir dann mit $N(\alpha) := \prod_{i=1}^{n} \sigma_i(\alpha)$ die Norm *von α (in $\mathbb{Q}(\varrho)$). Das Polynom $c_\alpha(x) := \prod_{i=1}^{n} (x - \sigma_i(\alpha))$ nennen wir* charakteristisches Polynom *von α.*

Dann gilt der folgende, aus der Algebra bekannte Satz:

1.40. Satz

Sei $\mathbb{Q}(\varrho)$ ein Erweiterungskörper von $\mathbb{Q}$ vom Grad n und sei $\alpha \in \mathbb{Q}(\varrho)$ mit Minimalpolynom m_α vom Grad k. Dann gilt:

- *Für das charakteristische Polynom c_α von α gilt: $c_\alpha = m_\alpha^{\frac{n}{k}} \in \mathbb{Q}[x]$, d.h. insbesondere: k teilt n.*
- *Für den letzten Koeffizienten $c_{\alpha,0}$ des charakteristischen Polynoms gilt: $c_{\alpha,0} = (-1)^n N(\alpha)$. Insbesondere liegt also $N(\alpha)$ stets in $\mathbb{Z} \setminus \{0\}$, wenn α eine algebraisch ganze Zahl $\neq 0$ ist.*

Beweis:

Für die erste Behauptung findet sich ein Beweis beispielsweise in [Ne92] im Beweis zu Satz 2.6, S. 9f.

Nach Definition 1.39 ist klar, daß die Norm von α bis auf das Vorzeichen der letzte Koeffizient des charakteristischen Polynoms von α ist. Da α eine algebraisch ganze Zahl ist, hat das Minimalpolynom von α Koeffizienten in $\mathbb{Z}$, wobei der letzte Koeffizient nicht 0 sein kann, sonst hätte das Minimalpolynom einen Faktor x. Da das charakteristische Polynom eine Potenz des Minimalpolynoms ist, hat es also ebenfalls Koeffizienten in $\mathbb{Z}$, wobei wiederum der letzte Koeffizient von 0 verschieden ist. Also ergibt sich die zweite Behauptung. □

Nun stellen wir die Basiselemente mittels sogenannter *Konjugiertenvektoren* dar, die Darstellung einer Zahl ϱ ist also von der Form $\underline{\varrho} = (\sigma_1(\varrho), \ldots, \sigma_n(\varrho))^T \in \mathbb{C}^n$. Dabei unterscheiden wir auch hier nicht zwischen der Basis und der Matrix, deren Spaltenvektoren die Basisvektoren sind. In dieser Darstellung ist jetzt theoretisch auch die Multiplikation möglich:

- Zwei Konjugiertenvektoren werden komponentenweise miteinander multipliziert, wobei Multiplikationen in $\mathbb{C}$ ausgeführt werden.
- Anschließend kann man aus dem Ergebnisvektor in $\mathbb{C}^n$ rekonstruieren, welche Linearkombination der Basisvektoren zu diesem Ergebnis führt, allerdings müssen Rundungsprobleme berücksichtigt werden. Waren die beiden Konjugiertenvektoren vorher ausreichend exakt approximiert, so führt dieses Gleichungssystem zur richtigen Lösung.

Diese Darstellung enthält allerdings eine gewisse Redundanz: Da die Minimalpolynome algebraisch ganzer Zahlen nur Koeffizienten aus $\mathbb{Z}$ enthalten, ist mit jeder komplexen Zahl auch die konjugiert–komplexe Zahl eine Nullstelle. Wenn wir also die Einbettungen geeignet sortieren, so erhalten wir folgenden Aufbau des Konjugiertenvektors:

$$\underline{\varrho} = (\sigma_1(\varrho), \ldots, \sigma_r(\varrho), \sigma_{r+1}(\varrho), \ldots, \sigma_{r+\frac{n-r}{2}}(\varrho), \overline{\sigma_{r+1}(\varrho)}, \ldots, \overline{\sigma_{r+\frac{n-r}{2}}(\varrho)})^T,$$

wobei $\sigma_1, \ldots, \sigma_r$ reellwertige Einbettungen und $\sigma_{r+1}, \ldots, \sigma_{r+\frac{n-r}{2}}$ komplexwertige Einbettungen sind und mit $\overline{x}$ das Konjugiert–komplexe von x bezeichnet wird.

Damit können wir also unsere Repräsentation etwas vereinfachen, indem wir die konjugiert–komplexen Einbettungen weglassen. Damit erhalten wir:

$$\underline{\varrho} = (\sigma_1(\varrho), \ldots, \sigma_r(\varrho), \Re(\sigma_{r+1}(\varrho)), \Im(\sigma_{r+1}(\varrho)), \ldots, \Re(\sigma_{r+\frac{n-r}{2}}(\varrho)), \Im(\sigma_{r+\frac{n-r}{2}}(\varrho)))^T \in \mathbb{R}^n.$$

Hierbei ist zu beachten, daß die zuvor komponentenweise erfolgende Multiplikation über $\mathbb{C}$ jetzt durch eine etwas kompliziertere Multiplikation über $\mathbb{R}$ ersetzt wird.

Im weiteren bezeichnen wir mit $\underline{\alpha}$ den Konjugiertenvektor (in der vereinfachten Repräsentation) einer algebraischen Zahl α, d.h. wir definieren:

1.41. Definition
Sei $\mathbb{K}$ ein algebraischer Zahlkörper. Dann definieren wir den Homomorphismus „$\underline{\cdot}$“ wie folgt:

$$\underline{\cdot} : \mathbb{K} \longrightarrow \mathbb{R}^n,$$

$$\alpha \longmapsto \underline{\alpha} = (\sigma_1(\alpha), \ldots, \sigma_r(\alpha), \Re(\sigma_{r+1}(\alpha)), \Im(\sigma_{r+1}(\alpha)), \ldots, \Re(\sigma_{r+\frac{n-r}{2}}(\alpha)), \Im(\sigma_{r+\frac{n-r}{2}}(\alpha)))^T,$$

wobei die Addition auf $\mathbb{R}^n$ komponentenweise definiert ist und die Multiplikation wie folgt:

- *Die ersten r Komponenten werden komponentenweise multipliziert.*
- *Die folgenden $n - r$ Komponenten werden im Sinne der Multiplikation in $\mathbb{C}$ paarweise miteinander multipliziert.*

Für eine Basis $\Omega = (\omega_1, \ldots, \omega_n)$ einer Ordnung bezeichnen wir die Repräsentation der Basis mittels Konjugiertenvektoren mit $\underline{\Omega} := (\underline{\omega}_1, \ldots, \underline{\omega}_n)$.

Wenn wir die Basis explizit im Rechner repräsentieren müssen, werden wir diese Methode benutzen, um die einzelnen Basiselemente darzustellen. Inwieweit man mit dieser Methode multiplizieren kann, wollen wir wiederum an einem Beispiel demonstrieren:

1.42. Beispiel (Fortsetzung von Beispiel 1.37)
Wenn wir wieder mit zwei Nachkommastellen im Binärsystem arbeiten, repräsentieren wir nun unsere Basis $(1, \sqrt{-2}, \sqrt{5}, \sqrt{-10})$ durch

$$\begin{pmatrix} 1 & 0 & 2.25 & 0 \\ 0 & 1.5 & 0 & 3.25 \\ 1 & 0 & -2.25 & 0 \\ 0 & 1.5 & 0 & -3.25 \end{pmatrix}$$

Wenn wir in dieser Darstellung multiplizieren, z.B. $\sqrt{-2}\sqrt{-2} = -2$, so berechnen wir intern: $(0, 1.5, 0, 1.5)^T(0, 1.5, 0, 1.5)^T = (-2.25, 0, -2.25, 0)^T$. Hier sieht man sofort, daß

$-\sqrt{5}$ nicht als Lösung in Frage kommt. Wenn wir richtig runden, müssen wir jetzt das Gleichungssystem mit der angegebenen Matrix und rechter Seite $(-2, 0, -2, 0)$ lösen und erhalten eindeutig das korrekte Ergebnis.

Darstellung mittels der Multiplikationstafel

Für das praktische Rechnen innerhalb der Ordnung stört an der vorangehenden Darstellung allerdings der große Aufwand, den man treiben muß, um zwei Zahlen zu multiplizieren. Hierzu muß nämlich stets ein Gleichungssystem gelöst werden. Deshalb stellen wir noch eine alternative Darstellungsmethode vor, in der sich schnell multiplizieren läßt. Betrachten wir zunächst eine Multiplikation in einer Ordnung mit Basis $\omega_1, \ldots, \omega_n$:

$$\sum_{i=1}^{n} a_i\omega_i \sum_{j=1}^{n} b_j\omega_j = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_ib_j)(\omega_i\omega_j)$$

Wenn wir also wissen, was das Produkt von zwei Basiselementen ist, können wir leicht multiplizieren. Daher berechnen wir Zahlen w_{ijk} aus $\mathbb{Z}$ mit

$$\omega_i\omega_j = \sum_{k=1}^{n} w_{ijk}\omega_k.$$

Diese Werte speichern wir in einer Tabelle, der sogenannten *Multiplikationstafel*, mit deren Hilfe wir nun multiplizieren können. In unserem obigen Beispiel (vgl. Beispiel 1.42) genügt dann ein einfaches Nachschlagen in dieser Tafel um das Ergebnis der Multiplikation zu bestimmen!

Die Multiplikationstafel kann man offensichtlich aus der Basis konstruieren, und da wir es mit einer Ordnung zu tun haben, und also das Produkt zweier Elemente wieder eine $\mathbb{Z}$−Linearkombination der Basisvektoren ergeben muß, sind die Einträge dieser Tabelle in der Tat Zahlen aus $\mathbb{Z}$, die jedoch im allgemeinen sehr groß sein können. Diese Multiplikationstafel ist offensichtlich abhängig von der Wahl der Basis, daher definieren wir:

1.43. Definition

Sei $\Omega = (\omega_1, \ldots, \omega_n)$ eine Basis einer Ordnung $\mathcal{O}$. Dann bezeichnen wir mit $\mathrm{MT}(\Omega)$ *die* Multiplikationstafel *zu dieser Basis. Mit* $|\mathrm{MT}(\Omega)|$ *bezeichnen wir den maximalen Eintrag in der Multiplikationstafel.*

Für die höhere Geschwindigkeit bei der Multiplikation bezahlt man allerdings durch einen größeren Speicherbedarf: Statt n^2 reeller Zahlen in der Darstellung der Basis durch Konjugiertenvektoren benötigt man jetzt n^3 ganze Zahlen, um die Multiplikationstabelle zu speichern.

Zum Schluß wollen wir nochmals die drei Möglichkeiten der Darstellung einer Ordnung zusammenfassen:

Zusammenfassung der verschiedenen Darstellungen

- Zunächst können wir die Ordnung durch eine Basis Ω aus komplexen Zahlen darstellen. Diese Darstellung ist nur für theoretische Betrachtungen interessant, hierfür werden wir sie aber tatsächlich verwenden.
- Wir können die Basiselemente auch durch Konjugiertenvektoren darstellen, dann erhalten wir eine Darstellung der Ordnung mittels der Matrix $\underline{\Omega}$, deren Einträge gerundete reelle Zahlen sind. Diese Darstellung benötigen wir für spezielle Aufgaben, die eine explizite Darstellung der Basis benötigen.
- Schließlich können wir auch die Multiplikationstafel $\mathrm{MT}(\Omega)$ verwenden, mit der das Rechnen innerhalb der Ordnung am einfachsten ist.

Die Darstellung mit Hilfe der Multiplikationstafel wählen wir von nun an als Standarddarstellung. In dieser Form sollen Ein– und Ausgaben erfolgen. Da man die Basis aus der Multiplikationstabelle rekonstruieren kann, wie wir später noch sehen werden, führt dies zu keiner Einschränkung. Solange nur elementare Operationen auf der Ordnung ausgeführt werden, rechnen wir also mit der Darstellung durch die Multiplikationstabelle. Bei Bedarf können wir dann zur Darstellung mittels der Basis (repräsentiert durch Konjugiertenvektoren) wechseln.

Kapitel 2

Approximationen

2.1 Definitionen

Reelle Zahlen können wir im Rechner nicht exakt darstellen, wir müssen sie also runden. Wie dies geschieht, wollen wir im folgenden erklären.

2.1. Definition (Approximationen)
Sei $q \in \mathbb{N}$. Für eine reelle Zahl a bezeichnen wir mit $a\{q\}$ eine Zahl aus $2^{-q}\mathbb{Z}$, für die gilt:

$$|a - a\{q\}| \leq 2^{-q-1}.$$

q heißt dabei Präzision *oder* Genauigkeit *der Approximation.*

Wenn man zudem noch eine Konvention trifft, die regelt, welche Zahl aus $2^{-q}\mathbb{Z}$ man auswählt, falls $a \in 2^{-q}\mathbb{Z} + 2^{-q-1}$, dann ist für jede Zahl $a \in \mathbb{R}$ $a\{q\}$ eindeutig bestimmt. Diese Konvention kann z.B. so aussehen, daß man in diesen Grenzfällen stets aufrundet.

Für eine komplexe Zahl $c = c_1 + ic_2$ mit $c_1, c_2 \in \mathbb{R}$ bezeichnen wir dann mit $c\{q\}$ die komplexe Zahl, die man durch Approximation von Real- und Imaginärteil erhält, d.h. $c\{q\} = c_1\{q\} + ic_2\{q\}$.

Für einen Vektor $\underline{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ bezeichnen wir mit $\underline{a}\{q\}$ den Vektor aus den Näherungswerten an die Komponenten, d.h. den Vektor $(a_1\{q\}, \ldots, a_n\{q\})$.

Für eine Matrix $A = (\underline{a}_1, \ldots, \underline{a}_m) = (a_{ij})_{\substack{1 \le i \le n \\ 1 \le j \le m}} \in \mathbb{R}^{n \times m}$ *bezeichnen wir mit* $A\{q\}$ *die Matrix* $(a_{ij}\{q\})_{\substack{1 \le i \le n \\ 1 \le j \le m}}$.

Später wollen wir wichtige Abschätzungen für die Qualität der Ergebnisse, die man beim Rechnen mit Approximationen erhält, beweisen. Dazu benötigen wir einige weitere Begriffe:

2.2. Definition

Sei $A \in \mathbb{R}^{n \times m}$. *Dann bezeichnen wir mit* $\lambda_i(A)$ *die (euklidische) Länge des i.-kleinsten Spaltenvektors von A, d.h.*

$$\lambda_i(A) = \min\{r \in \mathbb{R}_{\ge 0} : \textit{es gibt } i \textit{ Spaltenvektoren von } A \textit{ der Länge} \le r\}.$$

Schließlich bezeichnen wir für eine Matrix $A \in \mathrm{GL}(n, \mathbb{R})$ *mit* $\mathrm{dft}(A)$ *den* Defekt *von* A, *d.h.*

$$\mathrm{dft}(A) := \frac{\prod_{i=1}^{n} \|\underline{a}_i\|}{|\det A|} \ .$$

Die Bedeutung des Defekts wollen wir hier noch etwas näher erläutern: Die Zahl $\mathrm{dft}(A)$ ist ein Maß dafür, wie orthogonal die Spaltenvektoren von A sind. Aufgrund der Hadamardschen Ungleichung (1.1) gilt genauer: Es ist $\mathrm{dft}(A) = 1$ genau dann, wenn die Spaltenvektoren orthogonal sind, sonst ist $\mathrm{dft}(A) > 1$.

Wenn wir also eine Matrix aus $\mathbb{R}^{n \times n}$ orthogonalisieren wollen und dies mit einer Näherungsmatrix statt mit der exakten Matrix ausführen, dann wird in die Abschätzung für den Fehler, der dabei entsteht, in natürlicher Weise der Defekt einfließen, denn je weiter die Ausgangsmatrix davon entfernt war, orthogonal zu sein, d.h. je größer der Defekt ist, desto stärker kann sich der anfängliche Rundungsfehler auswirken.

Man kann den Defekt auch benutzen, um eine Abschätzung für die Einträge der inversen Matrix zu erhalten:

2.3. Satz (Inverse Matrix)

Sei $A = (a_{ij})_{1 \le i,j \le n} \in \mathrm{GL}(n, \mathbb{R})$, *also invertierbar, und sei* $A^{-1} = (b_{ij})_{1 \le i,j \le n}$ *die zugehörige inverse Matrix. Dann ist* $|b_{ij}| \le \frac{\mathrm{dft}(A)}{\lambda_1(A)}$ *und* $\|A^{-1}\| \le n \frac{\mathrm{dft}(A)}{\lambda_1(A)}$.

Beweis:
Aus der Kramerschen Regel folgt wie in (1.2):

$$\begin{aligned}|b_{ij}| &\leq \frac{\prod\limits_{\substack{k=1\\k\neq i}}^{n} \|\underline{a}_k\|}{|\det(A)|}\\ &\leq \frac{\prod\limits_{k=1}^{n} \|\underline{a}_k\|}{\lambda_1(A)|\det(A)|}\\ &= \frac{\operatorname{dft}(A)}{\lambda_1(A)}\ .\end{aligned}$$

Daraus ergibt sich:

$$\begin{aligned}\|A^{-1}\| &\leq \sqrt{n^2\left(\max_{1\leq i,j\leq n} |b_{ij}|\right)^2}\\ &\leq n\frac{\operatorname{dft}(A)}{\lambda_1(A)}\ .\end{aligned}$$

□

2.2 Qualität von Approximationen

Häufig muß man abschätzen, wie gut eine Approximation gegebener Genauigkeit die exakte Eingabe annähert. Solche Abschätzungen liefert der folgende Satz:

2.4. Satz
Für komplexe Zahlen c gilt:

$$|c - c\{q\}| < 2^{-q}. \tag{2.1}$$

Für Vektoren $\underline{a} \in \mathbb{R}^n$ gelten folgende Abschätzungen:

$$\|\underline{a} - \underline{a}\{q\}\| \leq \sqrt{n}2^{-q-1} \tag{2.2}$$

und

$$\|\underline{a} - \underline{a}\{q\}\|_\infty \leq 2^{-q-1}. \tag{2.3}$$

Für Matrizen $A \in \mathbb{R}^{n\times n}$ ergibt sich:

$$\|A - A\{q\}\| \leq n2^{-q-1} \tag{2.4}$$

sowie

$$\|A - A\{q\}\|_\infty \leq \sqrt{n}2^{-q-1}. \tag{2.5}$$

Beweis:
(2.1) folgt unmittelbar aus der Definition, es ist nämlich

$$|c - c\{q\}| = \sqrt{|\Re(c) - \Re(c)\{q\}|^2 + |\Im(c) - \Im(c)\{q\}|^2} \leq \sqrt{2}2^{-q-1} < 2^{-q}.$$

Aus

$$\|\underline{a} - \underline{a}\{q\}\| = \sqrt{\sum_{i=1}^{n}(a_i - a_i\{q\})^2} \leq \sqrt{n(2^{-q-1})^2} = \sqrt{n}2^{-q-1}$$

folgt (2.2).

Mit

$$\|\underline{a} - \underline{a}\{q\}\|_\infty = \max_{1\leq i\leq n} |a_i - a_i\{q\}| \leq 2^{-q-1}$$

ist (2.3) gezeigt.

Wegen

$$\|A - A\{q\}\| = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij} - a_{ij}\{q\})^2} \leq \sqrt{n^2(2^{-q-1})^2} = n2^{-q-1}$$

gilt (2.4).

Aufgrund von

$$\|A - A\{q\}\|_\infty = \max_{1\leq i\leq n} \sqrt{\sum_{j=1}^{n}(a_{ij} - a_{ij}\{q\})^2} \leq \max_{1\leq i\leq n} \sqrt{n(2^{-q-1})^2} = \sqrt{n}2^{-q-1}$$

erhalten wir (2.5). □

2.3 Fehleranalysen

Beim Rechnen mit Matrizen aus $\mathbb{R}^{n\times n}$ ist es erforderlich, Approximationen zu verwenden. Dadurch erhält man natürlich verfälschte Ergebnisse, etwa für die Determinante oder die inverse Matrix. Wir wollen nun untersuchen, inwieweit diese Ergebnisse die tatsächlich gesuchten Werte annähern. Dabei zeigen wir zunächst folgendes Resultat:

2.5. Satz (Determinante der Approximationsmatrix)
Sei $A \in \mathbb{R}^{n\times n}$ gegeben. Dann gilt:

$$|\det(A\{q\}) - \det(A)| \leq \|\underline{a}_1\| \cdots \|\underline{a}_n\| \left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right).$$

Beweis:

$$\begin{aligned}
|\det(A\{q\}) - \det(A)| &\leq \sum_{j=1}^{n} |\det(\underline{a}_1, \ldots, \underline{a}_{j-1}, \underline{a}_j\{q\} - \underline{a}_j, \underline{a}_{j+1}\{q\}, \ldots, \underline{a}_n\{q\})| \\
&\leq \sum_{j=1}^{n} \prod_{i=1}^{j-1} \|\underline{a}_i\| \|\underline{a}_j\{q\} - \underline{a}_j\| \prod_{i=j+1}^{n} \|\underline{a}_i\{q\}\| \text{ nach (1.1)} \\
&\leq \sum_{j=1}^{n} \prod_{i=1}^{j-1} \|\underline{a}_i\| 2^{-q}\sqrt{n} \prod_{i=j+1}^{n} \left(\|\underline{a}_i\| + 2^{-q}\sqrt{n}\right) \text{ nach (2.2)} \\
&= 2^{-q}\sqrt{n} \sum_{j=1}^{n} \prod_{\substack{i=1 \\ i\neq j}}^{n} \|\underline{a}_i\| \prod_{i=j+1}^{n} \left(1 + \frac{2^{-q}\sqrt{n}}{\|\underline{a}_i\|}\right) \\
&\leq 2^{-q}\sqrt{n} \sum_{j=1}^{n} \frac{\|\underline{a}_1\| \cdots \|\underline{a}_n\|}{\lambda_1(A)} \left(1 + \frac{2^{-q}\sqrt{n}}{\lambda_1(A)}\right)^{n-j} \\
&= 2^{-q}\sqrt{n} \frac{\|\underline{a}_1\| \cdots \|\underline{a}_n\|}{\lambda_1(A)} \sum_{j=0}^{n-1} \left(1 + \frac{2^{-q}\sqrt{n}}{\lambda_1(A)}\right)^{j} \\
&= 2^{-q}\sqrt{n} \frac{\|\underline{a}_1\| \cdots \|\underline{a}_n\|}{\lambda_1(A)} \frac{\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1}{2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}} \\
&= \|\underline{a}_1\| \cdots \|\underline{a}_n\| \left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right) .
\end{aligned}$$

□

2.6. Bemerkung

Genauer ergibt sich mit (2.2) sogar die etwas schärfere Abschätzung

$$|\det(A\{q\}) - \det(A)| \leq \|\underline{a}_1\| \cdots \|\underline{a}_n\| \left(\left(1 + 2^{-q-1}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right) .$$

Da es für die asymptotische Laufzeit unserer Algorithmen aber keine Rolle spielt, ob wir den Wert von q um 1 erhöhen oder nicht, begnügen wir uns mit der etwas schwächeren Abschätzung, um die folgenden Formeln etwas übersichtlicher zu halten.

