数体ふるい法における多項式選択部分・線形代数部分・イデアル平方根計算部分についての調査研究

２００４年１月

株式会社富士通研究所

下山 武司

伊豆 哲也

数体ふるい法における多項式選択部分・線形代数部分・イデアル平方根計算部分についての調査研究

富士通株式会社

2004年1月

本報告書について

整数の素因数分解問題 (Integer Factoring Problem) とは, 与えられた整数 n の自明でない因数を求める問題である. この問題は数論の分野で古くから知られていた問題であるが, 効率的な解法, つまり n のサイズ (ビット長) の多項式時間で解けるアルゴリズムは現在のところ知られていない. このような素因数分解問題の難しさはいくつかの公開鍵暗号・電子署名などに利用され, 暗号解読の難しさの根拠となっている. 従って素因数分解問題の理論的な解法に関する研究はもちろんのこと, どれだけのサイズの素因数分解がどれだけのリソースで素因数分解可能であるかを現実的に把握することは, セキュリティ保護の観点か重要な意味を持つ.

数体ふるい法 (NFS; Number Field Sieve method) [13] は素因数分解問題に対する現時点での最良アルゴリズムであり, その計算時間は n のサイズの準指数関数で表される. ここで数体ふるい法の (理論的な) 計算時間を T^{NFS} とすると, 漸近的には (つまり $n \to \infty$ のときには)

$$T^{\mathrm{NFS}} = L_n\left[\frac{1}{3}, 1.526\right]$$

となることが知られている [14]. ただし関数 $L_n[u, v]$ は

$$L_n[u, v] = \exp\left((v + o(1))(\log\log n)^u (\log n)^{1-u}\right)$$

で定義される関数で, $0 < u < 1$ のときに n の準指数関数となる. また実際の素因数分解記録においても数体ふるい法は優れており, 現時点 (2004 年 1 月) での記録である 576-bit 合成数の素因数分解にも数体ふるい法が使用されている [8].

数体ふるい法は多項式選択ステップ, 関係式探索 (ふるい) ステップ, 行列計算 (線型代数計算) ステップ, イデアル平方根計算ステップ, 最終計算 ($\boldsymbol{n}$ の分解) ステップの 5 つのステップから構成される. 従来の数体ふるい法の実装では, 関係式探索部分の高速化に主眼が置かれてきた. しかし分解対象となる合成数が大きくなるにつれて他の部分の比重が相対的に高まってきている.

本報告書は, 数体ふるい法における多項式選択法部分・行列計算部分・イデアルの平方根計算部分に関する調査報告を目的とする. 第 1 章で数体

ふるい法のアルゴリズムの概略をまとめた後, 第 2 章で多項式選択部分,
第 3 章で行列計算部分, 第 4 章でイデアル平方根部分の調査報告を行う.

目 次

第1章　数体ふるい法

本章では数体ふるい法 (Number Field Sieve method, NFS) のアルゴリズムの概要をまとめる. 数体ふるい法全体のさらに詳しい流れについては, 既存の解説 ([10] など) を参照されたい. 以下では素因数分解したい合成数を n として表す. また複素数, 実数, 有理数, 整数, 自然数の集合をそれぞれ $\mathbb{C}, \mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}$ と表す.

1.1　数体ふるい法の概要

整数 n が整数係数の既約多項式 $f(x) \in \mathbb{Z}[x]$ と適当な整数 m に対して

$$f(m) \equiv 0 \pmod{n}$$

と表されているとする. $f(x)$ の複素数解の 1 つを $\theta \in \mathbb{C}$ とし, 代数体を $K = \mathbb{Q}(\theta)$, その整数環を $\mathcal{O}_K$ で表す. ここで $\mathcal{O}_K = \mathbb{Z}[\theta]$ は素元分解環であると仮定する.