2.7. Korollar

Sei $A \in \mathbb{R}^{n\times n}$. Dann gilt:

$$|\det(A\{q\})| \leq |\det(A)| \left(1 + \mathrm{dft}(A)\left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right)\right) .$$

Beweis:

Aus Satz 2.5 folgt zunächst durch Erweitern mit $|\det(A)|$:

$$|\det(A\{q\}) - \det(A)| \leq |\det(A)|\,\mathrm{dft}(A) \left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right) .$$

Indem wir die Dreiecksungleichung anwenden, ergibt sich die Behauptung. □

2.8. Bemerkung
Analog kann man eine Abschätzung in die umgekehrte Richtung zeigen:

$$|\det(A\{q\})| \geq |\det(A)| \left(1 - \mathrm{dft}(A) \left(\left(1 + 2^{-q} \frac{\sqrt{n}}{\lambda_1(A)}\right)^n - 1\right)\right) \quad .$$

Im folgenden werden wir nur den Fall regulärer Matrizen benötigen. Wenn wir eine Transformation einer Matrix $A \in \mathrm{GL}(n, \mathbb{R})$ ausführen, z.B. mittels einer Matrix $T \in \mathbb{Z}^{n\times n}$, dann berechnen wir in Wirklichkeit $A\{q\}T$. Wir wollen daher $\|A\underline{t}_i - A\{q\}\underline{t}_i\|$ abschätzen.

Dazu benötigen wir die folgende Hilfsbehauptung:

2.9. Lemma (Inverse Matrix der Approximationsmatrix)
Sei A eine Matrix aus $\mathrm{GL}(n, \mathbb{R})$ und $q \in \mathbb{N}$ mit $q \geq 2\log n + log\,\mathrm{dft}(A) - \log \lambda_1(A)$. Dann ist auch $A\{q\} \in \mathrm{GL}(n, \mathbb{R})$ und es gilt:

$$\|(A\{q\})^{-1}\| \leq 2n \frac{\mathrm{dft}(A)}{\lambda_1(A)} \quad .$$

Beweis:
Betrachte $A^{-1}A\{q\}$. Dann ist

$$\begin{aligned} \|I - A^{-1}A\{q\}\| &\leq \|A^{-1}\|\|A - A\{q\}\| \\ &\leq \|A^{-1}\| n 2^{-q-1} \text{ nach Satz 2.4 (2.4)} \\ &\leq n^2 \frac{\mathrm{dft}(A)}{\lambda_1(A)} 2^{-q-1} \text{ nach Satz 2.3.} \end{aligned}$$

Nach Wahl von q ist also $\|I - A^{-1}A\{q\}\| < 1$. Damit ist $\sum_{l=0}^{\infty}(I - A^{-1}A\{q\})^l$ eine (absolut) konvergente Reihe und

$$\sum_{l=0}^{\infty}(I - A^{-1}A\{q\})^l = \left(I - \left(I - A^{-1}A\{q\}\right)\right)^{-1} = \left(A^{-1}A\{q\}\right)^{-1},$$

d.h. $A^{-1}A\{q\}$ ist invertierbar. Da A nach Voraussetzung invertierbar ist, ist also auch $AA^{-1}A\{q\} = A\{q\}$ invertierbar. Somit existiert $(A\{q\})^{-1}$ und es gilt:

$$\|A^{-1} - (A\{q\})^{-1}\| \leq \|(A\{q\})^{-1}\|\|A\{q\} - A\|\|A^{-1}\|.$$

Daraus ergibt sich:

$$\begin{aligned}
\|(A\{q\})^{-1}\| &\le \|A^{-1}\| + \|(A\{q\})^{-1}\|\|A\{q\} - A\|\|A^{-1}\| \\
&\le \|A^{-1}\| \left(1 + \|(A\{q\})^{-1}\| n 2^{-q-1}\right) \text{ nach Satz 2.4} \\
&\le n \frac{\mathrm{dft}(A)}{\lambda_1(A)} \left(1 + \|(A\{q\})^{-1}\| n 2^{-q-1}\right) \text{ nach Satz 2.3.}
\end{aligned}$$

Somit folgt:

$$\|(A\{q\})^{-1}\| \left(1 - n^2 2^{-q-1} \frac{\mathrm{dft}(A)}{\lambda_1(A)}\right) \le n \frac{\mathrm{dft}(A)}{\lambda_1(A)}$$

und also:

$$\|(A\{q\})^{-1}\| \le \frac{n \,\mathrm{dft}(A)}{\lambda_1(A) - n^2 2^{-q-1}\,\mathrm{dft}(A)} \ .$$

Nach Wahl von q ist aber $2^q \ge n^2 \frac{\mathrm{dft}(A)}{\lambda_1(A)}$, also $\frac{\lambda_1(A)}{2} \ge 2^{-q-1} n^2 \,\mathrm{dft}(A)$, also $\lambda_1(A) - n^2 2^{-q-1}\,\mathrm{dft}(A) \ge \frac{\lambda_1(A)}{2}$. Damit ergibt sich nun:

$$\|(A\{q\})^{-1}\| \le 2n \frac{\mathrm{dft}(A)}{\lambda_1(A)} \ . \qquad \square$$

Damit können wir jetzt abschätzen, wie groß der Fehler ist, der dadurch entsteht, daß wir anstelle der Matrix $A \in \mathrm{GL}(n, \mathbb{R})$ die Matrix $A\{q\}$ transformieren:

2.10. Satz (Transformation der Approximationsmatrix)
Sei $A \in \mathrm{GL}(n, \mathbb{R})$, $T = (\underline{t}_1, \ldots, \underline{t}_n) \in \mathbb{Z}^{n \times n}$ sowie $q \in \mathbb{N}$ mit $q \ge 2 \log n + \log \mathrm{dft}(A) - \log \lambda_1(A)$. Dann gilt:

$$\|A\underline{t}_i\| \le \|A\{q\}\underline{t}_i\| \left(1 + 2^{-q} n^2 \frac{\mathrm{dft}(A)}{\lambda_1(A)}\right) \ .$$

Beweis:
Klar ist, daß die Abschätzung gilt, falls $\underline{t}_i$ der Nullvektor ist, daher nehmen wir nun an, daß $\underline{t}_i \ne (0, \ldots, 0)$ gilt. Dann gilt:

$$\|A\underline{t}_i - A\{q\}\underline{t}_i\| \le \|A - A\{q\}\|\|\underline{t}_i\|,$$

und daher ergibt sich

$$\begin{aligned}
\|A\underline{t}_i\| &\le \underbrace{\|A - A\{q\}\|}_{\le 2^{-q-1} n \text{ nach Satz 2.4}} \|\underline{t}_i\| + \|A\{q\}\underline{t}_i\| \\
&\le \|A\{q\}\underline{t}_i\| \left(1 + 2^{-q-1} n \frac{\|\underline{t}_i\|}{\|A\{q\}\underline{t}_i\|}\right) \ . \qquad (2.6)
\end{aligned}$$

Diese Abschätzung ist zulässig, da nach Lemma 2.9 auch $A\{q\} \in \mathrm{GL}(n, \mathbb{R})$ ist und daher aus $\|\underline{t}_i\| > 0$ folgt, daß auch $\|A\{q\}\underline{t}_i\| > 0$ gilt. Wegen Lemma 2.9 ist $A\{q\}$ auch invertierbar, damit ergibt sich:

$$\begin{aligned} \|(A\{q\})^{-1}\|\|A\{q\}\underline{t}_i\| &\geq \|\underline{t}_i\| \\ \Longleftrightarrow \|(A\{q\})^{-1}\| &\geq \frac{\|\underline{t}_i\|}{\|A\{q\}\underline{t}_i\|} \ . \end{aligned} \tag{2.7}$$

Wenn wir nun (2.7) in (2.6) einsetzen erhalten wir:

$$\begin{aligned} \|A\underline{t}_i\| &\leq \|A\{q\}\underline{t}_i\|(1 + 2^{-q-1}n \underbrace{\|(A\{q\})^{-1}\|}_{\leq 2n\frac{\mathrm{dft}(A)}{\lambda_1(A)} \text{ nach Lemma 2.9}}) \\ &\leq \|A\{q\}\underline{t}_i\| \left(1 + 2^{-q}n^2 \frac{\mathrm{dft}(A)}{\lambda_1(A)}\right) \ . \end{aligned}$$

□

Kapitel 3

Gitter

3.1 Definitionen

3.1. Definition (Gitter)
Eine Menge $L \subset \mathbb{R}^n$ mit

$$L = \{x_1\underline{b}_1 + \ldots + x_k\underline{b}_k : x_i \in \mathbb{Z},\ \underline{b}_i \in \mathbb{R}^n,\ \underline{b}_1, \ldots, \underline{b}_k \text{ linear unabhängig über } \mathbb{R}\}$$

heißt Gitter. $\underline{b}_1, \ldots, \underline{b}_k$ *heißt* Basis *des Gitters L, wobei wir auch hier wieder die eigentliche Basis $\{\underline{b}_1, \ldots, \underline{b}_k\}$ und die Matrix $(\underline{b}_1, \ldots, \underline{b}_k)$ identifizieren wollen.*

Eine für die Praxis wichtige, alternative Charakterisierung eines Gitters ergibt sich aus dem folgenden Satz:

3.2. Satz
Eine additive Untergruppe $L \subset \mathbb{R}^n$ ist genau dann ein Gitter, wenn gilt:

$$\inf_{0 \neq \underline{x} \in L} \|\underline{x}\| > 0. \tag{3.1}$$

Beweis:
Sei zunächst ein Gitter L mit Basis $\underline{b}_1, \ldots, \underline{b}_k$ gegeben. Dann ist klar, daß L eine additive Untergruppe des $\mathbb{R}^n$ ist. Würde nun (3.1) nicht gelten, so gäbe es Vektoren $\underline{x}^{(n)}$ mit

$$0 \neq \|\underline{x}^{(n)}\| < \frac{1}{n}, \quad \underline{x}^{(n)} = \sum_{i=1}^{k} a_i^{(n)}\underline{b}_i, \quad a_i^{(n)} \in \mathbb{Z} \text{ für alle } 1 \leq i \leq k,\ n \in \mathbb{N},$$

wobei für jedes $n \in \mathbb{N}$ mindestens einer der Koeffizienten $a_1^{(n)}, \ldots, a_k^{(n)}$ von Null verschieden wäre. Indem wir jeweils durch den betragsgrößten Koeffizienten dividieren — insbesondere also *nicht* durch 0 —, erhalten wir Linearkombinationen mit

$$\left\| \sum_{i=1}^{k} \bar{a}_i^{(n)} \underline{b}_i \right\| < \frac{1}{n}, \quad \bar{a}_i^{(n)} \in [-1,1] \text{ für alle } 1 \leq i \leq k, n \in \mathbb{N}.$$

Da jede beschränkte Folge einen Häufungspunkt haben muß und also eine konvergente Teilfolge enthält, können wir durch wiederholte Teilfolgenauswahl eine Indexfolge $\vartheta(n)$ erhalten (d.h. es gibt eine Funktion $\vartheta : \mathbb{N} \longrightarrow \mathbb{N}$, die injektiv und monoton wachsend ist), so daß die Koeffizientenfolgen $\left(\bar{a}_i^{(\vartheta(n))}\right)_{n \in \mathbb{N}}$ jeweils konvergieren. Wenn wir jeweils den Grenzwert $a_i := \lim_{n \to \infty} \bar{a}_i^{(\vartheta(n))} \in [-1,1]$ einsetzen, erhalten wir nun eine Darstellung der 0. Dabei können aber nicht alle Koeffizienten 0 sein, da in jedem Koeffiziententupel $(\bar{a}_1^{(n)}, \ldots, \bar{a}_k^{(n)})$ mindestens ein Koeffizient vom Betrag 1 enthalten ist. Dies ist ein Widerspruch zur Definition des Gitters.

Sei nun umgekehrt eine additive Untergruppe L in $\mathbb{R}^n$ mit (3.1) gegeben. Diese additive Untergruppe hat ein Erzeugendensystem $\underline{b}_1, \ldots, \underline{b}_k$, wobei diese Vektoren linear unabhängig über $\mathbb{Z}$ gewählt werden können. Es genügt zu zeigen, daß diese Vektoren damit bereits über $\mathbb{R}$ linear unabhängig sein müssen.

Angenommen, diese Vektoren seien über $\mathbb{R}$ linear abhängig, d.h. es existiert eine Linearkombination

$$\sum_{i=1}^{k} a_i \underline{b}_i = 0 \text{ mit } a_1, \ldots, a_k \in \mathbb{R}, \tag{3.2}$$

wobei mindestens ein a_i verschieden von Null ist. Dann können wir durch Multiplikation mit einer geeigneten Konstanten die Koeffizienten so normieren, daß gilt:

$$\max\{|a_i| : 1 \leq i \leq k\} = 1. \tag{3.3}$$

Dann betrachten wir für alle Zahlen $n \in \mathbb{N}$ die Linearkombinationen $\sum_{i=1}^{k} \lceil n a_i \rfloor \underline{b}_i$. Aus Gleichung (3.2) folgt dann

$$\begin{aligned} \left\| \sum_{i=1}^{k} \lceil n a_i \rfloor \underline{b}_i \right\| &= \left\| \sum_{i=1}^{k} (\lceil n a_i \rfloor - n a_i) \underline{b}_i \right\| && \text{wegen (3.2)} \\ &\leq \sum_{i=1}^{k} \|\underline{b}_i\| && \text{gemäß der Dreiecksungleichung} \\ &\leq k \max_{i=1,\ldots,k} \|\underline{b}_i\|. \end{aligned}$$

Dabei sind alle diese Linearkombinationen verschieden. Falls nämlich

$$\sum_{i=1}^{k} \lceil n_1 a_i \rceil \underline{b}_i = \sum_{i=1}^{k} \lceil n_2 a_i \rceil \underline{b}_i$$

gilt, so folgt $\lceil n_1 a_i \rceil = \lceil n_2 a_i \rceil$ für $1 \leq i \leq k$, da $\underline{b}_1, \ldots, \underline{b}_k$ über $\mathbb{Z}$ linear unabhängig sind. Mit (3.3) folgt hieraus $n_1 = n_2$.

Damit gibt es also im Kreis mit Radius $R := k \max_{i=1,\ldots,k} \|\underline{b}_i\|$ um den Nullpunkt unendlich viele verschiedene Vektoren aus L. Daher kann man für jedes $\epsilon > 0$ Vektoren $\underline{c}_1, \underline{c}_2 \in L$ finden mit $0 \neq \|\underline{c}_1 - \underline{c}_2\| < \epsilon$. Gäbe es nämlich eine positive untere Schranke $\delta > 0$ für diesen Abstand, so dürfte in jedem Kreis mit Durchmesser δ um einen beliebigen Punkt maximal ein Vektor aus L liegen. Da man den Kreis um den Nullpunkt mit Radius R mit endlich vielen Kreisen mit Radius $\frac{\delta}{2}$ überdecken kann (der abgeschlossene Kreis um den Nullpunkt mit Radius R ist nämlich eine kompakte Menge!), ergäbe sich dann ein Widerspruch dazu, daß es in diesem Kreis unendlich viele verschiedene Vektoren aus L geben soll.

Da L nach Voraussetzung eine additive Untergruppe des $\mathbb{R}^n$ ist, liegt mit zwei Vektoren auch ihre Differenz in L. Also erhalten wir für jedes $\epsilon > 0$ einen Vektor $\underline{c} \in L$ mit $0 \neq \|\underline{c}\| < \epsilon$. Dies stellt aber einen Widerspruch zu (3.1) dar. □

3.3. Beispiel

1. Die Spaltenvektoren von $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ erzeugen das Gitter $L = \mathbb{Z}^2$.

2. Die Zahlen 1 und $\sqrt{2}$ erzeugen *kein* Gitter, da sie nur über $\mathbb{Q}$, nicht aber über $\mathbb{R}$ linear unabhängig sind. Insbesondere kann man wie im vorangegangenen Beweis zeigen, daß im Intervall $[-1, 1]$ unendlich viele verschiedene Zahlen der Form $a + b\sqrt{2}$ mit $a, b \in \mathbb{Z}$ liegen und also der Betrag der von 0 verschiedenen Zahlen dieser Form durch keine Zahl $\epsilon > 0$ nach unten beschränkt ist.

3. Unser zentrales Beispiel stellt nun den Bezug zu Ordnungen her: Sei $\mathcal{O}$ eine Ordnung mit Basis Ω. Dann bilden die Konjugiertenvektoren der Basiselemente, also in der Notation von Definition 1.41 $\underline{\Omega}$, die Basis eines Gitters. Um dies zu beweisen, zeigen wir zunächst eine Hilfsbehauptung, die auch später noch sehr nützlich sein wird.

3.4. Lemma

Sei $\mathcal{O}$ eine Ordnung, die als $\mathbb{Z}$–Modul Rang n hat, und sei $\alpha \in \mathcal{O}$, $\alpha \neq 0$. Dann gilt für die Darstellung $\underline{\alpha}$ von α mittels eines Konjugiertenvektors:

$$\|\underline{\alpha}\| \geq \sqrt{\frac{n}{2}}.$$

Beweis: (vgl. [BuKe90])

Wir bezeichnen mit $\sigma_1, \ldots, \sigma_{r_1}$ die reellwertigen Einbettungen des von $\mathcal{O}$ erzeugten Zahlkörpers und mit $\sigma_{r_1+1}, \ldots, \sigma_{r_1+2r_2}$ die komplexwertigen Einbettungen, wobei gelten soll: $r_1 + 2r_2 = n$ und $\sigma_{r_1+r_2+i} = \overline{\sigma_{r_1+i}}$ für $1 \leq i \leq r_2$. Da α in der Ordnung $\mathcal{O}$ liegt, ist α eine algebraisch ganze Zahl. Für die Norm $N(\alpha)$ einer algebraisch ganzen Zahl $\alpha \neq 0$ gilt aber nach Satz 1.40:

$$|N(\alpha)| = \prod_{i=1}^{n} |\sigma_i(\alpha)| \geq 1. \tag{3.4}$$

Aus der Analysis kennen wir die Ungleichung vom arithmetischen und geometrischen Mittel: $\sqrt[n]{\prod_{i=1}^{n} a_i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i$. Damit folgt nun:

$$|N(\alpha)|^{2/n} \leq \frac{1}{n} \sum_{i=1}^{n} |\sigma_i(\alpha)|^2 \leq \frac{2}{n} \sum_{i=1}^{r_1+r_2} |\sigma_i(\alpha)|^2 = \frac{2}{n} \|\underline{\alpha}\|^2$$

und somit unter Verwendung von (3.4) die Behauptung. □

Damit können wir nun sehr leicht beweisen, daß wir $\underline{\Omega}$ als Gitterbasis auffassen können. Genauer zeigen wir:

3.5. Lemma

Sei $\Omega = (\omega_1, \ldots, \omega_n)$ die Basis einer Ordnung $\mathcal{O}$. Dann ist $\underline{\Omega} = (\underline{\omega}_1, \ldots, \underline{\omega}_n)$ die Basis eines n–dimensionalen Gitters, d.h. es gilt: $\underline{\omega}_1, \ldots, \underline{\omega}_n$ sind über $\mathbb{R}$ linear unabhängig.

Beweis:

Eine übliche Methode, diese Aussage zu beweisen besteht darin, zu zeigen, daß die Einbettungen σ_i, $1 \leq i \leq n$ linear unabhängig über $\mathbb{R}$ sind. Daraus ergibt sich dann die Behauptung. Wir wissen allerdings dank Satz 3.2 bereits einiges über die Struktur von Gittern, was wir nun ausnutzen können. Aufgrund von Satz 3.2 folgt die Behauptung nämlich offensichtlich aus dem vorangegangenen Lemma 3.4. □

Da wir dabei reellwertige Gitter betrachten, können wir die Basis wiederum nur näherungsweise angeben, da wir die exakten Werte nicht darstellen können. Analog zu den Näherungen an Matrizen definieren wir daher:

3.6. Definition (Näherungsgitter)
Sei $L \subset \mathbb{R}^n$ ein Gitter mit Basis $\underline{b}_1, \ldots, \underline{b}_k$. Dann bezeichnen wir mit $L\{q\}$ das Näherungsgitter der Präzision *q, d.h. $L\{q\}$ ist das Gitter mit Basis $\underline{b}_1\{q\}, \ldots, \underline{b}_k\{q\}$.*

Im folgenden beschränken wir uns der Einfachheit halber auf den Fall $k = n$. Nur diesen Fall werden wir nämlich im weiteren benötigen. Die Definitionen lassen sich jedoch mit einigem technischen Aufwand auch auf den Fall $k < n$ übertragen.

3.7. Definition
Sei $L \subset \mathbb{R}^n$ ein Gitter mit Basis $\underline{b}_1, \ldots, \underline{b}_n$. Dann bezeichnen wir mit $\lambda_i(L)$ das i-te sukzessive Minimum von L, d.h.

$$\begin{aligned}\lambda_i(L) \quad = \quad & \min\{r \in \mathbb{R}_{>0} : \textit{es gibt } i \textit{ linear unabhängige Gittervektoren} \\ & \qquad \textit{der (euklidischen) Länge} \leq r\}.\end{aligned}$$

Die Determinante

$$\det L := |\det(\underline{b}_1, \ldots, \underline{b}_n)|$$

heißt Determinante des Gitters *L bzw.* Gitterdeterminante *von L.*

3.8. Satz
Die Determinante eines Gitters L ist unabhängig von der Basiswahl.

Beweis:
Da die Basistransformationen, die ein Gitter unverändert lassen, genau die unimodularen Transformationen sind, bleibt der Betrag der Determinanten der Basis bei einem Basiswechsel unverändert. $\square$

Wichtig ist auch, daß wir die Ergebnisse, die wir im vorangehenden Kapitel für Determinanten von Näherungsmatrizen erhalten haben, einfach übertragen können. So erhalten wir z.B. die folgenden Aussagen:

3.9. Bemerkung
Sei $B = (\underline{b}_1, \ldots, \underline{b}_n)$ eine Basis des Gitters L. Wegen $\det L := |\det(\underline{b}_1, \ldots, \underline{b}_n)|$ gelten die Abschätzungen aus Korollar 2.7 und Bemerkung 2.8 ebenso für die Determinante des Approximationsgitters.

Aus Lemma 2.9 folgt insbesondere für alle $q \in \mathbb{N}$ mit $q \geq 2\log n + \log \mathrm{dft}(B) - \log \lambda_1(B)$: $L\{q\}$ ist auch ein Gitter der Dimension n und $(\underline{b}_1\{q\}, \ldots, \underline{b}_n\{q\})$ ist eine Basis des Gitters $L\{q\}$.

3.2 Der LLL–Algorithmus

Bei Berechnungen für Gitter tritt häufig folgendes Problem auf: Gegeben sei ein Gitter $L \subset \mathbb{R}^n$ mit Basis $\underline{b}_1, \ldots, \underline{b}_n$. Bestimme eine Basis aus möglichst kurzen Gittervektoren!

Für Gitter, deren Basisvektoren alle in $\mathbb{Z}^n$ liegen und für die also $L \subset \mathbb{Z}^n$ gilt, sind Lösungen für dieses Problem bekannt: Es gibt Algorithmen, die wirklich eine Basis aus den kürzestmöglichen Vektoren berechnen, diese benötigen jedoch exponentielle Zeit. Daher wollen wir hier nur den **LLL**-Algorithmus von Lenstra, Lenstra und Lovász vorstellen, der in [LLL82] veröffentlicht wurde. Dieser berechnet in polynomieller Zeit zwar nicht die kürzeste Basis, aber doch eine Basis aus relativ kurzen Vektoren.

Die Idee des Algorithmus ist, eine Basis zu berechnen, die möglichst nahe an der Gram-Schmidt-orthogonalisierten Basis liegt. Dabei ist die Gram-Schmidt-orthogonalisierte Basis $\underline{b}_1^*, \ldots, \underline{b}_n^*$ induktiv wie folgt definiert:

$$\begin{aligned}
\underline{b}_1^* &= \underline{b}_1 \\
\underline{b}_i^* &= \underline{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \underline{b}_j^* \\
\mu_{ij} &= \frac{\langle \underline{b}_i, \underline{b}_j^* \rangle}{\langle \underline{b}_j^*, \underline{b}_j^* \rangle}
\end{aligned}$$

Der Algorithmus sieht nun folgendermaßen aus:

3.10. Algorithmus (**LLL**-Algorithmus)
Eingabe: Basis $\underline{b}_1, \ldots, \underline{b}_n$ von $L \subset \mathbb{Z}^n$.
Ausgabe: **LLL**-reduzierte Basis $\tilde{\underline{b}}_1, \ldots, \tilde{\underline{b}}_n$.

1. Initialisierung: Berechne die Gram-Schmidt-Basis $\underline{b}_1^*, \ldots, \underline{b}_n^*$ und bestimme für $1 \leq j < i \leq n$ die μ_{ij}. Berechne $B_i := \langle \underline{b}_i^*, \underline{b}_i \rangle$. $k := 2$. $\tilde{\underline{b}}_i := \underline{b}_i$ für $1 \leq i \leq n$.

2. Transformiere die Basis, so daß $|\mu_{k,k-1}| \leq \frac{1}{2}$, d.h. $\tilde{\underline{b}}_k := \tilde{\underline{b}}_k - \lfloor \mu_{k,k-1} \rceil \tilde{\underline{b}}_{k-1}$ und aktualisiere μ_{kj} $(1 \leq j \leq k-1)$.

3. IF $B_k \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) B_{k-1}$ THEN

   Transformiere die Basis, so daß $|\mu_{kl}| \leq \frac{1}{2}$, d.h. $\tilde{\underline{b}}_k := \tilde{\underline{b}}_k - \lfloor \mu_{k,l} \rceil \tilde{\underline{b}}_l$ und aktualisiere μ_{kj} $(1 \leq j \leq l)$ für $l = k-2, k-3, \ldots, 1$. $k++$.

   ELSE

   Vertausche $\tilde{\underline{b}}_k$ und $\tilde{\underline{b}}_{k-1}$ und aktualisiere μ_{ij} $(1 \leq j < i \leq n)$ und B_k, B_{k-1}. If $k > 2$, then $k--$.

   FI;

4. IF $k \leq n$ then goto (2) FI.

3.11. Bemerkung
Bei der Aktualisierung der μ_{ij} bzw. von B_k und B_{k-1} können die neuen Werte rekursiv aus den alten Werten berechnet werden. Insbesondere benötigt man zur Berechnung der neuen Werte der μ_{ij} und von B_k und B_{k-1} *nicht* die Werte der $\underline{b}_i$. Die Rekursionsformeln zur Aktualisierung von μ_{ij} bzw. von B_k und B_{k-1} finden sich ebenfalls in [LLL82].

Die Autoren des Artikels [LLL82] (auf den wir für den Beweis verweisen) haben in ihrer Arbeit den folgenden Satz bewiesen:

3.12. Satz (LLL-Reduktion)
Sei $L \subset \mathbb{Z}^n$ ein Gitter mit Basis $\underline{b}_1, \ldots, \underline{b}_n$ und sei $B \in \mathbb{Z}^{n \times n}$ mit $B := (\underline{b}_1, \ldots, \underline{b}_n)$. Dann berechnet Algorithmus 3.10 eine reduzierte Gitterbasis $\tilde{B} = (\tilde{\underline{b}}_1, \ldots, \tilde{\underline{b}}_n)$ mit folgenden Eigenschaften:

- $\|\tilde{\underline{b}}_1\| \leq 2^{\frac{n-1}{4}} (\det L)^{\frac{1}{n}}$
- $\|\tilde{\underline{b}}_i\| \leq 2^{\frac{n-1}{2}} \lambda_i(L)$ *für alle* $1 \leq i \leq n$
- $\prod_{i=1}^{n} \|\tilde{\underline{b}}_i\| \leq 2^{\frac{n(n-1)}{4}} (\det L)$

Dazu benötigt Algorithmus 3.10 $O(n^4 \max(1, \log \lambda_n(B)))$ arithmetische Operationen, wobei die Operationen mit ganzen Zahlen der binären Länge $O(n \max(1, \log \lambda_n(B)))$ ausgeführt werden.

3.3 Eine Modifikation des LLL-Algorithmus

Wir wollen nun den **LLL**-Algorithmus auf Gitter $L \subset \mathbb{R}^n$ anwenden. Dazu müssen wir aber ein Näherungsgitter $L\{q\}$ berechnen und dieses reduzieren. Wenn man den **LLL**-Algorithmus aber auf Approximationen an reellwertige Gitterbasen anwendet, erhält man natürlich nur eine Reduktion der Approximation. An diesem Ergebnis sind wir aber nicht interessiert. Statt dessen wollen wir die Transformationsmatrix T erhalten, mit der das Gitter reduziert wird. Mit Hilfe dieser Transformationsmatrix kann man dann die Darstellung der exakten Basis transformieren. Damit erhält man – wie wir zeigen werden – ein Ergebnis, das fast genauso gut ist, als hätte man die exakte Basis mit exakter Rechnung reduziert.

Wir modifizieren den **LLL**-Algorithmus nun so, daß statt der reduzierten Basis die Matrix T berechnet wird:

3.13. Algorithmus (Modifizierter **LLL**-Algorithmus)

Eingabe: Basis $\underline{b}_1\{q\},\ldots,\underline{b}_n\{q\}$ von $L\{q\}$.

Ausgabe: Transformationsmatrix $T = (\underline{t}_1,\ldots,\underline{t}_n)$, mit der man $(\underline{b}_1,\ldots,\underline{b}_n)$ multiplizieren muß, um eine nahezu **LLL**-reduzierte Basis zu erhalten.

1. Initialisierung: $T := I_n$. Berechne die Gram-Schmidt-Basis $\underline{b}_1^*,\ldots,\underline{b}_n^*$ und bestimme für $1 \leq j < i \leq n$ die μ_{ij} mit $\underline{b}_i^* = \underline{b}_i\{q\} - \sum_{j=1}^{i-1} \mu_{ij}\underline{b}_j^*$. Berechne $B_i := \langle \underline{b}_i^*, \underline{b}_i\{q\}\rangle$. $k := 2$.

2. Transformiere die Basis, so daß $|\mu_{k,k-1}| \leq \frac{1}{2}$, d.h. $\underline{t}_k := \underline{t}_k - \lfloor \mu_{k,k-1} \rceil \underline{t}_{k-1}$ und aktualisiere μ_{kj} $(1 \leq j \leq k-1)$ mit Hilfe der Rekursionsformeln.

3. IF $B_k \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) B_{k-1}$ THEN

   Transformiere die Basis, so daß $|\mu_{kl}| \leq \frac{1}{2}$, d.h. $\underline{t}_k := \underline{t}_k - \lfloor \mu_{k,l} \rceil \underline{t}_l$ und aktualisiere mit Rekursion μ_{kj} $(1 \leq j \leq l)$ für $l = k-2, k-3, \ldots, 1$. $k++$.

   ELSE

   Vertausche $\underline{t}_k$ und $\underline{t}_{k-1}$ und aktualisiere μ_{ij} $(1 \leq j < i \leq n)$ und B_k, B_{k-1} mittels der Rekursionsformeln. If $k > 2$, then $k--$.

   FI;

4. IF $k \leq n$ then goto (2) FI.

3.14. Bemerkung

Der einzige Unterschied zu Algorithmus 3.10 besteht also darin, daß in Algorithmus 3.13 die Transformationen mit einer Einheitsmatrix anstelle von $B = (\underline{b}_1,\ldots,\underline{b}_n)$ als Startwert ausgeführt werden, wenn man davon absieht, daß Algorithmus 3.13 Zahlen aus $2^{-q}\mathbb{Z}$ statt Zahlen aus $\mathbb{Z}$ als Eingabe erhält. Insbesondere müssen wir wegen Bemerkung 3.11 die jeweils aktuellen Werte, die man für die Basis erhalten würde, nicht berechnen.

3.4 Analyse des modifizierten Algorithmus

Nun müssen wir noch zeigen, daß dieser Algorithmus auch eine Transformationsmatrix liefert, die das Gewünschte leistet. Genauer beweisen wir folgenden Satz:

3.15. Satz (Korrektheit)
Nach jedem Teilschritt von Algorithmus 3.13 gilt:

$$B\{q\}T = \tilde{B},$$

wobei $\tilde{B} = (\tilde{\underline{b}}_1, \ldots, \tilde{\underline{b}}_n)$ *die Basis ist, die man aus* $B\{q\}$ *nach diesem Schritt durch Basistransformation gemäß Algorithmus 3.10 erhalten würde. Insbesondere gilt dies also am Ende des Algorithmus, wenn* $\tilde{B}$ *eine* **LLL***-reduzierte Basis von* $B\{q\}$ *wäre.*

Beweis:
Wir zeigen die Aussage induktiv.

Klar ist, daß die Behauptung nach Ausführung von Schritt (1) gilt. Wenn sie nun vor Schritt (2) gilt, gilt sie auch danach:

$$\begin{aligned}
\tilde{\underline{b}}_{k,\text{aktuell}} &= \tilde{\underline{b}}_{k,\text{old}} - \lfloor \mu_{k,k-1} \rceil \tilde{\underline{b}}_{k-1,\text{old}} \\
&= B\{q\}\underline{t}_{k,\text{old}} - \lfloor \mu_{k,k-1} \rceil B\{q\}\underline{t}_{k-1,\text{old}} \qquad \text{nach Induktionsannahme} \\
&= B\{q\}\left(\underline{t}_{k,\text{old}} - \lfloor \mu_{k,k-1} \rceil \underline{t}_{k-1,\text{old}}\right) \\
&= B\{q\}\underline{t}_{k,\text{aktuell}}
\end{aligned}$$

Ebenso zeigt man, daß die Behauptung nach Ausführung des „then"-Falles von Schritt (3) gilt. Nach Ausführung des „else"- Zweigs gilt die Behauptung ebenfalls, da das Vertauschen der Spalten von T genau der Spaltenvertauschung in $\tilde{B}$ entspricht. □

Um Aussagen über die Güte der Reduktion bzw. über die Laufzeit zu beweisen, benutzen wir folgendes Lemma, das eine enge Beziehung zwischen Algorithmus 3.10 und Algorithmus 3.13 herstellt:

3.16. Lemma
Sei $L\{q\} \subset (2^{-q}\mathbb{Z})^n$ *ein Approximationsgitter mit Basis* $\underline{b}_1\{q\}, \ldots, \underline{b}_n\{q\}$*. Diese erhalte Algorithmus 3.13 als Eingabe. Dann werden von Algorithmus 3.13 die gleichen Operationen ausgeführt, die Algorithmus 3.10 für das Gitter* $L' = 2^q L\{q\} \subset \mathbb{Z}^n$ *mit Basis* $2^q\underline{b}_1\{q\}, \ldots, 2^q\underline{b}_n\{q\}$ *ausführt. Insbesondere stimmen die Werte der* μ_{ij} *stets überein.*

Beweis:
Die Werte der μ_{ij} aus Algorithmus 3.10 bezeichnen wir in diesem Beweis zur Unterscheidung von den Werten aus Algorithmus 3.13 mit μ'_{ij}. Dann gilt zu Beginn:

$$\mu'_{ij} = \frac{\langle 2^q \underline{b}_i\{q\}, 2^q \underline{b}_j\{q\}^* \rangle}{\langle 2^q \underline{b}_j\{q\}^*, 2^q \underline{b}_j\{q\}^* \rangle},$$

denn mittels Induktion ist klar, daß sich die Werte für $\underline{b}_i\{q\}^*$ in den beiden Algorithmen gerade um den Faktor 2^q unterscheiden, ebenso wie die Werte für die Startbasis. Nun kann man die Faktoren 2^q aber aus den Skalarprodukten in Zähler und Nenner jeweils herausziehen und kürzen. Damit gilt nach Schritt (1):

$$\mu'_{ij} = \mu_{ij}. \tag{3.5}$$

Die Werte der $B_i = \langle \underline{b}_i\{q\}^*, \underline{b}_i\{q\} \rangle$ aus Algorithmus 3.13 unterscheiden sich zu Anfang von den Werten, die Algorithmus 3.10 erhält ($B_i = \langle 2^q \underline{b}_i\{q\}^*, 2^q \underline{b}_i\{q\} \rangle$), gerade um den Faktor 4^q. In der Entscheidung von Schritt (3) spielt aber nur das Verhältnis zweier solcher B_i eine Rolle, also wird hier jeweils der gleiche Fall ausgewählt.

Daher kann man nun durch Induktion leicht zeigen, daß Gleichung (3.5) stets erhalten bleibt, da in den Schritten (2) und (3) stets die gleichen Operationen ausgeführt werden.

$\square$

Nun wollen wir untersuchen, wie gut das Ergebnis ist, das man erreichen kann, wenn man mit einer Näherungsbasis startet.

3.17. Satz (Qualität des Ergebnisses)
Sei $L \subset \mathbb{R}^n$ ein Gitter im $\mathbb{R}^n$ mit Basis $B := (\underline{b}_1, \ldots, \underline{b}_n)$ und sei $L\{q\}$ mit $q \geq 2\log n + \log \operatorname{dft}(B) - \log \lambda_1(B)$ das Näherungsgitter, das Algorithmus 3.13 als Eingabe erhält. Für das Ergebnis $T = (\underline{t}_1, \ldots, \underline{t}_n)$ von Algorithmus 3.13 gilt:

$$\prod_{i=1}^{n} \|B\underline{t}_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L) \left[1 + \operatorname{dft}(B)\left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(B)}\right)^n - 1\right)\right] \left[1 + 2^{-q} n^2 \frac{\operatorname{dft}(B)}{\lambda_1(B)}\right]^n .$$

Beweis:
Nach Bemerkung 3.9 ist insbesondere $\{\underline{b}_1\{q\}, \ldots, \underline{b}_n\{q\}\}$ eine Menge linear unabhängiger Vektoren. Dann erhalten wir aber wegen Lemma 3.16 mit Satz 3.12 das folgende Ergebnis: Es ist $\prod_{i=1}^n \|\tilde{\underline{b}}_i\| \leq 2^{\frac{n(n-1)}{4}} (\det L\{q\})$, wobei $(\tilde{\underline{b}}_1, \ldots, \tilde{\underline{b}}_n)$ die **LLL**-reduzierte Gitterbasis zu

$L\{q\}$ ist. (Mit Algorithmus 3.10 erhielten wir nämlich $\prod_{i=1}^{n}\|2^q\tilde{\underline{b}}_i\| \le 2^{\frac{n(n-1)}{4}} \det(2^q L\{q\})$. Mit $\det(2^q L\{q\}) = 2^{nq}(\det L\{q\})$ folgt also diese Behauptung). Also gilt mit Satz 2.10

$$\prod_{i=1}^{n} \|B\underline{t}_i\| \le 2^{\frac{n(n-1)}{4}} \det(L\{q\}) \left(1 + 2^{-q} n^2 \frac{\mathrm{dft}(B)}{\lambda_1(B)}\right)^n$$

Mit Korollar 2.7 folgt dann die Behauptung. □

3.18. Bemerkung

Der vorangehende Satz besagt, daß man durch Wahl eines hinreichend großen Wertes für q beliebig nahe an das „exakte" Ergebnis herankommen kann, daß man erhalten würde, wenn man den **LLL**-Algorithmus mit exakter Eingabe exakt rechnen lassen würde.

3.19. Bemerkung

Bei normaler **LLL**-Reduktion kann man zeigen, daß die Basis, die der Algorithmus als Eingabe erhält, garantiert nicht verlängert wird, da in jedem Einzelschritt die Vektoren allenfalls verkürzt werden. Dieses Resultat ist hier jedoch nur für die Näherungsbasis, auf die die **LLL**-Reduktion angewandt wird, gesichert. Die eigentlich zu reduzierende Basis kann durch Anwendung der berechneten Transformation innerhalb der zulässigen Grenzen tatsächlich länger werden, etwa wenn die Eingabe schon reduziert ist, die Näherungsbasis jedoch nicht reduziert ist.

Schließlich zeigen wir noch, daß der modifizierte Algorithmus im wesentlichen die gleiche Laufzeit wie der Originalalgorithmus hat:

3.20. Satz (Laufzeit)

Sei $L \subset \mathbb{R}^n$ ein Gitter mit Basis $(\underline{b}_1, \ldots, \underline{b}_n) =: B$ und sei $L\{q\}$ mit $q \ge 2\log n + \log \mathrm{dft}(B) - \log \lambda_1(B)$ das Näherungsgitter, das Algorithmus 3.13 als Eingabe erhält. Dann führt Algorithmus 3.13 $O(n^4 \max(1, q + \log \lambda_n(B)))$ arithmetische Operationen aus, wobei die Zahlen, mit denen die Operationen ausgeführt werden, binäre Länge $O(n \max(1, \log \lambda_n(B) + q))$ haben. Also ergibt sich eine Laufzeit von

$$O(n^6 \max(1, q + \log \lambda_n(B))^3).$$

Beweis:

Nach Lemma 3.16 benötigt Algorithmus 3.13 dieselbe Zahl arithmetischer Operationen, die Algorithmus 3.10 mit Basis $2^q\underline{b}_1\{q\}, \ldots, 2^q\underline{b}_n\{q\}$ benötigen würde, also nach Satz 3.12

$$O(n^4 \max(1, \log(2^q \lambda_n(B\{q\})))) = O(n^4 \max(1, q + \log \lambda_n(B)))$$

Die Zahlen zur Darstellung der μ_{ij} und der B_i haben die gleiche Länge wie die Zahlen, die in Algorithmus 3.10 mit der entsprechend modifizierten Basis auftreten würden. In der Tat unterscheiden sich die auftretenden Zahlen höchstens durch die Interpretation der jeweils letzten Bits als Nachkommastellen in Algorithmus 3.13. Also haben nach Satz 3.12 die auftretenden Zahlen die Länge $O(n \max(1, q + \log \lambda_n(B)))$.

Es bleibt zu zeigen, daß diese Schranke auch für alle Einträge in T gilt. Dazu beachte, daß nach jedem Schritt (wegen Satz 3.15) gilt:

$$2^q B\{q\} \underline{t}_i = 2^q \underline{\tilde{b}}_i,$$

wobei $\underline{\tilde{b}}_i$ wie in Satz 3.15 definiert ist. Dann gilt nach der Kramerschen Regel (vgl. (1.2)):

$$|t_{ij}| \leq \frac{\prod\limits_{\substack{k=1 \\ k \neq j}}^{n} 2^q \|\underline{b}_k\{q\}\| 2^q \|\underline{\tilde{b}}_i\|}{|\det(2^q B\{q\})|} \qquad \text{für } 1 \leq i, j \leq n.$$

Nun ist aber $2^q B\{q\}$ eine reguläre Matrix mit Einträgen aus $\mathbb{Z}$, also gilt $\det(2^q B\{q\}) \in \mathbb{Z}$ und $\det(2^q B\{q\}) \neq 0$, d.h. $|\det(2^q B\{q\})| \geq 1$, also:

$$\|\underline{t}_i\|_\infty \leq \max_{j=1,\ldots,n} \prod_{\substack{k=1 \\ k \neq j}}^{n} \underbrace{2^q \|\underline{b}_k\{q\}\|}_{\substack{\leq 2^q \lambda_n(B) + \frac{\sqrt{n}}{2} \leq 2 \cdot 2^q \lambda_n(B), \\ \text{da } q \geq 2 \log n + \log \operatorname{dft}(B) - \log \lambda_1(B)}} 2^q \|\underline{\tilde{b}}_i\| \quad \text{für } 1 \leq i \leq n.$$

Damit hat ein Eintrag in T also höchstens Länge

$$\begin{aligned} &(n-1)(q + 1 + \log \lambda_n(B)) + \underbrace{\text{Länge eines Eintrages in } 2^q \tilde{B}}_{\in O(n \max(1, q + \log \lambda_n(B))) \text{ nach Satz 3.12}} \\ &= O(n \max(1, q + \log \lambda_n(B))). \end{aligned}$$

Nach Satz 1.1, in dem wir den Zeitbedarf elementarer Operationen in $\mathbb{N}$ untersucht haben, ergibt sich nun die angegebene Laufzeit, da wir diese Operationen im wesentlichen ebenso in $2^{-q}\mathbb{Z}$ durchführen können. Über $2^{-q}\mathbb{Z}$ können wir nämlich ebenso rechnen, wie wir es nach Multiplikation mit 2^q über $\mathbb{Z}$ tun würden, lediglich müssen wir zusätzlich noch auf die Position des Kommas achten. □

3.21. Bemerkung

Wir können statt gewöhnlicher „schriftlicher" Multiplikation auch schnellere Techniken verwenden, wie in Bemerkung 1.2 angedeutet wurde. Damit können wir die Laufzeit von Algorithmus 3.13 noch auf

$$O(n^{5+\epsilon} \max(1, q + \log \lambda_n(B))^{2+\epsilon})$$

senken. Wenn wir mit sehr großer Präzision rechnen, können diese asymptotisch schnellen Multiplikationsalgorithmen eventuell sogar in der Praxis nutzbringend angewendet werden.

Kapitel 4

Algorithmen für Ordnungen

In den folgenden Abschnitten sei stets $\mathcal{O}$ eine Ordnung des algebraischen Zahlkörpers $\mathbb{K}$, die gegeben sei durch die Multiplikationstafel $\mathrm{MT}(\Omega) = (w_{ijk})_{1 \leq i,j,k \leq n}$ bezüglich einer $\mathbb{Z}$–Basis $\Omega = (\omega_1, \ldots, \omega_n)$. Dann gilt also für den Körpergrad von $\mathbb{K}$ über $\mathbb{Q}$: $[\mathbb{K} : \mathbb{Q}] = n$.