F を $\mathbb{Z}$ の素数の集合, G を $\mathcal{O}_K$ の素元と単数の集合とする. a, b を互いに素な整数で, $a + bm$ は F-smooth かつ $a + b\theta$ は G-smooth なもの, つまり

$$\begin{cases} a + bm &= \prod_{p \in F} p^{e(p)} \\ a + b\theta &= \prod_{\pi \in G} \pi^{e(\pi)} \end{cases}$$

と分解できているとする. このようなペア (a, b) を $\#F + \#G$ 個よりも多く集めたとすると, 各べき指数を mod 2 で考えたベクトル

$$((e(p) \bmod 2)_{p \in F}, (e(\pi) \bmod 2)_{\pi \in G})$$

は線型従属となり, 適当に組み合わせることで次のような関係式が得られる:

$$\begin{cases} \prod (a_i + b_i m) &= \left(\prod_{p \in F} p^{e(p)}\right)^2, \\ \prod (a_i + b_i \theta) &= \left(\prod_{\pi \in G} p^{e(\pi)}\right)^2. \end{cases}$$

ここで $\prod(a_i+b_i\theta)=s(\theta)^2$, $s(x)\in\mathbb{Z}[x]$ とおく. $s(\theta)$ を $\mathbb{Z}[\theta]$ から $\mathbb{Z}/n\mathbb{Z}$ への環準同型

$$\begin{array}{rccc}\phi: & \mathbb{Z}[\theta] & \longrightarrow & \mathbb{Z}/n\mathbb{Z} \\ & f(\theta) & \longmapsto & f(m) \bmod n\end{array}$$

で写せば

$$\left(\prod_{\pi\in G} p^{e(\pi)}\right)^2 \equiv s(m)^2 \pmod{n}$$

となり, $x^2\equiv y^2 \pmod{n}$ の型の合同式が得られる. よって $\gcd(x\pm y, n)$ を計算することで (確率 $1/2$ で) n の約数が求められる. これが数体ふるい法の原理である.

1.2　特殊数体ふるい法

前節で数体ふるい法の概要を述べたが, いくつかのパラメータに対する仮定を用いていた. 特に $\mathcal{O}_K$ が素元分解環になるという条件は初期の数体ふるい法においては必須であり, このため分解可能な合成数は限られていた. 例えば $f(x)=x^3+2$ によって生成される代数体 $K=\mathbb{Q}(\sqrt[3]{-2})$ の整数環は素元分解環になるため, $n=2^{128}+1$ を

$$2n=f(2^{43})$$

と表すことによって分解を行っていた. このような場合の数体ふるい法を特殊数体ふるい法 (SNFS; Specialized Number Field Sieve method) と呼ぶ.

しかし実は整数環におけるイデアルの平方根計算アルゴリズムが開発されてからは, 整数環が素元分解環であることは重要でなくなった (4 章). そこで現在では多項式 $f(x)$ があらかじめ指定されている場合を特殊数体ふるい法と呼んでいる.

1.3　一般数体ふるい法

特殊数体ふるい法の大成功を受けて一般の合成数の素因数分解への適用も試みられた. それが一般数体ふるい法 (GNFS; General Number Field Sieve method) である. 実際の素因数分解実験においても一般数体ふるい法は成功を修めており, 現時点 (2004 年 1 月) の分解記録である 576-bit

表 1.1: 一般数体ふるい法による素因数分解記録

分解年	桁数	(ビット長)	合成数	分解者
1996	130 桁	(430-bit)	RSA-130	Montgomery
1998	140 桁	(463-bit)	RSA-140	Montgomery
1999	155 桁	(512-bit)	RSA-512	Montgomery
2002	158 桁	(524-bit)	$2^{953}+1$	Bahr,Franke,Kleinjung
2003	160 桁	(530-bit)	RSA-160	Bahr,Franke,Kleinjung,Lochter,Böhm
2003	174 桁	(576-bit)	RSA-576	Franke et al.

合成数の分解にも一般数体ふるい法が用いられた [8]. (参考までに一般数体ふるい法による分解記録を表 1.1 にまとめる.)

一般数体ふるい法では整数環 $\mathcal{O}_K$ は素元分解環とは限らない. そこで素元分解の代わりに $a+b\theta$ の生成するイデアルを素イデアル分解する. このため新たに整数環上でのイデアル平方根計算が必要となる. 素イデアル分解は $a+b\theta$ のノルム $\mathcal{N}_K(a+b\theta)$ を素因数分解することで得られるので, $f(x)$ の係数が小さい方が素イデアル分解が簡単となる. 従って一般数体ふるい法では「良い」多項式の生成法が重要である.

改めて整理すると一般数体ふるい法は以下の 5 つのステップから構成される.

- **多項式選択**

  合成数 n に対し, 整数係数の既約多項式 $f_1(x),\ f_2(x) \in \mathbb{Z}[x]$ を求める[1] . ただし $f_1(x),\ f_2(x)$ は $\mathbb{Z}/n\mathbb{Z}$ 上で共通解 $m \in \mathbb{Z}/n\mathbb{Z}$ を持つ, すなわち

  $$f_1(m) \equiv f_2(m) \equiv 0 \pmod{n}$$

  を満たすとする. $\theta_1,\ \theta_2 \in \mathbb{C}$ を $f_1(x)=0,\ f_2(x)=0$ の解とし, 代数体 $\mathbb{Q}(\theta_1),\ \mathbb{Q}(\theta_2)$ を考える. このとき写像

  $$\begin{array}{rccc} \varphi_1 & : & \mathbb{Z}[\theta_1] & \to & \mathbb{Z}/n\mathbb{Z} \\ & & a+b\theta_1 & \mapsto & a+bm \end{array} \qquad \begin{array}{rccc} \varphi_2 & : & \mathbb{Z}[\theta_2] & \to & \mathbb{Z}/n\mathbb{Z} \\ & & a+b\theta_2 & \mapsto & a+bm \end{array}$$

  はいずれも環準同型となる.

- **関係式探索**

  互いに素な整数の組 (a,b) であって, $a+b\theta_1,\ a+b\theta_2$ の分解が得られるようなものの集合 S を求める. 現実的にはふるい (sieve) と呼

[1] 実際, $f_2(x)$ は 1 次式なので, 多項式 $f(x)$ と整数 m のペアと見なすこともある.

ばれる処理を用いることから, 本ステップをふるいステップと呼ぶこともあり, 数体ふるい法の名称の由来でもある.

- **行列計算 (線型代数計算)**

  S の部分集合のうち, $\alpha_1 = \prod_{(a,b)\in T}(a + b\theta_1)$, $\alpha_2 = \prod_{(a,b)\in T}(a + b\theta_2)$ が $\mathbb{Z}[\theta_1]$, $\mathbb{Z}[\theta_2]$ の平方数となるような部分集合 T を求める.

- **イデアルの平方根計算**

  $\alpha_1 = \beta_1^2$, $\alpha_2 = \beta_2^2$ となるような $\beta_1 \in \mathbb{Z}[\theta_1]$, $\beta_2 \in \mathbb{Z}[\theta_2]$ を求める.

- **n の分解**

  $\gcd(\varphi_1(\beta_1) \pm \varphi_2(\beta_2), n)$ から n を分解する.

従来の一般数体ふるい法の実装では, 主に関係式探索ステップの高速化に主眼に置かれてきた. しかし分解対象となる合成数が大きくなるにつれて他の部分の比重が相対的に高まってきている.

第2章　多項式選択

本章では一般数体ふるい法における多項式選択法に関する調査報告を行う. 以下では m 進展開をベースにした従来の多項式選択法について簡単に説明した後, 現在の素因数分解実験において標準的に使用されている, B.A. Murphy による多項式選択法 [19] について報告する.

2.1　多項式選択法の概要

多項式選択とは, 素因数分解のターゲットである合成数 n を入力として, 2 つの整数係数既約多項式 $f_1(x)$, $f_2(x)$ を選択することである[1] . 多項式の優劣によって一般数体ふるい法全体の計算時間が大きく左右されてしまうため, 選択には十分な探索が必要となる [16]. なお特殊数体ふるい法においてはあらかじめ多項式が与えられているため, 多項式選択の必要はない[2] .

選択する多項式の最適な次数は簡単な計算から求めることができるので (d_1 の最適値は 2.3.1 節で求める. また $d_2 = 1$ である), 多項式選択において必要な条件は, $f_1(x)$, $f_2(x)$ が共通な整数解 m を持つことだけであり, 実際, このような多項式ペア $f_1(x)$, $f_2(x)$ を求めることは可能である. 例えば, あらかじめ定められた $f_1(x)$ の次数 d_1 に対して $m = \lfloor n^{\frac{1}{d_1}} \rfloor$ を求め, n の m 進展開を $n = c_{d_1} m^{d_1} + c_{d_1-1} m^{d_1-1} + c_0$ とおいたときに

$$\begin{aligned} f_1(x) &= c_{d_1} x^{d_1} + c_{d_1-1} x^{d_1-1} + \cdots + c_0, \\ f_2(x) &= x - m \end{aligned}$$

と定める[3] . ここで $\lfloor x \rfloor$ は x を越えない最小の整数を表す (Gauss 記号). このとき $f_1(m) = f_2(x) = 0$ となり, 多項式ペアとして必要な条件を満たしている[4] . 次に, このような多項式ペアを複数構成することも可能であ

---

[1] 非斉次多項式 $f_i(x)$ の代わりに斉次多項式 $F_i = F_i(x, y) = y^{d_i} f(x/y)$ ($d_i = \deg f_i(x)$) を用いることもある.