4.1 Berechnung einer Basis

Um den Schritt der **LLL**–Reduktion ausführen zu können, der uns eine kurze Basis liefern soll, benötigen wir eine Darstellung der Basis einer Ordnung mittels Konjugiertenvektoren, in der Terminologie von Definition 1.41 wollen wir also für eine gegebene Multiplikationstafel $\mathrm{MT}(\Omega)$ nun $\underline{\Omega}$ berechnen. Dies geschieht, indem wir zunächst die Darstellung eines primitiven Elements als Linearkombination der Basiselemente und dessen Minimalpolynom bestimmen und dann die Nullstellen dieses Polynoms mit einer vorgegebenen Genauigkeit annähern. Damit erhalten wir den Konjugiertenvektor des primitiven Elements. Zum Schluß konstruieren wir aus der Darstellung des primitiven Elements als Koeffizientenvektor eine Darstellung zder Basiselemente als Linearkombination der Potenzen des primitiven Elements und erhalten damit die Konjugiertenvektoren der Basiselemente. Wenn wir dabei die Anforderungen an die Genauigkeit aus Satz 3.17 berücksichtigen, ist nach Lemma 2.9 garantiert, daß wir linear unabhängige Konjugiertenvektoren erhalten, und somit ist sogar gesichert, daß wir die Nullstellen des Minimalpolynoms voneinander trennen können.

4.1.1 Konstruktion von Minimalpolynomen

Zunächst wollen wir untersuchen, wie man zu einer durch ihren Koeffizientenvektor gegebenen algebraischen Zahl das Minimalpolynom berechnen kann. Unsere Methode beruht auf dem folgenden Satz:

4.1. Satz

Sei α eine durch ihren Koeffizientenvektor $\underline{a} = (a_1, \ldots, a_n)^T$ gegebene algebraische Zahl. Dann sind die Eigenwerte der Matrix A_α, die bezüglich der Basis $\omega_1, \ldots, \omega_n$ zu der Abbildung $\mathcal{O} \longrightarrow \mathcal{O}, \beta \longmapsto \alpha\beta$ gehört, gerade die Zahlen $\sigma_i(\alpha)$, wobei σ_i die n verschiedenen Einbettungen von $\mathbb{K}$ in $\mathbb{C}$ durchläuft. Dabei können wir A_α aus $\underline{a}$ und $\mathrm{MT}(\Omega)$ in Zeit $O(n^3 \log^2(n|\mathrm{MT}(\Omega)||\underline{a}|))$ berechnen und für die Einträge gilt $|A_\alpha| \leq n|\underline{a}||\mathrm{MT}(\Omega)|$.

Beweis:

Die Matrix A_α können wir wie folgt berechnen: Es ist

$$\begin{aligned} \alpha\omega_i &= (\sum_{j=1}^{n} a_j\omega_j)\omega_i \\ &= (\sum_{j=1}^{n} a_j\omega_j\omega_i) \\ &= \sum_{j=1}^{n}\sum_{k=1}^{n} a_j w_{ijk}\omega_k. \end{aligned}$$

Also ergibt sich die i-te Spalte der Matrix als

$$(\sum_{j=1}^{n} a_j w_{ij1}, \ldots, \sum_{j=1}^{n} a_j w_{ijn})^T.$$

Damit erhalten wir $|A_\alpha| \leq n|\underline{a}||\mathrm{MT}(\Omega)|$.

Zur Berechnung einer Spalte der Matrix führen wir n^2 Multiplikationen von Zahlen der Größe $|\mathrm{MT}(\Omega)|$ mit Zahlen der Größe $|\underline{a}|$ aus und addieren jeweils n davon auf. Also ergibt sich ein Zeitaufwand von $O(n^2 \log|\mathrm{MT}(\Omega)| \log|\underline{a}|)$ für die Multiplikationen und ein Aufwand von $O(n^2 \log(n|\underline{a}||\mathrm{MT}(\Omega)|))$ für die Additionen. Die Berechnung aller Einträge der Matrix ist also in Zeit $O(n^3 \log^2(n|\mathrm{MT}(\Omega)||\underline{a}|))$ möglich.

Jetzt wollen wir noch die Aussage über die Eigenwerte von A_α beweisen. Dabei bezeichnen wir hier ausnahmsweise mit $\underline{\alpha}$ den vollständigen Konjugiertenvektor einer algebraischen Zahl α, d.h. $\underline{\alpha} = (\sigma_1(\alpha), \ldots, \sigma_n(\alpha))^T \in \mathbb{C}^n$. Dann ist zunächst klar, daß ein

Vektor $\underline{b} = (b_1, \ldots, b_n)^T$ von A_α auf $\underline{c} = (c_1, \ldots, c_n)^T$ abgebildet wird, wobei $\sum_{i=1}^n c_i\omega_i = \sum_{i=1}^n a_i\omega_i \sum_{i=1}^n b_i\omega_i$. Daraus ergibt sich das ein Konjugiertenvektor $\underline{\beta} = \sum_{i=1}^n b_i\underline{\omega_i}$ auf den entsprechenden Konjugiertenvektor $\sum_{i=1}^n c_i\underline{\omega_i} = \sum_{i=1}^n a_i\underline{\omega_i} \sum_{i=1}^n b_i\underline{\omega_i}$ abgebildet wird. Nun muß aber $A_\alpha\underline{\beta}$ dasselbe Ergebnis liefern wie $\underline{\alpha}\underline{\beta}$. Unter Berücksichtigung der Regeln zur Multiplikation von Konjugiertenvektoren erkennt man also: Die Vektoren mit $\sum_{i=1}^n b_i\underline{\omega_i} = \underline{e}_j$ sind genau die Eigenvektoren der Matrix A_α und werden gerade auf $\sigma_j(\alpha)\underline{e}_j$ abgebildet ($1 \leq j \leq n$). Also sind die Eigenwerte gerade die Zahlen $\sigma_i(\alpha)$, $1 \leq i \leq n$. □

Satz 4.1 besagt also, das das charakteristische Polynom von A_α mit dem charakteristischen Polynom von α im Sinne von Definition 1.39 übereinstimmt. Daher können wir jetzt zeigen:

4.2. Satz
Das Minimalpolynom $m_\alpha(X) = \sum_{i=0}^{k} m_{\alpha,i}X^i$ *eines Elements* α*, dargestellt durch* $\underline{a} = (a_1, \ldots, a_n)^T$ *mit* $\alpha = \sum_{i=1}^n a_i\omega_i \in \mathcal{O}$ *kann in Zeit*

$$O(n^4 \log^2(n|\operatorname{MT}(\Omega)||\underline{a}|)) \tag{4.1}$$

berechnet werden. Dabei sind die Koeffizienten von m_α *durch*

$$|m_{\alpha,i}| \leq (n^2|\underline{a}||\operatorname{MT}(\Omega)|)^n \textit{ für alle } 1 \leq i \leq k \tag{4.2}$$

und die Nullstellen $z_1, \ldots, z_k$ *von* m_α *durch*

$$|z_i| \leq n^2|\underline{a}||\operatorname{MT}(\Omega)| \textit{ für alle } 1 \leq i \leq k \tag{4.3}$$

beschränkt.

Beweis:
Wir berechnen zunächst die Matrix A_α. Nach Satz 4.1 gilt nun

$$|A_\alpha| \leq n|\underline{a}||\operatorname{MT}(\Omega)|.$$

Zunächst gilt die Koeffizientenabschätzung (4.2) nach Proposition 1.19 nur für das charakteristische Polynom der Matrix A_α. Ebenso wie im Beweis von Proposition 1.19 folgt aber aus dem Satz von Gershgorin auch die Ungleichung (4.3) und daraus die Abschätzung (4.2), da das Minimalpolynom einen Grad $\leq n$ hat.

Wiederum laut Satz 4.1 benötigen wir Zeit $O(n^3 \log^2(n|\operatorname{MT}(\Omega)||\underline{a}|))$ um A_α zu berechnen. Nach Satz 1.21 benötigen wir weiterhin Zeit

$$O(n^4 \log^2(n^2|\operatorname{MT}(\Omega)||\underline{a}|)) = O(n^4 \log^2(n|\operatorname{MT}(\Omega)||\underline{a}|)),$$

um das charakteristische Polynom c_α der Matrix A_α zu berechnen.

Da nach Satz 1.40 das charakteristische Polynom von A_α eine Potenz des Minimalpolynoms von α ist (genauer: die $\frac{n}{k}$–Potenz), erhalten wir das Minimalpolynom nun gerade durch folgende Gleichung:

$$m_\alpha = \frac{c_\alpha}{\operatorname{ggT}(c_\alpha, c'_\alpha)} \ . \tag{4.4}$$

(4.4) gilt aufgrund folgender Gleichungen:

$$c_\alpha = m_\alpha^f = \prod_{i=1}^{k} (X - \alpha_i)^f \qquad \text{mit } f = \frac{n}{k} \,,$$

wobei α_i die Nullstellen des Minimalpolynoms durchläuft und also

$$c'_\alpha = f \sum_{i=1}^{k} (X - \alpha_i)^{f-1} \prod_{j \neq i} (X - \alpha_j)^f.$$

Somit ergibt sich über $\mathbb{Z}$ als ggT gerade m_α^{f-1} und somit die Gleichung (4.4).

Nach Satz 1.30 kann nun m_α als Kofaktor des ggT in Zeit

$$O(n^3 \log s \log(n \log s) + n^2 \log^2 s)$$

berechnet werden, wenn s eine obere Schranke für die Beträge der Koeffizienten aller beteiligten Polynome ist, die außerdem $s > n$ erfüllt. Für m_α und c_α haben wir bereits die Schranke aus (4.2) gezeigt. Diese gilt mit dem Satz von Gershgorin ebenso für den zu berechnenden ggT. Für c'_α ergibt sich dann, da durch Ableiten eines Polynoms vom Grad $\leq n$ jeder Koeffizient höchstens mit n multipliziert wird, $n(n^2|\underline{a}||\operatorname{MT}(\Omega)|)^n$ als Schranke. Für den zugehörigen Kofaktor $f \sum_{i=1}^{k} \prod_{j \neq i} (X - \alpha_j)$ ergibt sich mit dem Satz von Gershgorin und wegen $fk = n$ dieselbe Schranke. Insgesamt können wir also

$$s := n(n^2|\underline{a}||\operatorname{MT}(\Omega)|)^n$$

wählen. Somit erhalten wir wegen $O(\log s) = O(n \log(n|\underline{a}||\operatorname{MT}(\Omega)|))$ als Laufzeit für die Berechnung des Kofaktors:

$$\begin{aligned} O(n^4 \log(n|\underline{a}||\operatorname{MT}(\Omega)|) \log(n \log(n|\underline{a}||\operatorname{MT}(\Omega)|)) + n^4 \log^2(n|\underline{a}||\operatorname{MT}(\Omega)|)) \\ = O(n^4 \log^2(n|\underline{a}||\operatorname{MT}(\Omega)|)). \end{aligned}$$

Also ergibt sich die angegebene Gesamtlaufzeit. $\square$

4.3. Bemerkung
Dieses Verfahren funktioniert nur für Zahlen aus der gegebenen Ordnung. Insbesondere erhalten wir stets ein normiertes Polynom mit Koeffizienten in $\mathbb{Z}$. Das Minimalpolynom eines beliebigen Körperelements wird allerdings kein normiertes Polynom aus $\mathbb{Z}[x]$ sein. Daher muß man einige Modifikationen an diesem Verfahren vornehmen, wenn man die Minimalpolynome beliebiger Körperelemente berechnen will! Da diese Modifikationen jedoch nicht schwierig sind und im weiteren auch nicht benötigt werden, wollen wir sie hier nicht weiter diskutieren.

4.1.2 Konstruktion eines primitiven Elements

Da $\omega_1, \ldots, \omega_n$ über $\mathbb{Q}$ den gesamten algebraischen Zahlkörper $\mathbb{K}$ erzeugen, können wir ein primitives Element, d.h. ein Element, dessen Potenzen $\mathbb{K}$ über $\mathbb{Q}$ erzeugen, und sein Minimalpolynom berechnen, wobei dieses primitive Element in $\mathcal{O}$ liegt: Zunächst wissen wir, daß es in einem algebraischen Zahlkörper stets ein primitives Element gibt. Genauer ergibt sich aus dem Beweis des aus der Algebra bekannten Satzes vom primitiven Element, wie er etwa in [Go73] (S. 280f) nachzulesen ist, der folgende Satz:

4.4. Satz
Sei $\mathbb{K}$ eine Körpererweiterung von $\mathbb{Q}$ vom Grad n und seien $\alpha, \beta \in \mathbb{K}$. Dann gibt es ein $\lambda \in \{0, 1, \ldots, (n-1)^2 + 1\}$ mit $\mathbb{Q}(\alpha, \beta) = \mathbb{Q}(\alpha + \lambda\beta)$.

Beweis:
Der in [Go73] angegebene Beweis zeigt, daß man als primitives Element eine Linearkombination der Form $\alpha + \lambda\beta$ wählen kann, wenn man λ geeignet wählt. Dabei ergibt sich aus dem Beweis, daß maximal $(n-1)^2 + 1$ Werte für λ existieren, so daß $\mathbb{Q}(\alpha + \lambda\beta) \neq \mathbb{Q}(\alpha, \beta)$. Wenn man also $(n-1)^2 + 2$ verschiedene Werte für λ einsetzt, so muß wenigstens einer davon ein primitives Element liefern. $\square$

Um zu überprüfen, ob ein Element $\alpha + \lambda\beta$ einen Körper größeren Grades erzeugt als α, genügt es, die Grade der beiden Minimalpolynome zu vergleichen. Wenn wir den Grad des Minimalpolynoms von ϱ vereinfachend mit $\deg(\varrho)$ bezeichnen, können wir aufgrund von Satz 4.4 das Minimalpolynom eines primitiven Elements wie folgt berechnen (vgl. auch [BuSp92]):

4.5. Algorithmus (primitives Element)
Eingabe: Multiplikationstafel $\mathrm{MT}(\Omega)$ einer Ordnung $\mathcal{O}$ mit $\Omega = (\omega_1, \ldots, \omega_n)$.
Ausgabe: Koeffizientenvektor $\underline{r}_1 = (r_{11}, \ldots, r_{n1})^T$ und normiertes Minimalpolynom aus $\mathbb{Z}[X]$ einer algebraischen Zahl ϱ mit $\mathbb{Q}(\varrho) = \mathbb{Q}(\omega_1, \ldots, \omega_n)$.

1. $\varrho := \omega_1$, d.h. $\underline{r}_1 := \underline{e}_1$; $i := 1$;
2. WHILE $\deg(\varrho) < n$ DO /* $\deg(\varrho) = \deg(\text{Minimalpolynom von } \varrho)$ */
   - $i++$; $\lambda := 0$;
   - FOR $\mu := 1$ TO $(n-1)^2 + 1$ DO
     /* Bestimme durch Ausprobieren hinreichend vieler Kandidaten */
     /* ein primitives Element von $\mathbb{Q}(\omega_1, \ldots, \omega_i)$ */

     IF $\deg(\varrho + \mu\omega_i) > \deg(\varrho + \lambda\omega_i)$ THEN $\lambda := \mu$ FI
     /* Neuer Kandidat für das primitive Element von $\mathbb{Q}(\varrho, \omega_i)$. */
     /* Erzeugt eventuell nur einen echten Zwischenkörper! */
   - OD;
   - $\varrho := \varrho + \lambda\omega_i$, d.h. $r_{i1} := \lambda$ /* Primitives Element von $\mathbb{Q}(\varrho, \omega_i)$ */
3. OD. /* Jetzt primitives Element von $\mathbb{K}$ */

4.6. Bemerkung
Für praktische Anwendungen beachte man, daß die Abschätzung für λ in Satz 4.4 sehr pessimistisch ist. Präziser ergibt sich aus dem Beweis in [Go73] sogar, daß man in Satz 4.4 statt $\lambda \in \{0, 1, \ldots, (n-1)^2+1\}$ sogar $\lambda \in \{0, 1, \ldots, (\deg(\alpha)-1)(\deg(\beta)-1)+1\}$ verlangen kann.

Aus Algorithmus 4.5 ergibt sich der folgende Satz:

4.7. Satz
Sei $\mathcal{O}$ eine Ordnung mit Basis $\omega_1, \ldots, \omega_n$. Dann kann man das Minimalpolynom eines primitiven Elements ϱ mit $\mathbb{Q}(\varrho) = \mathbb{Q}(\omega_1, \ldots, \omega_n)$ in Zeit

$$O(n^7 \log^2(n |\mathrm{MT}(\Omega)|))$$

berechnen. In dieser Zeit läßt sich auch eine Transformationsmatrix

$$T = (\underline{r}_0, \underline{r}_1, \ldots, \underline{r}_{n-1}) \in \mathbb{Z}^{n\times n} \textit{ mit } |T| \le n^{8n-1} |\,\mathrm{MT}(\Omega)|^{2n-1}$$

berechnen, so daß $\Omega T = R$, *wobei* $\Omega = (\omega_1, \ldots, \omega_n)$ *und* $R = (\det M_\varrho, \varrho, \ldots, \varrho^{n-1})$.

Beweis:
Wir müssen $O(n^3)$–mal die Körpergrade der von zwei gegebenen Elementen erzeugten Körper miteinander vergleichen. Dazu müssen wir jeweils die Minimalpolynome der zu vergleichenden Elemente berechnen. Dies dauert nach Satz 4.2 jeweils Zeit $O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)||\underline{r}_1|))$, wobei $\underline{r}_1$ den Koeffizientenvektor von ϱ bezeichnet. Da nach Konstruktion von ϱ stets $|\underline{r}_1| \le n^2$ gilt, benötigt man jeweils Zeit $O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|))$ um ein Minimalpolynom zu berechnen. Somit ergibt sich die angegebene Zeitschranke.

Nun wollen wir sehen, wie man die Matrix T berechnen kann. Eine Darstellung von ϱ als Linearkombination der Zahlen $\omega_1, \ldots, \omega_n$ liefert uns schon der Algorithmus. Die Potenzen von ϱ können wir dann mit der Multiplikationstafel effizient berechnen. Es bezeichne r_{ij} die i-te Komponente des Koeffizientenvektors $\underline{r}_j$ von ϱ^j, d.h. es ist

$$\varrho^j = \sum_{i=1}^n \omega_i r_{ij}.$$

Dann gilt:

$$\begin{aligned}
\varrho^{j+1} &= \left(\sum_{i=1}^n r_{ij}\omega_i\right) \varrho \\
&= \left(\sum_{i=1}^n r_{ij}\omega_i\right) \left(\sum_{k=1}^n r_{k1}\omega_k\right) \\
&= \sum_{i=1}^n \sum_{k=1}^n r_{ij} r_{k1} \omega_i \omega_k \\
&= \sum_{i=1}^n \sum_{k=1}^n \sum_{l=1}^n r_{ij} r_{k1} w_{ikl} \omega_l.
\end{aligned}$$

Also erhalten wir folgende Rekursionsformel:

$$r_{l,j+1} = \sum_{i=1}^n \left(r_{ij} \sum_{k=1}^n (r_{k1} w_{ikl}) \right) \text{ für alle } 1 \le j < n, 1 \le l \le n. \tag{4.5}$$

Da nach Konstruktion $|r_{k1}| \le n^2$ ist, erhalten wir also für $|\underline{r}_{j+1}| = \max_{1\le l\le n} |r_{l,j+1}|$:

$$|\underline{r}_{j+1}| \le n^4 |\underline{r}_j||\,\mathrm{MT}(\Omega)| \text{ für alle } 1 \le j < n.$$

Daher können wir mit vollständiger Induktion leicht zeigen:

$$|\underline{r}_j| \leq n^{4j-2} |\operatorname{MT}(\Omega)|^{j-1} \text{ für alle } 1 \leq j \leq n. \tag{4.6}$$

Zur Berechnung von $r_{l,j+1}$ gemäß (4.5) müssen wir also zunächst n^2 Multiplikationen von Zahlen mit $O(\log n)$ Bit mit Zahlen mit $O(\log|\operatorname{MT}(\Omega)|)$ Bit ausführen und je n Produkte aufsummieren. Die n Ergebnisse haben $O(\log n + \log|\operatorname{MT}(\Omega)|)$ Bit und müssen nun jeweils mit Zahlen der Länge $O(j(\log n + \log|\operatorname{MT}(\Omega)|))$ Bit multipliziert werden. Anschließend wird wieder aufsummiert. Also erhalten wir wegen $j \leq n$ einen Zeitaufwand von

$$\begin{aligned}
& O(\underbrace{n^2 \log n \log|\operatorname{MT}(\Omega)|}_{n^2 \text{ Multiplikationen}} + \underbrace{n^2(\log n + \log|\operatorname{MT}(\Omega)|)}_{n^2 \text{ Additionen}} + \\
& \quad \underbrace{nn(\log n + \log|\operatorname{MT}(\Omega)|)^2}_{\text{weitere } n \text{ Multiplikationen}} + \underbrace{nn(\log n + \log|\operatorname{MT}(\Omega)|)}_{\text{weitere } n \text{ Additionen}}) \\
= \; & O(n^2 \log n \log|\operatorname{MT}(\Omega)| + n^2 \log^2(n|\operatorname{MT}(\Omega)|)) \\
= \; & O(n^2 \log^2(n|\operatorname{MT}(\Omega)|)).
\end{aligned}$$

Die Berechnung aller Koeffizienten einer Potenz dauert n-mal so lange, also dauert die Berechnung aller n Potenzen Zeit

$$O(n^4 \log^2(n|\operatorname{MT}(\Omega)|)) \in O(n^7 \log^2(n|\operatorname{MT}(\Omega)|)).$$

Sei nun $m_\varrho(X) = X^n + m_{n-1}X^{n-1} + \cdots + m_1 X + m_0$ das Minimalpolynom von ϱ. Nachdem wir $\varrho^1, \ldots, \varrho^n$ berechnet haben, können wir diese Potenzen gemäß dem Minimalpolynom zu $(-1)^n m_0 = \det(M_\varrho)$ reduzieren. Damit erhalten wir also die angegebene Basis, wobei die Transformationsmatrix offensichtlich Koeffizienten aus $\mathbb{Z}$ hat.