[2] 従って本章では, 単に数体ふるい法という場合には一般数体ふるい法を意味する.

[3] 選択法から $c_{d_1} = 1$ となるが, 同様にして $c_{d_1} \neq 1$ となる構成法も考えられる.

[4] ここで得られた $f_1(x)$ が既約である保証はないが, 可約の場合には n の素因数が求められるので, 我々にとっては都合が良い. 以下 $f_1(x)$ は既約と仮定する.

る. いま $f_1(x)$, $f_2(x)$ が上の条件を満たす多項式ペアであるとき, 任意の d_1-1 次以下の整数係数多項式 $p(x)$ に対して $\tilde{f}_1(x) = f_1(x)+(x-m)p(x)$ とおくと, $\tilde{f}_1(x)$, $f_2(x)$ は共通解 m を持つ多項式ペアとなる. 従って多項式選択ステップでは, このようにして生成されるたくさんの多項式ペアから最良のペアを探索していくことになる.

それでは数体ふるい法における多項式の良さとは何を指すのだろうか. 簡単に言うと, 良い多項式とは

> (同じ領域で比較したときに) 関係式探索ステップにおいてより多くのペア (a,b) をもたらす多項式

である. 関係式探索ステップで探索するペア (a,b) は, 多項式 $f_1(x)$, $f_2(x)$ (を斉次化したもの) に a,b を代入した値が smooth になることであったから, 良い多項式とは

> $f_1(x)$, $f_2(x)$ に (a,b) を代入した値が smooth になりやすい多項式

と言うこともできる. 一般に大きな値よりも小さな値の方が smooth になりやすいので, 良い多項式とは

> $f_1(x)$, $f_2(x)$ に (a,b) を代入した値の絶対値が小さい多項式

と言っても良い. Murphy はこれら多項式の良さを評価する関数 (収量) を導入し, 効率的な多項式探索を可能とした [19]. 多項式の良さについては 2.3 節で詳細に検討する.

多項式選択ステップの目標は, 多項式の良さを表す評価関数をもとに, たくさんの多項式ペアの候補から最適なものを見つけ出すことである. B.A. Murphy は解析数論的な評価関数 (収量) を構成することで, 効率的な多項式選択方法を提案した [19]. 歴史的に一般数体ふるい法が注目されるようになったのは 1996 年に RSA-130 の素因数分解に成功した以降であるが, それ以後現在に至るまで, Murphy の多項式選択法は現実に使用されている方法である[5] .

以下では文献 [19] を元に Murphy の多項式選択法について述べる. Murphy の多項式選択法は, m 進展開法をベースに候補となる多項式を多数生成し, 評価関数を元にその中から最適な多項式を探索していく. まず 2.2 節で m 進展開法についてまとめた後, 2.3 節で Murphy の評価関数である収量を導入する. 具体的な選択アルゴリズムについては 2.4 節で説明する.

[5] 2003 年 12 月に J.E.Gower が別の選択法を提案している [9].

2.2　m 進展開法

本節では多項式選択の基本となる m 進展開法について説明する.

2.2.1　m 進展開法

数体ふるい法の条件を満たす多項式の素朴な構成法は, 合成数 n の m 進展開を用いる方法である (m 進展開法, base-m method) [4]. あらかじめ定められた次数 d_1 に対して $m = C\lfloor n^{\frac{1}{d_1+1}} \rfloor$ とおく ($C > 1$ は後で定める定数) [6] . そして n の m 進展開 $n = \sum_{i=0}^{d_1} c_i^{(m)} m^i$ を求め, 多項式 $f_1(x)$, $f_2(x)$ を

$$f_1(x) = \sum_{i=0}^{d_1} c_i^{(m)} x^i, \quad f_2(x) = x - m$$

と定める. このとき多項式 $f_1(x)$, $f_2(x)$ は共通解 m を持つ. 一般に $f_1(x)$ はモニックとは限らない.