Zum Abschluß müssen wir also eine Darstellung von m_0 als Koeffizientenvektor berechnen. Es gilt $(-1)^n m_0 = (-1)^{n+1}\varrho^n - (-1)^n m_{n-1}\varrho^{n-1} - \cdots - (-1)^n m_1 \varrho$, also können wir den Koeffizientenvektor $\underline{r}_0$ für $(-1)^n m_0$ als entsprechende Linearkombination aus $\underline{r}_n, \ldots, \underline{r}_1$ berechnen. Nach Satz 4.2 ist dabei $|m_i| \leq (n^2|\underline{r}_1||\operatorname{MT}(\Omega)|)^n \leq n^{4n}|\operatorname{MT}(\Omega)|^n$ für alle $0 \leq i \leq n-1$. Mit der Abschätzung für die $\underline{r}_j$ (Gleichung (4.6)) ergibt sich dann insgesamt:

$$|T| \leq n\left(n^{4n}|\operatorname{MT}(\Omega)|^n n^{4n-2}|\operatorname{MT}(\Omega)|^{n-1}\right) \leq n^{8n-1}|\operatorname{MT}(\Omega)|^{2n-1}.$$

Für die n Skalarmultiplikationen und die n Vektoradditionen, die wir ausführen müssen, um den letzten Koeffizientenvektor zu berechnen, ergibt sich ein Zeitaufwand von

$$O(\underbrace{n^2}_{\substack{\text{Zahl der}\\ \text{Elementaroperationen}}} \underbrace{(n \log(n|\operatorname{MT}(\Omega)|)}_{\text{Länge 1. Faktor}} \underbrace{n \log(n|\operatorname{MT}(\Omega)|)}_{\text{Länge 2. Faktor}} + \underbrace{n \log(n|\operatorname{MT}(\Omega)|)}_{\text{Länge der Summe}}))$$

$$= O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|)) \in O(n^7 \log^2(n|\,\mathrm{MT}(\Omega)|)).$$

Also erhalten wir in der angegebenen Zeit die Matrix T. $\square$

4.8. Bemerkung
Man beachte insbesondere, daß das berechnete T *keine* unimodulare Transformation ist, d.h. ϱ und seine Potenzen bilden keine Basis der Ordnung $\mathcal{O}$, sondern nur eine Basis des Körpers! Insbesondere werden wir später auf ganz andere Weise eine unimodulare Transformation berechnen, so daß wir 1 als erste Komponente der neuen Basis erhalten.

4.1.3 Nullstellen von Minimalpolynomen

Nun wollen wir die komplexen Nullstellen des Minimalpolynoms $m_\varrho(X)$ des primitiven Elements ϱ mit vorgegebener Präzision q berechnen. Da ϱ ein primitives Element ist, hat m_ϱ Grad n. Nach einem Resultat von Schönhage ([Sch82], Theorem 2.1, S.6) können wir für $q \in \mathbb{N}$ in Zeit

$$O((n^3 \log n + qn^2) \log(nq) \log\log(nq))$$

Linearfaktoren $L_j(X) = u_j X + v_j$ eines Polynoms $P(X) = a_n X^n + \cdots + a_0$ mit komplexen Koeffizienten und $\sum_{k=0}^n |a_k| \leq 1$ berechnen, so daß $P(X) - L_1(X) \cdots L_n(X) = b^n X^n + \cdots + b_0$ die Ungleichung $\sum_{k=0}^n |b_k| \leq 2^{-q}$ erfüllt. Dabei benutzt man schnelle Multiplikationsmethoden mit Zeitkomplexität wie in Bemerkung 1.2 angegeben. Da man unter Umständen mit sehr großer Präzision, also mit sehr langen Zahlen rechnen muß, ist dies hier auch sinnvoll.

Daraus leitet der Autor eine Laufzeit von $O(n^3(q + \log n) \log(nq) \log\log(nq))$ für einen Algorithmus ab, der für beliebige Polynome die Nullstellen mit Betrag ≤ 1 bzw. die Kehrwerte der Nullstellen mit Betrag > 1 mit Präzision q bestimmt. (siehe [Sch82], Theoreme 19.1 und 19.2, S.63). Für unser Verfahren benötigen wir jedoch auch die Nullstellen mit Betrag > 1 mit einer gewissen Genauigkeit, nicht nur die Kehrwerte. Daher beweisen wir folgende Variante des Ergebnisses von Schönhage:

4.9. Satz

Die n komplexen Nullstellen des Minimalpolynoms des berechneten primitiven Elements aus $\mathcal{O}$, können in Zeit

$$O(n^3(\log n + q + \log|\operatorname{MT}(\Omega)|)\log(n(q+\log|\operatorname{MT}(\Omega)|))\log\log(n(q+\log|\operatorname{MT}(\Omega)|)))$$

mit Präzision q berechnet werden.

Beweis:

Nach dem Verfahren von Schönhage können wir in der angegebenen Zeit die Nullstellen mit Betrag ≤ 1 mit Präzision q berechnen. Außerdem können wir in der angegebenen Zeit die Kehrwerte der Nullstellen mit Betrag > 1 mit Präzision $\lceil 8\log n + 2\log|\operatorname{MT}(\Omega)|\rceil + q + 4$ berechnen. Wir werden nun zeigen, daß dies reicht, um auch die Nullstellen mit Betrag > 1 mit der geforderten Präzision anzunähern:

Sei also z_j eine Nullstelle des Polynoms mit $|z_j| > 1$ und u_j eine Näherung für $\frac{1}{z_j}$ mit der angegebenen Präzision. Dann können wir also mit Verfahren zur schnellen Multiplikation eine Approximation $\tilde{z}_j$ an $\frac{1}{u_j}$ mit Präzision $q+3$ in der vorgegebenen Zeit berechnen. Dann gilt unter Verwendung von (2.1):

$$\begin{aligned}
|\tilde{z}_j - z_j| &\leq |\tilde{z}_j - \frac{1}{u_j}| + |\frac{1}{u_j} - z_j| \\
&\leq 2^{-q-3} + |\frac{z_j}{u_j}||u_j - \frac{1}{z_j}| \\
&\leq 2^{-q-3} + |\frac{z_j}{u_j}|2^{-q-4}\frac{1}{n^8|\operatorname{MT}(\Omega)|^2}.
\end{aligned}$$

Aufgrund der Rechengenauigkeit gilt aber:

$$|u_j| \geq \frac{1}{|z_j|} - 2^{-q-4}\frac{1}{n^4|\operatorname{MT}(\Omega)|}.$$

Nach der Abschätzung (4.3) aus Satz 4.2 gilt $|z_j| \leq n^4|\operatorname{MT}(\Omega)|$, da das Minimalpolynom zu einem Koeffizientenvektor gehört, dessen Komponenten nach Satz 4.4 durch n^2 beschränkt sind. Damit ergibt sich:

$$\begin{aligned}
|u_j| &\geq \frac{1}{n^4|\operatorname{MT}(\Omega)|} - 2^{-q-4}\frac{1}{n^4|\operatorname{MT}(\Omega)|} \\
\iff \frac{1}{|u_j|} &\leq \frac{n^4|\operatorname{MT}(\Omega)|}{1-2^{-q-4}} \\
\implies \frac{|z_j|}{|u_j|} &\leq \frac{n^8|\operatorname{MT}(\Omega)|^2}{1-2^{-q-4}}.
\end{aligned}$$

Damit erhalten wir für unsere berechnete Nullstelle $\tilde{z}_j$:

$$\begin{aligned} |\tilde{z}_j - z_j| &\leq 2^{-q-3} + \frac{2^{-q-4}}{1 - 2^{-q-4}} \\ &< 2^{-q-3} + 2^{-q-3} = 2^{-q-2}, \text{ da } 1 - 2^{-q-4} > \frac{1}{2}. \end{aligned}$$

Wenn wir schließlich $\tilde{z}_j$ richtig auf Präzision q runden, was anhand der zusätzlichen Stellen und unter Berücksichtigung der vorangegangenen Auf– oder Abrundungen möglich ist, erhalten wir also ein Ergebnis der gewünschten Präzision. □

4.1.4 Konstruktion der Konjugiertenvektoren

Sei eine Ordnung $\mathcal{O}$ durch die Multiplikationstafel bezüglich einer Basis Ω gegeben. Um die Konjugiertenvektoren der Basis Ω mit Genauigkeit q zu bestimmen, können wir wie folgt vorgehen:

1. Zunächst berechnen wir das Minimalpolynom eines primitiven Elements ϱ und eine Transformationsmatrix T wie in Satz 4.7.

2. Dann berechnen wir den Konjugiertenvektor $\underline{\varrho}$ zu ϱ mit Genauigkeit $q + n + \left\lceil (8n^2 - \frac{9}{2}n - \frac{11}{2}) \log n + (2n^2 - 2n - 1) \log |\operatorname{MT}(\Omega)| \right\rceil$.

3. Daraus berechnen wir die Konjugiertenvektoren $\underline{\varrho}^j$ für $2 \leq j \leq n-1$.

4. Nun berechnen wir eine Darstellung der Basiselemente als Linearkombinationen von $\det(M_\varrho), \varrho, \ldots, \varrho^{n-1}$.

5. Zum Schluß berechnen wir noch die entsprechenden Linearkombinationen der Vektoren $\underline{\det(M_\varrho)}, \underline{\varrho}, \ldots, \underline{\varrho}^{n-1}$ und erhalten damit die Konjugiertenvektoren der Basis mit Genauigkeit q.

Daraus ergibt sich:

4.10. Satz
Sei eine Ordnung $\mathcal{O}$ durch die Multiplikationstabelle $\mathrm{MT}(\Omega)$ *einer Basis* $\Omega = (\omega_1, \ldots, \omega_n)$ *gegeben. Dann lassen sich die Konjugiertenvektoren* $\underline{\omega}_1, \ldots, \underline{\omega}_n$ *mittels des angegebenen Verfahrens in Zeit*

$$O((n^3 q + n^5 \log(n|\mathrm{MT}(\Omega)|)) \log(n(q + \log|\mathrm{MT}(\Omega)|)) \log\log(n(q + \log|\mathrm{MT}(\Omega)|)) \\ +n^7 \log^2(n|\mathrm{MT}(\Omega)|))$$

mit Präzision q berechnen.

Beweis:
Für die Berechnung des Minimalpolynoms eines primitiven Elements und einer zugehörigen Transformation T benötigen wir nach Satz 4.7 Zeit

$$O(n^7 \log^2(n|\mathrm{MT}(\Omega)|)).$$

Zur Berechnung der Nullstellen dieses Polynoms mit Genauigkeit

$$q + n + \left\lceil (8n^2 - \frac{9}{2}n - \frac{11}{2}) \log n + (2n^2 - 2n - 1) \log|\mathrm{MT}(\Omega)| \right\rceil$$

benötigen wir nach Satz 4.9 Zeit

$$O((n^3 q + n^5(\log n + \log|\mathrm{MT}(\Omega)|)) \log(n(q + \log|\mathrm{MT}(\Omega)|)) \log\log(n(q + \log|\mathrm{MT}(\Omega)|))).$$

Nun müssen wir die Nullstellen noch so sortieren, daß die reellen Nullstellen zu Anfang stehen, und aus jedem Paar von konjugiert–komplexen Nullstellen nehmen wir nur eine Nullstelle, die wir in Real- und Imaginärteil zerlegen. Damit erhalten wir den Konjugiertenvektor von ϱ. Der Zeitbedarf für dieses Umsortieren fällt nicht weiter ins Gewicht.

Wenn wir die Konjugiertenvektoren der Potenzen von ϱ durch iterierte Multiplikation berechnen, wird der Fehler jeweils höchstens um einen Faktor $2n^4|\mathrm{MT}(\Omega)|$ größer, da die Einträge in $\underline{\varrho}$ durch $n^4|\mathrm{MT}(\Omega)|$ beschränkt sind (nach Satz 4.2,(4.3)) und ein Eintrag in $\underline{\varrho^i}$ als Summe zweier Produkte berechnet wird. Damit erhalten wir für jeden Eintrag in einem Konjugiertenvektor unserer Basis $\underline{\det(M_\varrho)}, \underline{\varrho}, \ldots, \underline{\varrho^{n-1}}$ einen Fehler δ mit

$$\begin{aligned} \delta &\le 2^{-q-n-\left\lceil (8n^2 - \frac{9}{2}n - \frac{11}{2}) \log n - (2n^2-2n-1) \log|\mathrm{MT}(\Omega)| \right\rceil} (2n^4|\mathrm{MT}(\Omega)|)^{n-2} \\ &\le 2^{-q-1-\left\lceil (8n^2 - \frac{17}{2}n + \frac{3}{2}) \log n - (2n^2-3n+1) \log|\mathrm{MT}(\Omega)| \right\rceil}, \end{aligned}$$

da wir insgesamt $(n-2)$–mal eine solche fehlerverstärkende Multiplikation ausführen. Den Vektor zu $\underline{\det(M_\varrho)}$ können wir sogar präzise bestimmen, denn alle Komponenten, die Realteile einer Einbettung in $\mathbb{C}$ sind, stimmen gerade mit dem $(-1)^n$–fachen des letzten Koeffizienten des charakteristischen Polynoms überein, alle Komponenten die Imaginärteile sind, sind 0: $\det(M_\varrho)$ ist nämlich eine ganze Zahl und daher unter allen Einbettungen invariant. Anschließend können wir also alle Vektoren auf Präzision $q+1+\left\lceil(8n^2-\frac{17}{2}n+\frac{3}{2})\log n+(2n^2-3n+1)\log|\operatorname{MT}(\Omega)|\right\rceil$ runden, da wir die weiteren evtl. fehlerhaften Stellen nicht mehr benötigen.

Insgesamt müssen wir zur Berechnung dieser Vektoren $O(n^2)$ Multiplikationen von Zahlen mit $O(q+n^2\log(n|\operatorname{MT}(\Omega)|))$ Bit ausführen, da nach Satz 4.2,(4.3) die Einträge in $\underline{\varrho}$ durch $n^4|\operatorname{MT}(\Omega)|$ beschränkt sind, und also die Einträge in allen Potenzen durch $(n^4|\operatorname{MT}(\Omega)|)^{n-1}$. Daher haben wir nur $O(n\log(n|\operatorname{MT}(\Omega)|))$ Bit vor dem Komma und soviele hinter dem Komma, wie die Präzision angibt. Der Zeitbedarf für die $O(n^2)$ Multiplikationen ist also unter Verwendung schneller Multiplikationstechniken um einen Faktor n geringer als der Aufwand für die Berechnung der Nullstellen. Die ebenfalls anfallenden $O(n^2)$ Additionen benötigen sogar noch weniger Zeit.

Um eine Darstellung der Basiselemente als Linearkombination von $\det(M_\varrho),\underline{\varrho},\ldots,\underline{\varrho}^{n-1}$ zu erhalten, würde es genügen, die in Schritt (1) erhaltene Matrix T zu invertieren. Leider ist diese Matrix im allgemeinen jedoch nicht aus $\operatorname{GL}(n,\mathbb{Z})$. Da wir jedoch wissen, daß die Multiplikation mit T die Basis Ω wieder auf eine Basis des Körpers $\mathbb{K}$ abbildet, wissen wir, daß $\det(T)$ nicht Null sein kann. Daher können wir die adjungierte Matrix von T berechnen. Diese erhalten wir nach Korollar 1.17 in Zeit

$$O(\left[n^4\log(n\log(n|T|))+n^3\log(n|T|)\right]\log(n|T|)),$$

also mit der Schranke für T aus Satz 4.7 in Zeit

$$O(\left[n^4\log(n\log(n|\operatorname{MT}(\Omega)|))+n^4\log(n|\operatorname{MT}(\Omega)|)\right]n\log(n|\operatorname{MT}(\Omega)|))$$

$$=O(n^5\log^2(n|\operatorname{MT}(\Omega)|)).$$

Dabei berechnen wir automatisch auch die Determinante, die wir später noch brauchen. Nach Satz 4.7 haben die Einträge in T Größe $|T|\le n^{8n-1}|\operatorname{MT}(\Omega)|^{2n-1}$, daher erhalten wir laut Korollar 1.17 in der adjungierten Matrix Einträge $\le(\sqrt{n}n^{8n-1}|\operatorname{MT}(\Omega)|^{2n-1})^{n-1}$ $=n^{8n^2-\frac{17}{2}n+\frac{1}{2}}|\operatorname{MT}(\Omega)|^{2n^2-3n+1}$. Gemäß dieser Matrix können wir dann aus der Basis $\underline{\det(M_\varrho)},\underline{\varrho},\ldots,\underline{\varrho}^{n-1}$ entsprechende Linearkombinationen berechnen. Aufgrund der Schranke für die Einträge der adjungierten Matrix verstärkt die Multiplikation mit einem Matrixeintrag den Fehler höchstens um den Faktor $n^{8n^2-\frac{17}{2}n+\frac{1}{2}}|\operatorname{MT}(\Omega)|^{2n^2-3n+1}$.

Die Berechnung einer Linearkombination aus $n-1$ gerundeten Konjugiertenvektoren und einem exakten Vektor vergrößert den Fehler also um weniger als den Faktor

$$n^{8n^2-\frac{17}{2}n+\frac{3}{2}}|\,\mathrm{MT}(\Omega)|^{2n^2-3n+1}.$$

Damit erhalten wir also Vektoren deren Komponenten von den exakten Werten höchstens um 2^{-q-1} abweichen. Zum Schluß dividieren wir die erhaltenen Konjugiertenvektoren durch $\det(T)\neq 0$, was die Präzision nicht verschlechtert, da $\mathbb{Z}\ni|\det(T)|\geq 1$. Also erhalten wir wegen $\frac{1}{\det(T)}\,\mathrm{ad}(T)=T^{-1}$ das gewünschte Ergebnis und zwar mit der vorgegebenen Genauigkeit und können nun auf Präzision q runden bzw. bei der Division gleich entsprechend wenige Stellen berechnen, wobei wir evtl. wieder Informationen benötigen, ob wir zuvor auf– oder abgerundet haben.

Die benötigten n^3 Multiplikationen bzw. Additionen zur Berechnung der n Linearkombinationen, bzw. die n^2 Divisionen durch die Determinante benötigen wegen der Größe der beteiligten Zahlen mit Methoden zur schnellen Multiplikation wiederum nicht länger als die Berechnung der Nullstellen. Somit ergibt sich der im Satz angegebene Gesamtaufwand. □

4.2 Hinzufügen von 1 zur Basis

In diesem Abschnitt wollen wir die Basis $\Omega=(\omega_1,\ldots,\omega_n)$ von $\mathcal{O}$ so modifizieren, daß wir eine Basis $\Omega'=(\omega_1',\ldots,\omega_n')$ mit $\omega_1'=1$ erhalten. Dazu müssen wir zunächst eine Darstellung der 1 mittels der Basis $\omega_1,\ldots,\omega_n$ bestimmen, was immer möglich ist, da 1 auf jeden Fall ein Element von $\mathcal{O}$ ist. Den dabei erhaltenen Koeffizientenvektor ergänzen wir dann zu einer unimodularen Matrix, mit der wir anschließend Ω transformieren können.

Um die Darstellung in der Form

$$1=\sum_{i=1}^{n}x_i\omega_i \quad \text{mit } x_1,\ldots,x_n\in\mathbb{Z} \tag{4.7}$$

zu finden, multiplizieren wir die Gleichung mit ω_1 und erhalten also

$$\sum_{i=1}^{n}x_i\omega_1\omega_i=\omega_1$$

und somit unter Benutzung der Multiplikationstafel:

$$\sum_{i=1}^{n} x_i \sum_{k=1}^{n} w_{1ik}\omega_k = \omega_1$$

Wenn wir also eine Darstellung mit Koeffizientenvektoren wählen, statt die Basis explizit einzusetzen, erhalten wir:

$$\sum_{i=1}^{n} x_i \begin{pmatrix} w_{1i1} \\ \vdots \\ w_{1in} \end{pmatrix} = \underline{e}_1, \tag{4.8}$$

wobei $\underline{e}_1$ der erste Einheitsvektor ist.

Da die Elemente aus Ω linear unabhängig über $\mathbb{Z}$ sind, sind auch die Elemente aus $\{\omega_1\omega_i : 1 \le i \le n\}$ linear unabhängig. Wären die Elemente aus $\{\omega_1\omega_i : 1 \le i \le n\}$ nämlich linear abhängig, dann gäbe es eine Linearkombination $\sum_{i=1}^{n} a_i\omega_1\omega_i = 0$. Dann wäre aber auch $\sum_{i=1}^{n} a_i N(\omega_1)\omega_i = 0$, wobei $N(\omega_1)$ die Norm von ω_1 bezeichnet. Da aber ω_1 eine Zahl $\neq 0$ aus der Ordnung $\mathcal{O}$ ist und also nach Satz 1.40 $N(\omega_1) \in \mathbb{Z} \setminus \{0\}$, wären dann auch die Elemente aus Ω linear abhängig über $\mathbb{Z}$.

Da also die Zahlen $\omega_1\omega_1, \ldots, \omega_1\omega_n$ linear unabhängig über $\mathbb{Z}$ sind, sind aber auch die Vektoren $(w_{1i1}, \ldots, w_{1in})^T, 1 \le i \le n$ linear unabhängig, also hat das Gleichungssystem (4.8) höchstens eine und also genau eine Lösung über $\mathbb{Z}$. Damit gilt:

4.11. Satz

Sei eine Ordnung $\mathcal{O}$ gegeben durch die Multiplikationstafel $\mathrm{MT}(\Omega) = (w_{ijk})_{1 \le i,j,k \le n}$ bezüglich einer Basis $\Omega = (\omega_1, \ldots, \omega_n)$. Dann kann eine Darstellung der 1 in der Form $\sum_{i=1}^{n} x_i\omega_i = 1$ mit $\|\underline{x}\|_\infty \le (\sqrt{n}|\,\mathrm{MT}(\Omega)|)^{n-1}$ in Zeit $O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|))$ bestimmt werden.

Beweis:

Die Einträge in (4.8) sind beschränkt durch $|\,\mathrm{MT}(\Omega)|$, also ergibt sich mit (1.3) die angegebene Schranke für $\|\underline{x}\|_\infty$, da die Matrix des Gleichungssystems aus linear unabhängigen Spaltenvektoren besteht und der Betrag der Determinanten also mindestens 1 ist.

Mit Korollar 1.15 ergibt sich eine Laufzeit von

$$\begin{aligned} &O(\left[n^4 \log(n \log(n|\,\mathrm{MT}(\Omega)|)) + n^3 \log(n|\,\mathrm{MT}(\Omega)|)\right] \log(n|\,\mathrm{MT}(\Omega)|)) \\ &\quad = O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|)). \end{aligned}$$

□

4.12. Bemerkung

Ebenso wie wir Gleichung (4.7) mit ω_1 multipliziert haben, um daraus (4.8) zu erhalten, können wir (4.7) auch mit einem anderen ω_j, $2 \leq j \leq n$ multiplizieren und erhalten dann ein entsprechendes Gleichungssystem, nämlich $\sum_{i=1}^{n} x_i \left(w_{ji1}, \ldots, w_{jin}\right)^T = \underline{e}_j$. Auf diese Weise ist es mit wenig zusätzlichem Aufwand möglich, ein Gleichungssystem mit möglichst kleinen Einträgen in der Systemmatrix auszuwählen und so möglichst kleine Lösungsvektoren zu garantieren. Ob sich dieser Aufwand lohnt und ob man damit wirklich bessere Ergebnisse erhält, könnte man anhand einer Implementierung untersuchen.