数体ふるい法の関係式探索ステップにおいて, なるべくたくさんのペア (a, b) を見つけられるようにするには, 代入された値はなるべく小さくあって欲しく, 多項式 $f_1(x)$, $f_2(x)$ の係数の絶対値は小さくあって欲しい. しかし m 進展開法では, 係数は $c_i^{(m)} < m = O\left(n^{\frac{1}{d_1+1}}\right)$ となっているため, 係数を小さくするには m を小さくするしかない. このため m が大きい場合 (つまり n が巨大な場合) には, m 進展開法によって係数の小さな多項式を生成することは困難である.

2.2.2　改良 m 進展開法

多項式 $f_1(x)$ の係数を小さくする方法に改良 m 進展開法 (improved base-m method) がある. いま n の m 進展開における係数 $c_{d_1}^{(m)}, \ldots, c_0^{(m)}$ が与えられたとき, 以下の手順によって係数列 $c_{d_1+1}, \ldots, c_0$ を求める (符号付き m 進展開):

$i = 0, \ldots, d_1$ に対し, $c_i^{(m)} > \frac{m}{2}$ ならば $c_i \leftarrow c_i^{(m)} - m$, $c_{i+1}^{(m)} \leftarrow c_{i+1}^{(m)} + 1$, そうでなければ $c_i \leftarrow c_i^{(m)}$ とする.

このとき新しい係数は $|c_i| \leq \frac{m}{2}$ を満たすので, m 進展開法で得られる係数に比べれば, そのサイズは半分となっている.

[6] $m = \lfloor n^{\frac{1}{d_1}} \rfloor$ とすると $c_{d_1} = 1$ となってしまい, 多項式 $f_1(x)$ はモニックになってしまう. これは後述する評価関数において好ましくないので, m を小さ目に設定することでモニックになることを回避する. なお単に $m = \lfloor n^{\frac{1}{d_1+1}} \rfloor$ とすると, $f_1(x)$ は d_1 次にならないことがある.

2.2.3　傾斜多項式

改良 m 進展開法が使用したテクニックは, $x = m$ を解に持つ多項式から, $x = m$ を解に持つ別の多項式の構成法を示唆している. 多項式 $f_1(x)$ が $x - m$ を解に持つとき, 任意の $d_1 - 1$ 次以下の整数係数多項式 $p(x)$ に対して $\tilde{f}_1(x) = f_1(x) + (x - m)p(x)$ とおくと, $\tilde{f}_1(x)$ は $x = m$ を解に持つ. 特に $p(x)$ として一次式を用いれば, 低次の項の係数だけが変化することになる.

確かに改良 m 進展開法で得られる係数のサイズは $m/2$ に削減できているが, 十分ではない. しかし全ての係数 c_i のサイズを小さくすることは困難であるため, 以下のような傾斜多項式を採用する. 数体ふるい法の関係式探索部分で必要なペア (a, b) は, $a + bm$, $a + b\theta$ がともに smooth となるようなものであった. ここで $a + b\theta$ が smooth であることと, そのノルム (に修正を加えたもの)

$$c_d \mathcal{N}_K(a + b\theta) = (-b)^{d_1} f\left(-\frac{a}{b}\right) = c_{d_1} a^{d_1} + c_{d_1 - 1} a^{d_1 - 1}(-b) + \cdots + c_0(-b)^{d_1}$$

が smooth であることは同値である. このとき

$$c_d \mathcal{N}_K(a + b\theta) = a^{d_1}\left(c_{d_1} + c_{d_1 - 1}\left(\frac{-b}{a}\right) + \cdots + c_0\left(\frac{-b}{a}\right)^{d_1}\right)$$

と変形できるが, 関係式を探索する領域においては $|b| \ll |a|$ となることが多いので, 係数 $c_{d_1}, c_{d_1 - 1}, \ldots, c_0$ を同じサイズに設定するよりも, c_{d_1} が小さく, c_0 が大きくなるように傾斜配分する方が有効である. このような多項式を傾斜多項式 (skewed polynomial) という. 以下の多項式選択では傾斜多項式の探索を行う.