Den Vektor $\underline{x}$ ergänzen wir nun zu einer unimodularen Matrix. Dazu benutzen wir den folgenden Algorithmus aus [Bu92] bzw. [BuSp92], wobei wir allerdings im auf den Algorithmus folgenden Satz eine Schranke für die Größe des Ergebnisses zeigen werden, die deutlich besser ist als die in [BuSp92] angegebene:

4.13. Algorithmus (Basisergänzung)

Eingabe: Vektor $\underline{x} = (x_1, \ldots, x_n)$ mit $\mathrm{ggT}(x_1, \ldots, x_n) = 1$.

Ausgabe: Matrix $X = (\underline{x}_1, \ldots, \underline{x}_n) \in \mathrm{GL}(n, \mathbb{Z})$ mit $\underline{x}_1 = \underline{x}$.

1. $y_i := \left\{ \begin{smallmatrix} x_i & \text{für } 1 \leq i \leq n \\ -1 & \text{für } i = n+1 \end{smallmatrix} \right.$
2. $\underline{b}_i := \left\{ \begin{smallmatrix} \underline{e}_i & \text{für } 1 \leq i \leq n \\ \underline{x} & \text{für } i = n+1 \end{smallmatrix} \right.$
3. FOR $i = 1$ TO $n-1$ DO
   (a) Berechne a, b mit $\mathrm{ggT}(y_1, \ldots, y_{i+1}) = a y_{i+1} - b\, \mathrm{ggT}(y_1, \ldots, y_i)$;
   /* In [BuSp92] sind irrtümlich die Rollen von a und b vertauscht! */
   (b) $c := \frac{\mathrm{ggT}(y_1, \ldots, y_i)}{\mathrm{ggT}(y_1, \ldots, y_{i+1})}$; $d := \frac{y_{i+1}}{\mathrm{ggT}(y_1, \ldots, y_{i+1})}$;
   (c) $(\underline{b}_i, \underline{b}_{i+1}) := (\underline{b}_i, \underline{b}_{i+1}) \begin{pmatrix} a & c \\ b & d \end{pmatrix}$;
   (d) $y_{i+1} := \mathrm{ggT}(y_1, \ldots, y_{i+1})$; $y_i := 0$;
   (e) IF $y_{i+1} = 1$ THEN $X := (\underline{x}, \underline{b}_1, \ldots, \underline{b}_i, \underline{b}_{i+2}, \ldots, \underline{b}_n)$; BREAK FI;
4. OD.

Dann erhalten wir für die Größe der Einträge in X und für die Laufzeit des Algorithmus folgende Resultate:

4.14. Satz

Sei $\underline{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ mit $\mathrm{ggT}(x_1, \ldots, x_n) = 1$. Dann benötigt Algorithmus 4.13 Zeit $O(n^2 \log^2(\|\underline{x}\|_\infty))$, um zu dem gegebenen Vektor eine Matrix $X \in \mathrm{GL}(n, \mathbb{Z})$ mit $|X| \leq \|\underline{x}\|_\infty^3$ zu berechnen, deren erste Spalte $\underline{x}$ ist.

Beweis:

Wir beweisen zunächst die Korrektheit des Algorithmus. Dazu zeigen wir induktiv, daß nach jeder Iteration $y_1\underline{b}_1 + \ldots + y_{n+1}\underline{b}_{n+1} = 0$ gilt und daß $\underline{b}_1, \ldots, \underline{b}_{n+1}$ den Raum $\mathbb{Z}^n$ erzeugen. Daß dies nach der nullten Iteration, also zu Beginn der ersten Iteration gilt, ist nach Wahl der Startwerte klar.

Gelte die Invariante nach der $(i-1)$–ten Iteration. In der i–ten Iteration werden zunächst $a, b \in \mathbb{Z}$ berechnet mit $\mathrm{ggT}(y_1, \ldots, y_{i+1}) = ay_{i+1} - b\,\mathrm{ggT}(y_1, \ldots, y_i)$ und $c, d \in \mathbb{Z}$ mit

$$\begin{aligned} ad - bc &= a\frac{y_{i+1}}{\mathrm{ggT}(y_1, \ldots, y_{i+1})} - b\frac{\mathrm{ggT}(y_1, \ldots, y_i)}{\mathrm{ggT}(y_1, \ldots, y_{i+1})} \\ &= \frac{1}{\mathrm{ggT}(y_1, \ldots, y_{i+1})}\left(ay_{i+1} - b\,\mathrm{ggT}(y_1, \ldots, y_i)\right) \\ &= \frac{\mathrm{ggT}(y_1, \ldots, y_{i+1})}{\mathrm{ggT}(y_1, \ldots, y_{i+1})} = 1, \end{aligned}$$

d.h. $\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathrm{GL}(2, \mathbb{Z})$. Damit ist gesichert, daß auch am Ende der i–ten Iteration $\underline{b}_1, \ldots, \underline{b}_{n+1}$ den Raum $\mathbb{Z}^n$ erzeugen.

Wir bezeichnen ab jetzt für $j = i, i+1$ mit $\underline{b}'_j$ die Werte für $\underline{b}_j$ nach der $(i-1)$–ten Iteration und mit $\underline{b}_j$ die Werte nach der i–ten Iteration und analog verwenden wir y'_j und y_j ($j = 1, \ldots, i+1$). Dann gilt nach der i–ten Iteration:

$$\begin{aligned} \underbrace{y_i}_{=0}\underline{b}_i + y_{i+1}\underline{b}_{i+1} &= y_{i+1}(c\underline{b}'_i + d\underline{b}'_{i+1}) \\ &= \mathrm{ggT}(y'_1, \ldots, y'_{i+1})\left(\frac{\mathrm{ggT}(y'_1, \ldots, y'_i)}{\mathrm{ggT}(y'_1, \ldots, y'_{i+1})}\underline{b}'_i + \frac{y'_{i+1}}{\mathrm{ggT}(y'_1, \ldots, y_{i+1'})}\underline{b}'_{i+1}\right) \\ &= \underbrace{\mathrm{ggT}(y'_1, \ldots, y'_i)}_{=y'_i}\underline{b}'_i + y'_{i+1}\underline{b}'_{i+1}. \end{aligned}$$

Da sich jeweils nur $\underline{b}_i$ und $\underline{b}_{i+1}$ ändern, folgt damit auch:

$$y_1\underline{b}_1 + \ldots + y_{n+1}\underline{b}_{n+1} = y'_1\underline{b}'_1 + \ldots + y'_{n+1}\underline{b}'_{n+1} = 0,$$

d.h. die Invariante gilt auch am Ende der i–ten Iteration.

Wenn nun $y_i = 1$ für ein i gilt, dann läßt sich $\underline{b}_i$ offensichtlich durch eine $\mathbb{Z}$-Linearkombination der anderen Vektoren ausdrücken, kann also aus dem Erzeugendensystem gestrichen werden. Also leistet der Algorithmus das Verlangte.

Da $\mathrm{ggT}(x_1, \ldots, x_n) = 1$ werden im ungünstigsten Fall n-1 Iterationen des Algorithmus ausgeführt. Um die behauptete Schranke für $|X|$ zu beweisen, zeigen wir induktiv die folgende stärkere Hilfsbehauptung:
Für $0 \le i \le n-1$ gilt nach dem i-ten Schleifendurchlauf:

1. $\|\underline{b}_j\|_\infty \le \|\underline{x}\|_\infty^3$ und für alle $1 \le j \le i$.
2. $\|\underline{b}_{i+1}\|_\infty \le \|\underline{x}\|_\infty \left|\frac{\mathrm{ggT}(x_1,x_2)}{\mathrm{ggT}(x_1,\ldots,x_{i+1})}\right|$ und $b_{k,i+1} = 0$ für alle $k > i+1$.
3. $\underline{b}_j = \underline{e}_j$ für $j > i+1$.

Induktionsanfang (Für $i = 0$ ist die Behauptung erfüllt):
Vor dem ersten Schleifendurchlauf ist die Behauptung nach Wahl der Startwerte klar.

Induktionsschluß ($i - 1 \longrightarrow i$):
Wir verwenden die Bezeichnungen $\underline{b}'_j$ und $\underline{b}_j$ ($j = i, i+1$) wie oben. In der i-ten Iteration ändern sich nur $\underline{b}_i$ und $\underline{b}_{i+1}$. Daher ist (3) nach Induktionsvoraussetzung klar. Ebenso ist (1) für $j \le i-1$ klar. Nun berechnen wir $\underline{b}_i = a\underline{b}'_i + b\underline{b}'_{i+1}$ und $\underline{b}_{i+1} = c\underline{b}'_i + d\underline{b}'_{i+1}$. Nach Induktionsvoraussetzung (2) ist $b'_{k,i} = 0$ für alle $k > i$ und wegen (3) ist $\underline{b}'_{i+1} = \underline{e}_{i+1}$. Daher ergibt sich zunächst $b_{k,i+1} = 0$ für alle $k > i+1$ und beim Berechnen der Linearkombinationen von $\underline{b}_i$ und $\underline{b}_{i+1}$ ist stets eine der beiden zu addierenden Komponenten $= 0$. Also gilt: $\|\underline{b}_i\|_\infty = \max(|a|\|\underline{b}'_i\|_\infty, |b|\|\underline{b}'_{i+1}\|_\infty)$ bzw. $\|\underline{b}_{i+1}\|_\infty = \max(|c|\|\underline{b}'_i\|_\infty, |d|\|\underline{b}'_{i+1}\|_\infty)$. Da nach Korollar 1.6 $a, b, d \le \|\underline{x}\|_\infty$ gilt, ergeben sich also (1) für $j = i$ und (2), indem wir die aktuellen Werte für a, b, c und d abschätzen, wobei wir ausnutzen, daß in der i–ten Iteration offensichtlich für alle $i \le k \le n$ gilt: $\mathrm{ggT}(x_1, \ldots, x_k) = \mathrm{ggT}(y_1, \ldots, y_k)$. Somit ergibt sich:

$$\begin{aligned}
\|\underline{b}_i\|_\infty &= \max(|a|\|\underline{b}'_i\|_\infty, |b|\|\underline{e}_{i+1}\|_\infty) \\
&\le \|\underline{x}\|_\infty^2 \left|\frac{\mathrm{ggT}(x_1,x_2)}{\mathrm{ggT}(x_1,\ldots,x_i)}\right| \text{ nach Induktionsvoraussetzung (2) und (3).} \\
&\le \|\underline{x}\|_\infty^3, \text{ da } \left|\frac{\mathrm{ggT}(x_1,x_2)}{\mathrm{ggT}(x_1,\ldots,x_i)}\right| \le \|\underline{x}\|_\infty,
\end{aligned}$$

also ist (1) erfüllt. Analog ergibt sich

$$\begin{aligned}
\|\underline{b}_{i+1}\|_\infty &= \max(|c|\|\underline{b}'_i\|_\infty, |d|\|\underline{e}_{i+1}\|_\infty) \\
&\leq \left|\frac{\mathrm{ggT}(x_1,\ldots,x_i)}{\mathrm{ggT}(x_1,\ldots,x_{i+1})}\right| \|\underline{x}\|_\infty \left|\frac{\mathrm{ggT}(x_1,x_2)}{\mathrm{ggT}(x_1,\ldots,x_i)}\right| \\
&= \|\underline{x}\|_\infty \left|\frac{\mathrm{ggT}(x_1,x_2)}{\mathrm{ggT}(x_1,\ldots,x_{i+1})}\right|,
\end{aligned}$$

d.h. auch (2) gilt. Damit erhalten wir also am Ende des Algorithmus die angegebene Schranke für $|X|$.

Dabei werden in jeder Iteration der erweiterte euklidische Algorithmus und 2 Divisionen ausgeführt. Darüber hinaus müssen $O(n)$ Multiplikationen zur Transformation der Vektoren ausgeführt werden. Alle auftretenden Zahlen sind dabei, wie wir gesehen haben, durch $\|\underline{x}\|_\infty^3$ beschränkt. Eine Iteration benötigt also Zeit $O(n\log^2\|\underline{x}\|_\infty)$. Daher brauchen wir insgesamt Zeit $O(n^2\log^2\|\underline{x}\|_\infty)$ um die Matrix zu berechnen. □

Wenn wir diese Ergebnisse zusammenfassen, erhalten wir folgende Aussage:

4.15. Satz

Wir benötigen Zeit

$$O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|)),$$

um zu einer gegebenen Multiplikationstafel einer Basis Ω eine Transformationsmatrix $X \in \mathrm{GL}(n,\mathbb{Z})$ mit

$$|X| \leq (\sqrt{n}|\,\mathrm{MT}(\Omega)|)^{3n-3}$$

zu berechnen, so daß die transformierte Basis ΩX die Zahl 1 als erstes Basiselement enthält.

Beweis:

Zunächst berechnen wir einen Vektor $\underline{x} = (x_1,\ldots,x_n)$ mit $\sum_{i=1}^n x_i\omega_i = 1$. Da $\underline{x}$ Lösung des Gleichungssystems (4.8) ist und also $\sum_{i=1}^n x_i w_{1i1} = 1$ gilt, ist

$$\mathrm{ggT}(x_1,\ldots,x_n) = 1,$$

sonst könnte man die Zahl 1 nicht als $\mathbb{Z}$–Linearkombination von $x_1,\ldots,x_n$ darstellen. Damit ist Algorithmus 4.13 anwendbar und liefert eine Matrix X, die das Gewünschte leistet. Die behauptete Schranke für $|X|$ und die Laufzeit ergeben sich unmittelbar aus Satz 4.11 und Satz 4.14. □

4.3 Transformation der Multiplikationstafel

Jetzt wollen wir $\Omega = (\omega_1, \ldots, \omega_n)$ mit einer gegebenen unimodularen Matrix T transformieren. Dazu wollen wir die Multiplikationstafel $\mathrm{MT}(\Omega T)$ bestimmen. Es sei dabei

$$\Omega T = (\Omega \underline{t}_1, \ldots, \Omega \underline{t}_n) =: (\tau_1, \ldots, \tau_n).$$

Dann suchen wir $(x_{ijk})_{1 \le i,j,k \le n} \in \mathbb{Z}^{n \times n \times n}$ mit

$$\tau_i \tau_j = \sum_{k=1}^{n} x_{ijk} \tau_k.$$

Fixiere nun i und j. Dann gilt also:

$$\begin{aligned} \tau_i \tau_j &= \Omega \underline{t}_i \Omega \underline{t}_j \\ &= \left(\sum_{k=1}^{n} \omega_k t_{ki} \right) \left(\sum_{l=1}^{n} \omega_l t_{lj} \right) \\ &= \sum_{k,l=1}^{n} t_{ki} t_{lj} \left(\sum_{m=1}^{n} w_{klm} \omega_m \right) \\ &= \Omega \underbrace{\left(\sum_{k,l=1}^{n} t_{ki} t_{lj} (w_{kl1}, \ldots, w_{kln})^T \right)}_{=: \underline{y}_{ij}} . \end{aligned}$$

Wir berechnen daher zunächst die Vektoren $\underline{y}_{ij}$ und lösen dann die Gleichungssysteme

$$\sum_{k=1}^{n} x_{ijk} \tau_k = \Omega \underline{y}_{ij}.$$

Unter Berücksichtigung von $\tau_k = \Omega \underline{t}_k$ erhalten wir daraus

$$\sum_{k=1}^{n} x_{ijk} \underline{t}_k = \underline{y}_{ij}, \tag{4.9}$$

denn die Zahlen τ_k, $1 \le k \le n$ sind ebenso wie die Zahlen ω_k, $1 \le k \le n$ über $\mathbb{Q}$ linear unabhängig, also kann die Linearkombination der rechten Seite nur dann mit der linken Seite übereinstimmen, wenn die Koeffizientenvektoren identisch sind. Wir müssen also nur noch die n^2 Gleichungssysteme (4.9) lösen und erhalten dann $\mathrm{MT}(\Omega T)$. Damit gilt:

4.16. Satz

Es sei eine Ordnung $\mathcal{O}$ *gegeben durch ihre Multiplikationstafel* $\mathrm{MT}(\Omega) = (w_{ijk})_{1 \leq i,j,k \leq n}$ *bezüglich einer Basis* Ω*. Ferner sei eine unimodulare Matrix* T *gegeben und es sei* $s(T)$ *eine obere Schranke für* $|T|$*, die gleichzeitig auch obere Schranke für* $|\,\mathrm{MT}(\Omega)|$ *ist, d.h.* $|T| + |\,\mathrm{MT}(\Omega)| = O(s(T))$*. Dann benötigen wir Zeit*

$$O(n^5 \log^2(ns(T))),$$

um die Multiplikationstafel $\mathrm{MT}(\Omega T)$ *zu bestimmen. Dabei gilt*

$$|\,\mathrm{MT}(\Omega T)| \leq n^{\frac{n+4}{2}} |T|^{n+1} |\,\mathrm{MT}(\Omega)|.$$

Beweis:

Wir müssen n^2 Gleichungssysteme lösen. Dabei müssen wir jeweils den Vektor $\underline{y}_{ij}$ berechnen, wozu zunächst jeweils n^2 Multiplikationen mit Zahlen der Länge $\log |T|$ auszuführen sind. Das Ergebnis muß dann jeweils mit $(w_{kl1}, \ldots, w_{kln})^T$ multipliziert werden. Bis hierher benötigen wir also Zeit $O(n^2(\log^2 |T| + n \log |T| \log |\,\mathrm{MT}(\Omega)|))$. Dann werden diese n^2 Vektoren komponentenweise aufaddiert, es ergibt sich also:

$$\|\underline{y}_{ij}\|_\infty \leq n^2 |T|^2 |\,\mathrm{MT}(\Omega)|. \tag{4.10}$$

Jede solche Summe ist also maximal $O(\log(n|T||\,\mathrm{MT}(\Omega)|))$ Bit lang. Daher erhalten wir einen solchen Vektor in Zeit $O(n^2(\log^2 |T| + n \log |T| \log |\,\mathrm{MT}(\Omega)| + n \log n|T||\,\mathrm{MT}(\Omega)|))$. Um also $\underline{y}_{ij}$ für alle $1 \leq i,j \leq n$ zu berechnen, benötigen wir Zeit $O(n^5(\log^2 s(T) + \log n))$.

Die n^2 Gleichungssysteme können wir simultan lösen, indem wir statt jeweils einen Vektor der rechten Seite gleich alle n^2 Vektoren der rechten Seiten transformieren. Damit benötigt eine Zeilenoperation des Gaußschen Algorithmus statt $O(n)$ nun $O(n^2)$ arithmetische Operationen, ansonsten läuft der Algorithmus wie üblich ab. Also benötigen wir nur n–mal solange, um die n^2 Gleichungssysteme zu lösen, wie um ein Gleichungssystem zu lösen. Da aber wegen (4.10) gilt:

$$\|\underline{y}_{ij}\| \leq \sqrt{n} n^2 |T|^2 |\,\mathrm{MT}(\Omega)|, \tag{4.11}$$

ist also $\|\underline{y}_{ij}\| \in O((ns(T))^{O(n)})$. Daher ist Korollar 1.15 anwendbar, denn T ist eine unimodulare Matrix, daher gibt es für alle n^2 Gleichungssysteme eine eindeutige Lösung in $\mathbb{Z}^n$. Also benötigen wir Zeit

$$O(\Big[n^5\log(n\log(ns(T))) + n^4\log(ns(T))\Big]\log(ns(T)))$$

$$= O(n^5\log^2(ns(T))),$$

um $\mathrm{MT}(\Omega T)$ aus den Werten der $\underline{y}_{ij}$ zu bestimmen. Da dies die Laufzeit zur Berechnung aller $\underline{y}_{ij}$ dominiert, erhalten wir insgesamt eine Laufzeit von

$$O(n^5\log^2(ns(T))).$$

Aus (4.11) folgt schließlich mit $|\det(T)| = 1$ und (1.3)

$$\begin{aligned} |\,\mathrm{MT}(\Omega T)| &\leq (\sqrt{n}|T|)^{n-1}\sqrt{n}n^2|T|^2|\,\mathrm{MT}(\Omega)| \\ &= n^{\frac{n+4}{2}}|T|^{n+1}|\,\mathrm{MT}(\Omega)|. \end{aligned}$$ $\Box$

Kapitel 5

Berechnung einer kurzen Darstellung

Wieder bezeichnen wir mit $\mathcal{O}$ eine Ordnung des algebraischen Zahlkörpers $\mathbb{K}$ und mit $\mathrm{MT}(\Omega) = (w_{ijk})_{1\leq i,j,k\leq n}$ die Multiplikationstafel von $\mathcal{O}$ bezüglich einer $\mathbb{Z}$–Basis $\Omega = (\omega_1, \ldots, \omega_n)$, wobei wieder n der Körpergrad von $\mathbb{K}$ über $\mathbb{Q}$ und $\sigma_1, \ldots, \sigma_n$ die Einbettungen von $\mathbb{K}$ in $\mathbb{C}$ sind. Dann bezeichnen wir gemäß Definition 1.41 mit $\underline{\Omega} = (\underline{\omega}_1, \ldots, \underline{\omega}_n)$ die Repräsentation der Basis mittels Konjugiertenvektoren.

5.1 Diskriminanten

Um zu definieren, wann eine Darstellung eine kurze Darstellung ist, benötigen wir noch einen zusätzlichen Begriff, nämlich die Diskriminante einer Ordnung.

5.1. Definition (Diskriminante)
Für eine $\mathbb{Q}$–Basis $\alpha_1, \ldots, \alpha_n$ von $\mathbb{K}$ bezeichnen wir mit

$$\Delta(\alpha_1, \ldots, \alpha_n) := \left[det \begin{pmatrix} \sigma_1(\alpha_1) & \ldots & \sigma_1(\alpha_n) \\ \vdots & \ddots & \vdots \\ \sigma_n(\alpha_1) & \ldots & \sigma_n(\alpha_n) \end{pmatrix} \right]^2$$

die Diskriminante von $\alpha_1, \ldots, \alpha_n$.

5.2. Bemerkung
Die Diskriminante hat folgende Eigenschaften:

- Sie ist offensichtlich unabhängig von der Reihenfolge der α_i und σ_j.
- Es ist stets $\Delta(\alpha_1, \ldots, \alpha_n) \in \mathbb{Q}$.
- Offensichtlich ist sie invariant unter unimodularen Transformationen der Basis.

Aufgrund dieser letzten Eigenschaft sind folgende Definitionen sinnvoll:

5.3. Definition
Unter der Diskriminanten $\Delta_{\mathcal{O}}$ einer Ordnung $\mathcal{O}$ verstehen wir die Diskriminante einer beliebigen $\mathbb{Z}$–Basis der Ordnung. Als Körperdiskriminante $\Delta_{\mathbb{K}}$ eines Körpers $\mathbb{K}$ bezeichnen wir die Diskriminante der maximalen Ordnung, d.h. des Ganzheitsringes $\mathcal{O}_{\mathbb{K}}$.