2.3　多項式の良さ

数体ふるい法で用いる斉次多項式を $F(x, y) = y^{d_1} f_1(x/y)$ とすると, 良い多項式として求められる条件は, (x, y) にさまざまな (a, b) を代入したときに, なるべく

$F(a, b)$ のとり得る値が smooth である

ことである. 一般に小さな値の方が smooth になりやすいことから, この条件は

$F(a, b)$ のとり得る値が小さい

と考えられ, $f(x) = f_1(x)$ はなるべく多くの実数解を持つことが求められていた. Murphy は解の個数だけでなく, 解の持つ性質までを考慮した評価関数 (α 関数) を導入し, 定量的に評価することを可能にした. さらに Murphy は α 関数値をもとに, 実際に得られる関係式の個数の推定を行う評価関数として収量 (yeild) を導入した.

本節では, 多項式の良さに大きな影響を与える多項式の次数の最適値を求めた後, α 関数と収量を定義する.

補足 2.1 (従来の指標) 従来から良い多項式の条件として, たくさんの実数解を持つ必要性は指摘されていた. 実際, $a + b\theta$ が smooth となるためには $f(-a/b)$ の絶対値を小さくすることが有効であり, 方程式 $f(x) = 0$ が実数解を持つ場合, その解の近辺では $f(-a/b)$ の絶対値は小さくなるからである.

さらに Elkenbracht-Huizing は (a, b) の定義域を $[-H_a, H_a] \times [1, H_b]$ とした時、実数解の中に $\pm H_a/H_b$ に近いものがあることを挙げている [7]. しかし実際の数値実験では必ずしも必須であるとはいえず, 重要度は低いと考えられる.

2.3.1　多項式の次数

多項式 $f_1(x)$ の次数 $d = d_1$ の最適値を考える. 関係式探索ステップにおいて, a, b の絶対値の最大値を U とおき, $c_i < m$ を仮定すると,

$$F_1(a,b) = \sum_{i=0}^{d} c_i a^i b^{d-i} < (d+1)mU^d, \quad F_2(a,b) = a - mb < mU$$

となり,

$$F_1(a,b) \cdot F_2(a,b) < (d+1)m^2U^{d+1} < 2dm^2U^{d+1} \tag{2.1}$$

を得る (ここで $F_i(x,y) = y^{d_i} f(x/y)$ である). $m \approx N^{\frac{1}{d+1}}$ とした場合, F_1 は一般にはモニックでなく, 式 (2.1) より

$$F_1(a,b) \cdot F_2(a,b) < 2dN^{\frac{2}{d+1}}U^{d+1}$$

を得る. 一方で数体ふるい法の計算時間は高々

$$\exp\left((1+o(1))\left(d\log d + \sqrt{(d\log d)^2 + 4\log(N^{\frac{1}{d+1}})\log\log(N^{\frac{1}{d+1}})}\right)\right)$$

で与えられる [14]. この関数の指数部を

$$\mathcal{E}(d,N) = d\log d + \sqrt{(d\log d)^2 + 4\log(N^{\frac{1}{d+1}})\log\log(N^{\frac{1}{d+1}})}$$

とおき, 関数値が最小となるような d を直接求めることにする. d, $N = 10^i$ を適当な範囲で変化させたときの $\mathcal{E}(d, N)$ の計算結果を表 2.1 に示す [19]. 太字は d がその桁数で最良であることを意味する. この結果より N が $10^{120} \sim 10^{220}$ の範囲では $d = 5$ が適当であるといえる.

i	$\mathcal{E}(3, 10^i)$	$\mathcal{E}(4, 10^i)$	$\mathcal{E}(5, 10^i)$	$\mathcal{E}(6, 10^i)$	$\mathcal{E}(7, 10^i)$
80	29.85	**29.08**	29.92	32.06	35.29
90	31.89	**30.83**	31.44	33.36	36.38
100	33.83	**32.51**	32.90	34.61	37.44
110	35.69	**34.13**	34.30	35.83	38.48
120	37.49	35.68	**35.67**	37.01	39.50
130	39.21	37.19	**36.99**	38.17	40.50
140	40.89	38.65	**38.27**	39.29	41.48
150	42.52	40.06	**39.51**	40.39	42.44
160	44.09	41.44	**40.73**	41.47	43.38
170	45.63	42.78	**41.92**	42.52	44.30
180	47.13	44.09	**43.08**	43.54	45.21
190	48.59	45.37	**44.21**	44.55	46.10
200	50.02	46.62	**45.32**	45.54	46.98
210	51.42	47.84	**46.41**	46.51	47.85
220	52.79	49.04	**47.47**	**47.47**	48.70
230	54.14	50.22	48.52	**48.40**	49.53
240	55.46	51.38	49.55	**49.33**	50.36
260	58.03	53.63	51.56	**51.13**	51.98
280	60.51	55.81	53.50	**52.88**	53.55
300	62.92	57.92	55.39	**54.58**	55.08

表 2.1: $\mathcal{E}(d, N)$ の計算値

2.3.2　解の性質

本節では解の性質を調べるために, 関数 $F = F(x, y)$ の典型的な値 (典型的 F 値) と呼ばれる指標を定義する.