Unter einer kurzen Darstellung einer Ordnung $\mathcal{O}$ verstehen wir nun eine Darstellung, deren Länge polynomiell in $\log|\Delta_{\mathcal{O}}|$ und dem Körpergrad n ist. Im Falle der Darstellung mittels einer Multiplikationstafel heißt das also, daß jeder Eintrag in der Multiplikationstafel betragsmäßig durch $2^{n^{O(1)} \log^{O(1)} |\Delta_{\mathcal{O}}|}$ beschränkt ist.

Nun wollen wir demonstrieren, wie man mit den bisher vorgestellten Algorithmen eine solche Darstellung konstruieren kann. Dabei kann man nach der eigentlichen Reduktion noch eine Transformation ausführen, die gewährleistet, daß in der Basis, die zur Multiplikationstafel der Ausgabe gehört, die 1 als erstes Basiselement auftritt. Hierdurch werden die Einträge in der Multiplikationstafel zwar wieder sehr viel größer, jedoch bleiben sie durch $2^{n^{O(1)} \log^{O(1)} |\Delta_{\mathcal{O}}|}$ beschränkt.

5.2 Reduktion der Multiplikationstafel

In diesem ersten Schritt berechnen wir eine $\mathbb{Z}$-Basis $\underline{\Omega} = (\underline{\omega}_1, \ldots, \underline{\omega}_n)$ aus Konjugiertenvektoren. Dies geschieht mit Präzision $q \geq 2\log n + \log \operatorname{dft}(\underline{\Omega}) - \log \lambda_1(\underline{\Omega})$ laut Satz 4.10 in Zeit

$$O((n^3 q + n^5 \log(n|\,\mathrm{MT}(\Omega)|) \log(n(q + \log|\,\mathrm{MT}(\Omega)|)) \log\log(n(q + \log|\,\mathrm{MT}(\Omega)|))) \\ + n^7 \log^2(n|\,\mathrm{MT}(\Omega)|)).$$

Die Basis $\underline{\Omega}$ kann man zugleich auch als Basis eines Gitters auffassen, wie wir in Lemma 3.5 gesehen haben. Daher können wir also mittels Algorithmus 3.13 eine kürzere Basis des von $\underline{\Omega}$ erzeugten Gitters bestimmen. Wenn wir $\underline{\Omega}\{q\}$ als Eingabe für Algorithmus 3.13 benutzen, erhalten wir nach Satz 3.20 für $q \geq 2\log n + \log \mathrm{dft}(\underline{\Omega}) - \log \lambda_1(\underline{\Omega})$ in Zeit

$$O(n^6 \max(1, q + \log \lambda_n(\underline{\Omega}))^3)$$

eine Transformationsmatrix T mit $\underline{\Omega}T = (\underline{\tau}_1, \ldots, \underline{\tau}_n)$, so daß nach Satz 3.17 gilt:

$$\prod_{i=1}^{n} \|\underline{\tau}_i\| \leq 2^{\frac{n(n-1)}{4}} |\det(\underline{\Omega})| \left[1 + \mathrm{dft}(\underline{\Omega}) \left(\left(1 + 2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n - 1\right)\right] \left[1 + 2^{-q} n^2 \frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n . \tag{5.1}$$

Damit haben wir also erreicht, daß unsere anfänglich gegebene Ordnung durch eine Basis mit relativ kurzen Konjugiertenvektoren repräsentiert ist. Nun werden wir zeigen, daß auch die Einträge in der zugehörigen Multiplikationstafel kurz sein müssen. Dazu benötigen wir zwei technische Lemmata, die auch dazu dienen, die Abhängigkeit von $\underline{\Omega}$, die in (5.1) noch enthalten ist, zu eliminieren:

5.4. Lemma

Wenn man mit $2r_2$ die Zahl der nicht reellwertigen Einbettungen von $\mathbb{K}$ in $\mathbb{C}$ bezeichnet, dann gilt

$$|\det(\underline{\Omega})| = 2^{-r_2}\sqrt{|\Delta_{\mathcal{O}}|}.$$

Beweis:

Es ist $n = r_1 + 2r_2$, wobei r_1 die Zahl der reellwertigen Einbettungen von $\mathbb{K}$ in $\mathbb{C}$ ist und $2r_2$ die Zahl der echt komplexwertigen Einbettungen, wobei jeweils die $(r_1 + k)$-te und die $(r_1 + r_2 + k)$-te Einbettung durch Konjugation auseinander hervorgehen.

Dann ist also

$$\det(\underline{\Omega}) = \det \begin{pmatrix} \sigma_1(\omega_1) & \ldots & \sigma_1(\omega_n) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1}(\omega_1) & \ldots & \sigma_{r_1}(\omega_n) \\ \Re(\sigma_{r_1+1}(\omega_1)) & \ldots & \Re(\sigma_{r_1+1}(\omega_n)) \\ \Im(\sigma_{r_1+1}(\omega_1)) & \ldots & \Im(\sigma_{r_1+1}(\omega_n)) \\ \vdots & \ddots & \vdots \\ \Re(\sigma_{r_1+r_2}(\omega_1)) & \ldots & \Re(\sigma_{r_1+r_2}(\omega_n)) \\ \Im(\sigma_{r_1+r_2}(\omega_1)) & \ldots & \Im(\sigma_{r_1+r_2}(\omega_n)) \end{pmatrix}$$

$$= \det \begin{pmatrix} \sigma_1(\omega_1) & \dots & \sigma_1(\omega_n) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1}(\omega_1) & \dots & \sigma_{r_1}(\omega_n) \\ \frac{1}{2}(\sigma_{r_1+1}(\omega_1) + \sigma_{r_1+r_2+1}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+1}(\omega_n) + \sigma_{r_1+r_2+1}(\omega_n)) \\ \frac{1}{2}(\sigma_{r_1+1}(\omega_1) - \sigma_{r_1+r_2+1}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+1}(\omega_n) - \sigma_{r_1+r_2+1}(\omega_n)) \\ \vdots & \ddots & \vdots \\ \frac{1}{2}(\sigma_{r_1+r_2}(\omega_1) + \sigma_{r_1+2r_2}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+r_2}(\omega_n) + \sigma_{r_1+2r_2}(\omega_n)) \\ \frac{1}{2}(\sigma_{r_1+r_2}(\omega_1) - \sigma_{r_1+2r_2}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+r_2}(\omega_n) - \sigma_{r_1+2r_2}(\omega_n)) \end{pmatrix}$$

$$= \det \begin{pmatrix} \sigma_1(\omega_1) & \dots & \sigma_1(\omega_n) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1}(\omega_1) & \dots & \sigma_{r_1}(\omega_n) \\ \sigma_{r_1+1}(\omega_1) & \dots & \sigma_{r_1+1}(\omega_n) \\ \frac{1}{2}(\sigma_{r_1+1}(\omega_1) - \sigma_{r_1+r_2+1}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+1}(\omega_n) - \sigma_{r_1+r_2+1}(\omega_n)) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1+r_2}(\omega_1) & \dots & \sigma_{r_1+r_2}(\omega_n) \\ \frac{1}{2}(\sigma_{r_1+r_2}(\omega_1) - \sigma_{r_1+2r_2}(\omega_1)) & \dots & \frac{1}{2}(\sigma_{r_1+r_2}(\omega_n) - \sigma_{r_1+2r_2}(\omega_n)) \end{pmatrix}$$

$$= \left(\frac{-1}{2}\right)^{r_2} \det \begin{pmatrix} \sigma_1(\omega_1) & \dots & \sigma_1(\omega_n) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1}(\omega_1) & \dots & \sigma_{r_1}(\omega_n) \\ \sigma_{r_1+1}(\omega_1) & \dots & \sigma_{r_1+1}(\omega_n) \\ \sigma_{r_1+r_2+1}(\omega_1) & \dots & \sigma_{r_1+r_2+1}(\omega_n) \\ \vdots & \ddots & \vdots \\ \sigma_{r_1+r_2}(\omega_1) & \dots & \sigma_{r_1+r_2}(\omega_n) \\ \sigma_{r_1+2r_2}(\omega_1) & \dots & \sigma_{r_1+2r_2}(\omega_n) \end{pmatrix}$$

Da die Diskriminante unabhängig von der Reihenfolge der σ_i ist, folgt nun mit der Definition der Diskriminanten die Behauptung. □

5.5. Lemma

Für einen einzelnen Vektor der reduzierten Basis gilt:

$$\|\underline{\tau}_j\| \leq n^{\frac{1-n}{2}} 2^{\frac{(n+2)(n-1)}{4} - r_2} \sqrt{|\Delta_{\mathcal{O}}|} \left[1 + \mathrm{dft}(\underline{\Omega}) \left(\left(1 + 2^{-q} \frac{\sqrt{n}}{\lambda_1(\underline{\Omega})} \right)^n - 1 \right) \right] \left[1 + 2^{-q} n^2 \frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})} \right]^n .$$

Beweis:
Es gilt:

$$\|\underline{\tau}_j\| \leq \frac{\prod_{i=1}^{n} \|\underline{\tau}_i\|}{(\min_{i=1,\ldots,n} \|\underline{\tau}_i\|)^{n-1}} \;.$$

Daraus ergibt sich mit (5.1) und Lemma 3.4:

$$\begin{aligned}\|\underline{\tau}_j\| \leq \; & \sqrt{\tfrac{2}{n}}^{\,n-1} 2^{\frac{n(n-1)}{4}} |\det(\underline{\Omega})| \; \left[1+ \mathrm{dft}(\underline{\Omega}) \left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n - 1\right)\right] \\ & \left[1+ 2^{-q} n^2 \frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n \;.\end{aligned}$$

Also ergibt sich mit Lemma 5.4 die Behauptung. □

Nun können wir den folgenden Satz beweisen:

5.6. Satz
Für die Einträge w'_{ijk} in der Multiplikationstafel von ΩT gilt:

$$\begin{aligned}|w'_{ijk}| \leq \; & n^{\frac{1}{2}-n} 2^{\frac{(3n+4)(n-1)+2}{4} - 2r_2} |\Delta_{\mathcal{O}}| \; \left[1+ \mathrm{dft}(\underline{\Omega}) \left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n - 1\right)\right]^3 \\ & \left[1+ 2^{-q} n^2 \frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^{3n} \;. \qquad (5.2)\end{aligned}$$

Beweis:
Nach Definition der Multiplikationstafel gilt für Konjugiertenvektoren $\underline{\tau}_i$, $\underline{\tau}_j$ $(1 \leq i,j \leq n)$ der reduzierten Basis:

$$\underline{\tau}_i \underline{\tau}_j = \sum_{k=1}^{n} w'_{ijk} \underline{\tau}_k,$$

oder anders formuliert:

$$\underline{\tau}_i \underline{\tau}_j = (\underline{\Omega} T)(w'_{ij1}, \ldots, w'_{ijn})^T.$$

Nach der Kramerschen Regel ergibt sich also analog zu Gleichung (1.2):

$$|w'_{ijk}| \leq \frac{1}{|\det(\underline{\Omega} T)|} \prod_{l \neq k} \|\underline{\tau}_l\| \|\underline{\tau}_i \underline{\tau}_j\|.$$

Mit (5.1) und wegen $|\det T| = 1$ folgt also:

$$\begin{aligned}|w'_{ijk}| \;\; \leq \;\; & \frac{1}{|\det(\underline{\Omega})| \min_{1 \leq l \leq n} \|\underline{\tau}_l\|} 2^{\frac{n(n-1)}{4}} |\det(\underline{\Omega})| \\ & \left[1+ \mathrm{dft}(\underline{\Omega}) \left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n - 1\right)\right] \left[1+ 2^{-q} n^2 \frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n \|\underline{\tau}_i \underline{\tau}_j\|\end{aligned}$$

$$= \frac{1}{\min\limits_{1\le l\le n}\|\underline{\mathcal{I}_l}\|} 2^{\frac{n(n-1)}{4}} \|\underline{\mathcal{I}_i}\|\|\underline{\mathcal{I}_j}\|$$
$$\left[1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\right]\left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n .$$

Nach Lemma 3.4 und Lemma 5.5 ergibt sich:

$$\begin{aligned}
|w'_{ijk}| \;\le\;& \sqrt{\frac{2}{n}}2^{\frac{n(n-1)}{4}}\left[1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\right]\left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n\\
&\left(n^{\frac{1-n}{2}}2^{\frac{(n+2)(n-1)}{4}-r_2}\sqrt{|\Delta_{\mathcal{O}}|}\right)^2\\
&\left(\left[1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\right]\left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n\right)^2\\
=\;& n^{\frac{1}{2}-n}2^{\frac{(3n+4)(n-1)+2}{4}-2r_2}|\Delta_{\mathcal{O}}|\\
&\left[1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\right]^3\left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^{3n} .
\end{aligned}$$

□

Nun müssen wir diese kurze Multiplikationstafel noch konkret berechnen. Um Satz 4.16 anwenden zu können, müssen wir noch eine Abschätzung für $|T|$ finden. Nach Satz 3.20 erhalten wir $|T| = O(2^{n\max(1,q+\log\lambda_n(\underline{\Omega}))})$. Da nun nach Lemma 3.4 $\log\lambda_n(\underline{\Omega}) > 0$ für $n > 2$ gilt, erhalten wir also: $s(T) = O(|\operatorname{MT}(\Omega)| + 2^{n(q+\log\lambda_n(\underline{\Omega}))})$.

Nach Satz 4.16 können wir die neue, kurze Multiplikationstafel also in Zeit

$$\begin{aligned}
&O(n^5\log^2(n|\operatorname{MT}(\Omega)| + n2^{n(q+\log\lambda_n(\underline{\Omega}))}))\\
&\quad= O(n^5(\log n + \log(|\operatorname{MT}(\Omega)| + 2^{n(q+\log\lambda_n(\underline{\Omega}))}))^2)\\
&\quad= O(n^5(n(q+\log\lambda_n(\underline{\Omega})) + \log(n|\operatorname{MT}(\Omega)|))^2)\\
&\quad= O(n^7(q+\log\lambda_n(\underline{\Omega}))^2 + n^5\log^2(n|\operatorname{MT}(\Omega)|))
\end{aligned}$$

berechnen. Dabei gilt die letzte Gleichheit, da stets einer der Terme $n(q+\log\lambda_n(\underline{\Omega}))$ und $\log(n|\operatorname{MT}(\Omega)|)$ das Maximum dieser beiden Werte annimmt und daher der mittlere Summand, der gemäß der binomischen Formel auftreten würde, verschwindet.

Um nun den gesamten Zeitaufwand für diesen Schritt abzuschätzen, zeigen wir folgende Ungleichungen, die auch für Abschätzungen in den folgenden Abschnitten relevant sind:

5.7. Satz

Es gilt:

$$\lambda_1(\underline{\Omega}) \;\ge\; \sqrt{\frac{n}{2}} \tag{5.3}$$

$$\lambda_n(\underline{\Omega}) \leq n^{\frac{5}{2}} |\operatorname{MT}(\Omega)| \tag{5.4}$$

$$\operatorname{dft}(\underline{\Omega}) \leq \frac{(\sqrt{2n}n^2 |\operatorname{MT}(\Omega)|)^n}{\sqrt{|\Delta_{\mathcal{O}}|}} \tag{5.5}$$

Beweis:
(5.3) ergibt sich aus Lemma 3.4.

Nun beweisen wir (5.4): Es ist $\lambda_n(\underline{\Omega}) \leq \sqrt{n}|\underline{\Omega}|$. Nach Konstruktion sind die Einträge in $\underline{\omega_i}$ für $1 \leq i \leq n$ Real– bzw. Imaginärteile von Nullstellen des Minimalpolynoms von ω_i. Nach Satz 4.2, Abschätzung (4.3), ergibt sich, daß die Nullstellen all dieser Minimalpolynome betragsmäßig durch $n^2|\operatorname{MT}(\Omega)|$ beschränkt sind, d.h. also:

$$|\underline{\Omega}| \leq n^2 |\operatorname{MT}(\Omega)|. \tag{5.6}$$

Damit ergibt sich (5.4).

Wegen $\operatorname{dft}(\underline{\Omega}) = \frac{1}{|\det \underline{\Omega}|} \prod_{i=1}^n \underline{\omega_i}$ gilt nun mit (5.4):

$$\operatorname{dft}(\underline{\Omega}) \leq \frac{1}{|\det \underline{\Omega}|} (\sqrt{n} n^2 |\operatorname{MT}(\Omega)|)^n$$

und mit Lemma 5.4 folgt also:

$$\operatorname{dft}(\underline{\Omega}) \leq \frac{1}{\sqrt{|\Delta_{\mathcal{O}}|}} 2^{r_2} (\sqrt{n} n^2 |\operatorname{MT}(\Omega)|)^n.$$

Wegen $n = r_1 + 2r_2$ ist $r_2 \leq \frac{n}{2}$, und so ergibt sich die Behauptung. □

Zum Schluß setzen wir die Teilergebnisse dieses Abschnitts zu folgendem Satz zusammen:

5.8. Satz
Sei eine Ordnung $\mathcal{O}$ durch ihre Multiplikationstafel bezüglich der Basis Ω gegeben. Ferner sei eine Zahl $q \in \mathbb{N}$ mit $q \geq 2\log n + \log \operatorname{dft}(\underline{\Omega}) - \log \lambda_1(\underline{\Omega})$ gegeben. Dann benötigen wir Zeit

$$O(n^6(q + \log(n|\operatorname{MT}(\Omega)|))^3 + n^7(q + \log(n|\operatorname{MT}(\Omega)|))^2),$$

um eine Multiplikationstabelle zu berechnen, die (5.2) erfüllt.

Beweis:
Indem wir die Laufzeiten der Teilschritte aufaddieren, erhalten wir wegen $\log \lambda_n(\underline{\Omega}) > 0$ für diese Transformation eine Gesamtlaufzeit von

$$O(\underbrace{(n^3 q + n^5 \log(n|\operatorname{MT}(\Omega)|)\log(n(q+\log|\operatorname{MT}(\Omega)|))\log\log(n(q+\log|\operatorname{MT}(\Omega)|))}_{\text{Berechnung der Nullstellen des Minimalpolynoms des primitiven Elements}}$$
$$+ \underbrace{n^7\log^2(n|\operatorname{MT}(\Omega)|)}_{\text{primitives Element bestimmen, Transformationen}} + \underbrace{n^6(q+\log\lambda_n(\underline{\Omega}))^3}_{\textbf{LLL}\text{–Reduktion}}$$
$$+ \underbrace{n^7(q+\log\lambda_n(\underline{\Omega}))^2 + n^5\log^2(n|\operatorname{MT}(\Omega)|)}_{\text{Transformation der Multiplikationstafel}}).$$

Aufgrund von (5.4) ergibt sich $\log \lambda_n(\underline{\Omega}) = O(\log(n|\operatorname{MT}(\Omega)|))$ und somit vereinfacht sich die Laufzeit zu

$$O(n^6(q+\log(n|\operatorname{MT}(\Omega)|))^3 + n^7(q+\log(n|\operatorname{MT}(\Omega)|))^2). \qquad \square$$

5.9. Bemerkung
Die kritischen Schritte, die die Laufzeit dominieren, sind hier also einerseits die **LLL**–Reduktion, wobei die Laufzeit hierfür durch schnelle Multiplikationstechniken asymptotisch noch gesenkt werden kann, und andererseits die Transformation der Multiplikationstafel, wobei ein besseres Ergebnis hierfür wohl nur durch eine bessere Abschätzung der in Algorithmus 3.13 berechneten Transformationsmatrix möglich wäre.

Benutzt man nun zur **LLL**–Reduktion schnelle Multiplikationsalgorithmen, dann kann man die Laufzeit auf $O(n^7(q+\log(n|\operatorname{MT}(\Omega)|))^{2+\epsilon})$, bzw. genauer auf

$$O(n^{5+\epsilon}(q+\log(n|\operatorname{MT}(\Omega)|))^{2+\epsilon} + n^7(q+\log(n|\operatorname{MT}(\Omega)|))^2)$$

senken, vgl. Bemerkung 3.21.

5.3 Hinzufügen von 1 zur Basis

Nun können wir, wie bereits angekündigt, in einem zweiten Schritt die Basis so modifizieren, daß die 1 als erstes Basiselement auftritt. Nach Satz 4.15 benötigen wir Zeit

$$O(n^4\log^2(n|\operatorname{MT}(\Omega T)|)),$$

um eine Transformationsmatrix X mit

$$|X| \leq (\sqrt{n}|\,\mathrm{MT}(\Omega T)|)^{3n-3} \tag{5.7}$$

zu berechnen, so daß ΩTX die 1 enthält. Da der Sinn der Transformation T darin bestand, die Einträge der Multiplikationstafel zu verkleinern, setzen wir nun voraus:

$$|\,\mathrm{MT}(\Omega T)| \leq |\,\mathrm{MT}(\Omega)|.$$

Dies ist o.B.d.A. möglich, da wir, falls dies nicht zutrifft, statt mit der „reduzierten Basis" mit der ursprünglichen Eingabe rechnen können, die dann bereits kurz genug gewesen sein muß. Dieses Problem kann einerseits auftreten, wenn die Eingabe so gewählt war, daß die Einträge in der Multiplikationstafel besonders kurz waren, obwohl die Basisvektoren lang waren, oder aber andererseits, wenn die Eingabebasis bereits sehr kurz war und die in Bemerkung 3.19 angedeutete Schwierigkeit der durch Rundungsfehler bedingten falschen Reduktion eingetreten ist. Damit erhalten wir also zur Berechnung der Transformation einen Zeitaufwand von

$$O(n^4 \log^2(n|\,\mathrm{MT}(\Omega)|))$$

und wir erhalten Einträge in X mit

$$|X| \leq (\sqrt{n}|\,\mathrm{MT}(\Omega)|)^{3n-3}.$$

Nun müssen wir die Multiplikationstafel, die wir im vorangehenden Schritt erhielten, mit X transformieren und erhalten nach Satz 4.16 in Zeit

$$O(n^5 \log^2(n(\sqrt{n}|\,\mathrm{MT}(\Omega)|)^{3n-3})) = O(n^7 \log^2(n|\,\mathrm{MT}(\Omega)|)) \tag{5.8}$$

eine Multiplikationstafel mit

$$\begin{aligned} |w'_{ijk}| &\leq n^{\frac{n+4}{2}}|X|^{n+1}|\,\mathrm{MT}(\Omega T)| \\ &\leq n^{\frac{n+4}{2}}(\sqrt{n}|\,\mathrm{MT}(\Omega T)|)^{3n^2-3}|\,\mathrm{MT}(\Omega T)| \quad \text{wegen (5.7)} \\ &= n^{\frac{3n^2+n+1}{2}}|\,\mathrm{MT}(\Omega T)|^{3n^2-2}. \end{aligned}$$

Wenn wir nun für $|\,\mathrm{MT}(\Omega T)|$ die durch die Reduktion gesicherte Schranke einsetzen, ergibt sich somit:

$$\begin{aligned} |w'_{ijk}| \leq\; & n^{-3n^3+3n^2+\frac{5n-1}{2}} 2^{\left(\frac{(3n+4)(n-1)+2}{4}-2r_2\right)(3n^2-2)} |\Delta_{\mathcal{O}}|^{3n^2-2} \\ & \left[1+\mathrm{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\right]^{9n^2-6} \left[1+2^{-q}n^2\frac{\mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^{9n^3-6n} . \end{aligned} \tag{5.9}$$

5.10. Bemerkung

Eine praktische Implementierung könnte die interessante Frage untersuchen, ob es sich lohnt, zunächst die Basis so zu transformieren, daß 1 als erstes Basiselement auftritt, und erst anschließend die Reduktion auszuführen. Auf diese Weise könnte man eine wesentlich kleinere Multiplikationstafel erhalten. Wenn man Glück hat, ist in der Basis, zu der man die Multiplikationstafel erhält, immer noch die 1 enthalten, obwohl die Reduktion dies keineswegs garantiert.

Falls jedoch die Ordnung z.B. so geartet ist, daß es eine Basis gibt, so daß stets alle Konjugierten eines Basiselements den gleichen Betrag haben, dann kann man zeigen, daß der Vektor, der die 1 darstellt, ein kürzester Vektor unter den Konjugiertenvektoren der Ordnung ist. Nach Lemma 3.4 ist dies auch der Fall, wenn das Minimalpolynom des primitiven Elements keine reellen Nullstellen hat, da dann die 1 gerade durch einen Vektor der Länge $\sqrt{\frac{n}{2}}$ dargestellt wird. Da nun der **LLL**–Algorithmus garantiert keinen Vektor verlängert, wird also die 1 in der Basis erhalten bleiben, wenn man mit hinreichender Genauigkeit rechnet. Allerdings tritt bei zu geringer Genauigkeit eventuell das Problem aus Bemerkung 3.19 auf.

Aber auch wenn dieser besonders günstige Fall nicht eintritt, gilt für den Vektor $\underline{1}$, der die 1 darstellt, immerhin $\|\underline{1}\| = \sqrt{r_1 + r_2}$ und somit $\sqrt{\frac{n}{2}} \leq \|\underline{1}\| \leq \sqrt{n}$. Aufgrund von Lemma 3.4 ist also $\underline{1}$ höchstens um einen Faktor $\sqrt{2}$ länger als der kürzeste Vektor. Da der **LLL**-Algorithmus aber (bei exakter Rechnung) selbst für den kürzesten berechneten Vektor nur $\|\underline{\tilde{b}}_1\| \leq \sqrt{2}^{n-1}\lambda_1(L)$ garantiert, sollte eine nicht allzu geringe Wahrscheinlichkeit bestehen, daß der Algorithmus die 1 nicht aus der Basis entfernt.

Für diese möglicherweise verbesserte Qualität des Ergebnisses muß man allerdings einen hohen Preis an Rechenzeit zahlen: Unter Umständen hat nämlich die **LLL**–Reduktion doch den Vektor $\underline{1}$ zerstört, dann kann man, wenn man unbedingt eine besonders kurze Basis mit 1 erhalten will, nochmals die **LLL**–Reduktion mit größerer Genauigkeit ausführen. Allerdings ist nicht unbedingt gesichert, daß die 1 bei exakter Rechnung wirklich erhalten bliebe, also terminiert dieses Verfahren gar nicht. Selbst wenn wir aber durch eine genauere Untersuchung der vorliegenden Ordnung wissen, daß exakte Rechnung die 1 in der Basis erhielte, können wir keine Schranke für die erforderliche Rechengenauigkeit angeben, mit der man rechnen muß, um den Konjugiertenvektor der 1 im Ergebnis wiederzufinden.

5.4 Zusammenfassung

Zum Schluß wollen wir noch untersuchen, mit welcher Genauigkeit man rechnen muß, um ein Resultat vorgegebener Güte zu erreichen, und wie hoch der Zeitaufwand dafür ist. Zunächst wollen wir aber sehen, welches Resultat wir erhalten, wenn wir mit der minimalen Genauigkeit rechnen, die erforderlich ist, damit wir die Korrektheit des Algorithmus garantieren können:

5.11. Satz
Für $q = \left\lceil \left(\frac{3}{2}+\frac{5}{2}n\right)\log n + n\log|\,\mathrm{MT}(\Omega)| - \frac{1}{2}\log(|\Delta_{\mathcal{O}}|) + \frac{n+1}{2}\right\rceil$ *erhalten wir aus einer gegebenen Multiplikationstafel* $\mathrm{MT}(\Omega)$ *in Zeit*

$$O(n^9\log^3(n|\,\mathrm{MT}(\Omega)|))$$

eine Multiplikationstafel $\mathrm{MT}(\Omega')$ *mit*

$$|\,\mathrm{MT}(\Omega')| < n^{\frac{1}{2}-n}2^{\frac{(3n+4)(n-1)+2}{4}-2r_2+3n+6}|\Delta_{\mathcal{O}}|.$$

Außerdem können wir in derselben Zeit eine Multiplikationstafel $\mathrm{MT}(\Omega'')$ *mit*

$$|\,\mathrm{MT}(\Omega'')| < n^{-3n^3+3n^2+\frac{5n-1}{2}}2^{\left(\frac{(3n+4)(n-1)+2}{4}-2r_2\right)(3n^2-2)+9n^3+18n^2-6n-12}|\Delta_{\mathcal{O}}|^{3n^2-2}$$

erhalten, so daß die erste Zahl in der Basis Ω'' *die Zahl* 1 *ist.*

Beweis:
Mit dem angegebenen Wert für q ergibt zunächst aus (5.3) und (5.5):

$$q \geq 2\log n + \log \mathrm{dft}(\underline{\Omega}) - \log\lambda_1(\underline{\Omega}).$$

Damit wissen wir zunächst, daß wir garantieren können, daß der modifizierte **LLL**–Algorithmus funktioniert, und wir erhalten für $\mathrm{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)$ folgende Abschätzung:

$$\begin{aligned}
\mathrm{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right) &\leq \mathrm{dft}(\underline{\Omega})\left(\left(1+\frac{1}{n\sqrt{n}\,\mathrm{dft}(\underline{\Omega})}\right)^n-1\right)\\
&= \mathrm{dft}(\underline{\Omega})\sum_{i=1}^{n}\binom{n}{i}\frac{1}{n^{\frac{3}{2}i}(\mathrm{dft}(\underline{\Omega}))^i}\\
&= \sum_{i=1}^{n}\binom{n}{i}\frac{1}{n^{\frac{3}{2}i}\underbrace{(\mathrm{dft}(\underline{\Omega}))^{i-1}}_{\geq 1}}
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{i=1}^{n} \frac{n(n-1)\cdots(n-i+1)}{i!}\frac{1}{n^{\frac{3}{2}i}} \\
&\leq \sum_{i=1}^{\infty} \frac{1}{n^{\frac{i}{2}}} = \frac{1}{1-\frac{1}{\sqrt{n}}} - 1 \quad .
\end{aligned}$$

Da $n \geq 2$ und also $\frac{1}{\sqrt{n}} < \frac{3}{4}$ erhalten wir:

$$1 + \operatorname{dft}(\underline{\Omega})\left(\left(1+\frac{1}{n\sqrt{n}+\operatorname{dft}(\underline{\Omega})}\right)^{n} - 1\right) < 4.$$

Damit erhalten wir also aus Satz 5.6:

$$\begin{aligned}
|\operatorname{MT}(\Omega')| &< n^{\frac{1}{2}-n} 2^{\frac{(3n+4)(n-1)+2}{4}-2r_2} |\Delta_{\mathcal{O}}| 4^3 2^{3n} \\
&= n^{\frac{1}{2}-n} 2^{\frac{(3n+4)(n-1)+2}{4}-2r_2+3n+6} |\Delta_{\mathcal{O}}|.
\end{aligned}$$

Also erhalten wir gemäß (5.9), wenn wir die 1 als (ersten) Basisvektor garantieren wollen:

$$\begin{aligned}
|\operatorname{MT}(\Omega'')| &< n^{-3n^3+3n^2+\frac{5n-1}{2}} 2^{\left(\frac{(3n+4)(n-1)+2}{4}-2r_2\right)(3n^2-2)} |\Delta_{\mathcal{O}}|^{3n^2-2} 4^{9n^2-6} 2^{9n^3-6n} \\
&= n^{-3n^3+3n^2+\frac{5n-1}{2}} 2^{\left(\frac{(3n+4)(n-1)+2}{4}-2r_2\right)(3n^2-2)+9n^3+18n^2-6n-12} |\Delta_{\mathcal{O}}|^{3n^2-2}
\end{aligned}$$

Nun zur Laufzeit: Wenn wir q wie im Satz angegeben wählen, so gilt wegen $|\Delta_{\mathcal{O}}| \in \mathbb{N}$ auch $\log |\Delta_{\mathcal{O}}| \geq 0$ und es ergibt sich: $q \in O(n \log(n|\operatorname{MT}(\Omega)|))$. Daher ist nach Satz 5.8 die Zeitkomplexität der Reduktion der ursprünglichen Tafel

$$O(n^6(n\log(n|\operatorname{MT}(\Omega)|))^3 + n^7(n\log(n|\operatorname{MT}(\Omega)|))^2)$$

Damit erhalten wir die Behauptung für die Laufzeit der Reduktion.

Für die anschließende Transformation, so daß die Basis die 1 enthält, benötigen wir dann nach (5.8) Zeit $O(n^7\log^2(n|\operatorname{MT}(\Omega)|))$. Dies erhöht also die zuvor berechnete Laufzeit nicht mehr weiter. □

5.12. Bemerkung

Durch Verwendung schneller Multiplikationstechniken bei der **LLL**–Reduktion kann man nach Bemerkung 5.9 die Laufzeit noch auf $O(n^9\log^{2+\epsilon}(n|\operatorname{MT}(\Omega)|))$ senken.

Will man nun dichter an das bestmögliche Ergebnis herankommen, so muß man mehr Zeit investieren. Zunächst vergröbern wir die Abschätzung für die Qualität des Ergebnisses, damit wir leichter eine Abschätzung für die benötigte Rechengenauigkeit finden können. Dann formulieren wir eine Bedingung an q, die erzwingt, daß der Faktor, um den man vom bestmöglichen Ergebnis entfernt ist, trotz der Vergröberung beliebig nahe an 1 liegt.

5.13. Proposition
Es gilt:

$$1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right)\leq\left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n .$$

Beweis:
Es ist

$$\begin{aligned}1+\operatorname{dft}(\underline{\Omega})\left(\left(1+2^{-q}\frac{\sqrt{n}}{\lambda_1(\underline{\Omega})}\right)^n-1\right) &= 1+\underbrace{\operatorname{dft}(\underline{\Omega})}_{\geq 1}\sum_{i=1}^n\binom{n}{i}2^{-iq}\frac{\sqrt{n}^i}{\lambda_1(\underline{\Omega})^i}\\ &\leq 1+\sum_{i=1}^n\binom{n}{i}2^{-iq}\frac{n^{2i}\operatorname{dft}(\underline{\Omega})^i}{\lambda_1(\underline{\Omega})^i}\\ &= \sum_{i=0}^n\binom{n}{i}2^{-iq}\frac{n^{2i}\operatorname{dft}(\underline{\Omega})^i}{\lambda_1(\underline{\Omega})^i}\\ &= \left[1+2^{-q}n^2\frac{\operatorname{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right]^n .\end{aligned}$$

□

Nun beweisen wir:

5.14. Satz
Sei $c>1$ gegeben. Für

$$\begin{aligned}q &= \max\Bigg(\left\lceil\frac{1}{2}(1+n+(3+n)\log n-\log|\Delta_{\mathcal{O}}|)+n\log|\operatorname{MT}(\Omega)|-\log(\sqrt[6n]{c}-1)\right\rceil,\\ &\qquad\left\lceil(\frac{3}{2}+\frac{5}{2}n)\log n+n\log|\operatorname{MT}(\Omega)|-\frac{1}{2}\log(|\Delta_{\mathcal{O}}|)+\frac{n+1}{2}\right\rceil\Bigg)\end{aligned}$$

erhalten wir aus einer gegebenen Multiplikationstafel $\operatorname{MT}(\Omega)$ *in Zeit*

$$O(n^9\log^3(n|\operatorname{MT}(\Omega)|)-n^6\log^3(\sqrt[6n]{c}-1))$$

eine Multiplikationstafel $\operatorname{MT}(\Omega')$ *mit*

$$|\operatorname{MT}(\Omega')|\leq n^{\frac{1}{2}-n}2^{\frac{(3n+4)(n-1)+2}{4}-2r_2}|\Delta_{\mathcal{O}}|c.$$

Außerdem erhalten wir mit

$$\begin{aligned}q &= \max\Bigg(\Bigg\lceil\frac{1}{2}(1+n+(3+n)\log n-\log|\Delta_{\mathcal{O}}|)+n\log|\operatorname{MT}(\Omega)|\\ &\qquad -\log\left(c^{\frac{1}{18(n^3+2n^2)}}-1\right)\Bigg\rceil,\\ &\qquad\left\lceil(\frac{3}{2}+\frac{5}{2}n)\log n+n\log|\operatorname{MT}(\Omega)|-\frac{1}{2}\log(|\Delta_{\mathcal{O}}|)+\frac{n+1}{2}\right\rceil\Bigg)\end{aligned}$$

in Zeit

$$O(n^9 \log^3(n|\,\mathrm{MT}(\Omega)|) - n^6 \log^3(c^{\frac{1}{18(n^3+2n^2)}} - 1))$$

eine Multiplikationstafel $\mathrm{MT}(\Omega'')$ *mit*

$$|\,\mathrm{MT}(\Omega'')| \leq n^{-3n^3+3n^2+\frac{5n-1}{2}} 2^{\left(\frac{(3n+4)(n-1)+2}{4} - 2r_2\right)(3n^2-2)} |\Delta_{\mathcal{O}}|^{3n^2-2} c,$$

so daß die erste Zahl in der Basis Ω'' *die Zahl* 1 *ist.*

Beweis:

Aufgrund der zweiten Komponente des Maximums ist sichergestellt, daß der Algorithmus funktioniert, wie wir im vorherigen Satz gesehen haben. Mit (5.3) wissen wir, daß gilt: $\log \lambda_1(\underline{\Omega}) \geq \frac{1}{2}(\log n - 1)$, aus (5.5) ergibt sich $\log \mathrm{dft}(\underline{\Omega}) \leq n(\log |\,\mathrm{MT}(\Omega)| + \frac{5}{2}\log n + \frac{1}{2}) - \frac{1}{2}\log |\Delta_{\mathcal{O}}|$. Aufgrund unserer Wahl von q (1. Komponente des Maximums) gilt somit:

$$q \geq 2\log n + \log \mathrm{dft}(\underline{\Omega}) - \log \lambda_1(\underline{\Omega}) - \log(\sqrt[6n]{c} - 1).$$

Daraus ergibt sich nach einigen einfachen Umformungen:

$$c \geq \left(1 + 2^{-q} \frac{n^2\, \mathrm{dft}(\underline{\Omega})}{\lambda_1(\underline{\Omega})}\right)^{6n},$$

und somit erhalten wir nach Proposition 5.13 und Satz 5.6 die Größenabschätzung für $\mathrm{MT}(\Omega')$.

Nun zum Zeitaufwand: Wenn wir q wie angegeben wählen, ergibt sich (wiederum wegen $|\Delta_{\mathcal{O}}| \in \mathbb{N}$) $q = O(n \log(n|\,\mathrm{MT}(\Omega)|) + \log \frac{1}{\sqrt[6n]{c}-1})$. Mit Satz 5.8 erhalten wir also als Laufzeit:

$$\begin{aligned} &O(n^6(n\log(n|\,\mathrm{MT}(\Omega)|) + \log \frac{1}{\sqrt[6n]{c}-1})^3 + n^7(n\log(n|\,\mathrm{MT}(\Omega)|) + \log \frac{1}{\sqrt[6n]{c}-1})^2) \\ &= O(n^9 \log^3(n|\,\mathrm{MT}(\Omega)|) - n^6 \log^3(\sqrt[6n]{c} - 1)), \end{aligned}$$

da stets (mindestens) einer der beiden Terme $n\log(n|\,\mathrm{MT}(\Omega)|)$ und $\log(\sqrt[6n]{c} - 1)$ das Maximum dieser beiden Werte annimmt und daher die restlichen Summanden, die man nach dem Binomischen Satz aus der Summe erhält, wegfallen.

Nun betrachten wir die zweite Aussage. Wiederum sichert die zweite Komponente des Maximums das Funktionieren des Algorithmus. Aus der ersten Komponenten ergibt sich wiederum durch einige Umformungen gemäß (5.3) und (5.5) unter Verwendung von Proposition 5.13 und (5.9) die Größenabschätzung für $\mathrm{MT}(\Omega'')$.

In der Laufzeitabschätzung müssen wir nun $q = O(n \log(n|\,\mathrm{MT}(\Omega)|) - \log(c^{\frac{1}{18(n^3+2n^2)}} - 1))$ einsetzen, und wir erhalten wie im ersten Teil des Beweises die behauptete Laufzeit, da q ausreichend groß ist, so daß die Zeit für die Transformation, die die 1 zur Basis hinzufügt, bereits enthalten ist. $\square$

5.15. Bemerkung

Ebenso wie im vorangehenden Satz sieht man wieder, daß man durch Verwendung schneller Multiplikationstechniken bei der **LLL**–Reduktion die Laufzeit nur wenig senken kann; man erhält eine Laufzeit von $O(n^9 \log^{2+\epsilon}(n|\,\mathrm{MT}(\Omega)|) - n^7 \log^{2+\epsilon}(\sqrt[6n]{c} - 1))$.

Nun wollen wir nochmals die in Bemerkung 5.10 angesprochene Modifikation betrachten, wobei wir nun voraussetzen wollen, daß wir (zufällig) die Vertauschung der beiden Teilschritte erfolgreich anwenden konnten, d.h. der Vektor $\underline{1}$ bleibt tatsächlich in der Basis erhalten. Während wir aber in der vorgenommen Laufzeitanalyse annehmen konnten, daß durch die Reduktion die Eingabe zumindest nicht verlängert wird und also das Hinzufügen von 1 auch nicht allzulange dauert, vergrößert das Hinzufügen der 1 die Multiplikationstafel eventuell erheblich, genauer gesagt wird nach Satz 4.15 und Satz 4.16 ein Eintrag bis zu n^2–mal so viele Bits wie zuvor haben. Daher benötigt die Reduktion nun sehr viel mehr Zeit, genauer gesagt muß man wohl den Faktor n^9, den wir in der Laufzeit gesehen haben, durch einen Faktor n^{15} ersetzen. Wenn man schnelle Multiplikationsalgorithmen für die **LLL**–Reduktion verwendet, fällt die Steigerung allerdings weniger dramatisch aus, dann reicht „schon" ein Faktor n^{13} anstelle des Faktors n^9.

5.16. Bemerkung

Ein realer Algorithmus sollte zur Berechnung natürlich anders vorgehen. Statt die schlechten a–priori–Abschätzungen für q zu verwenden, sollte ein Algorithmus die Konjugiertenvektoren zunächst mit geringer Genauigkeit ausrechnen, dann den Defekt bestimmen und anhand dieses Wertes die für die angestrebte Qualität des Ergebnisses notwendige Präzision abschätzen und bei Bedarf nachiterieren, um die Konjugiertenvektoren zu verbessern. Da Determinante und Betrag in ihren Argumenten stetig sind und somit insbesondere auch der Defekt, ist nicht zu erwarten, daß sich durch die größere Präzision der Konjugiertenvektoren die Abschätzung für die benötigte Präzision drastisch ändert. Auf diese Weise kann man den durch Gleichung (5.5) bedingten pessimistischen Faktor n in der Abschätzung für q in der Realität wahrscheinlich oft eliminieren und so auch für den ersten Schritt eine Laufzeit von $O(n^7 \cdots)$ erhalten. Damit haben die Reduktion und das Hinzufügen von 1 zur Basis beide in etwa den gleichen Zeitbedarf. Benutzt man dann für die **LLL**–Reduktion insbesondere auch noch schnelle Multiplikationsalgorithmen, so kann man unter günstigen Umständen Laufzeiten von $O(n^7 \log^{2+\epsilon}(n|\,\mathrm{MT}(\Omega)|))$ erreichen. Entsprechende Verbesserungen ergeben sich natürlich auch für die soeben angesprochene Vertauschung der beiden Schritte.

Literaturverzeichnis

[Ba05] P. Bachmann, *Zahlentheorie*, Fünfter Teil, Teubner, 1905.

[Br71] W. S. Brown, *On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors*, Journal of the Association for Computing Machinery 18, 1971, S. 478 – 504.

[BuKe90] J. Buchmann, V. Kessler, *Computing a short lattice basis from a generating system with applications*, Vorabdruck.

[BuSp92] J. Buchmann, O. van Sprang, *On short representations of orders and number fields*, Vorabdruck.

[Bu92] J. Buchmann, *Einführung in die Computeralgebra*, Skript zur Vorlesung im WS 91/92 an der Universität des Saarlandes.

[Co93] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer–Verlag, 1993.

[CoLeRi90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[Go73] L. J. Goldstein, *Abstract Algebra*, Prentice–Hall, 1973.

[Jä91] K. Jänich, *Lineare Algebra*, 4. Auflage, Springer–Verlag, 1991.

[LLL82] A. K. Lenstra, H. W. Lenstra Jr. , L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. 261, 1982, S. 515 – 534.

[LePa81] H. R. Lewis, C. H. Papadimitriou, *Elements of the theory of computation*, Prentice–Hall, 1981.

[LoMeWi86] J. Loeckx, K. Mehlhorn, R. Wilhelm, *Grundlagen der Programmiersprachen*, B. G. Teubner–Verlag, 1986.

[Me84] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*, Springer–Verlag, 1984

[Mi92] M. Mignotte, *Mathematics for Computer Algebra*, Springer–Verlag, 1992.

[Mü94] A. Müller, *Effiziente Algorithmen für Probleme der linearen Algebra über* $\mathbb{Z}$, Diplomarbeit an der Universität des Saarlandes, 1994.

[NeRo92] S. Neis, U. Roglin, *Parallele modulare Berechnungen für Matrizen*, Fortgeschrittenenpraktikum an der Universität des Saarlandes im SS 92.

[Ne92] J. Neukirch, *Algebraische Zahlentheorie*, Springer–Verlag, 1992.

[RoSc85] J. B. Rosser, L. Schoenfeld, *Approximate formulas for some functions of prime numbers*, Illinois Journal of Mathematics, Vol. 6, 1985.

[Sch82] A. Schönhage, *The fundamental theorem of algebra in terms of computational complexity*, Preliminary report, Universität Tübingen, 1982.

[StBu78] J. Stoer, R. Bulirsch *Einführung in die Numerische Mathematik II*, 2. Auflage, Springer–Verlag, 1978.

Stichwortverzeichnis