典型的 F 値

自然数 v を割る素数 p の最大べきを $\mathrm{ord}_p(v)$ で表す (v の p 進付値). ある集合 S に対して $v \in S$ が S を変化するとき, $\mathrm{ord}_p(v)$ の期待値を $\mathrm{cont}_p(S)$ とかく. S が十分大きいときには, $\mathrm{cont}_p(S)$ は

$$\mathrm{cont}_p(S) \approx \frac{\sum_{v \in S} \mathrm{ord}_p(v)}{|S|}$$

と近似することができる.

適当な大きさの整数 B に対し, $v \in S$ は平均的には

$$\log v = \sum_{p \le B} \mathrm{cont}_p(v) \cdot \log p$$

すなわち

$$v = e^{\sum_{p \le B} \mathrm{cont}_p(v) \cdot \log p}$$

と表すことができる.

S として関数 F の関数値の集合 S_F を考えたとき,

$$\mathrm{cont}_p(F) = \frac{\sum_{v \in S_F} \mathrm{ord}_p(v)}{|S_F|}$$

と定めることにすると, 関数 F の関数値は平均的には $e^{e^{\sum_{p \le B} \mathrm{cont}_p(F) \cdot \log p}}$ と表すことができる. この値を典型的 F 値 (typical F-value) と呼ぶ.

以下, 典型的 F 値と乱数のふるまいの違いを考察する.

$\mathrm{cont}_p(v)$ の近似式

$\mathrm{cont}_p(v)$ の近似計算法について述べる. v がランダムな整数の場合, $f(x)$ の関数値の場合, $F(x, y)$ の関数値の場合に分けて考える.

ランダムな場合　ランダムな整数 r を割る素数 p の最大べきの平均値は

$$\frac{1}{p} + \frac{1}{p^2} + \cdots = \frac{1}{p-1}$$

となるから, r の値域を R とすれば

$$\mathrm{cont}_p(R) = \frac{1}{p-1}$$

である. これは漸近的な結果であるが, 実際良い近似になっている.

$f(x)$ の関数値の場合　$v = f(x) = F(x,1)$ とする．$f(x) = 0$ の mod p での解は Hensel lifting により mod p^k での解に一意に対応するので, それぞれの寄与は $\frac{1}{p-1}$ である．したがって $f(x) = 0$ の mod p での異なる解の個数を q_p とすると,

$$\mathrm{cont}_p(f) = q_p \frac{1}{p-1}$$

となる．r のときと同様に, この式は良い近似となっている．

$F(x)$ の関数値の場合　$v = F(x,y)$ とする．このとき $F(x,y) = 0$ の mod p での解と mod p^k の解の間には一意的な対応関係がない．しかも $p|y$ の場合には新しい解の系列が現れてしまう (このような解を射影解と呼ぶ)．実際 $p|c_d$, $p|y$ の場合には $p|F(x,y)$ となる．

$\mathrm{cont}_p(F)$ の近似式を求めるには, p が分岐しない素数であることが必要である．以下 p は分岐しない場合を考える．なお p が分岐する素数であることと, p が $f(x) = F(x,1)$ の定める代数体の判別式を割ることは同値である．

$F(x,y) = 0$ の mod p での異なる解の個数を q_p とすると, q_p には $F(x,1)$ mod p の解と射影解が含まれる．これらを考慮すると

$$\mathrm{cont}_p(F) = q_p \frac{p}{p^2-1}$$

となる (証明は [19] の pp.42-43 を参照)．r のときと同様に, この式は良い近似となっている．

Murphy の α 関数

多項式の解の性質を表す関数として α 関数を導入する．数体ふるい法の関係式探索ステップにおいて, ある素数 p に関するふるいを行った後には, ランダムな整数 r に対応するレジストリには

$$\log r - \sum_{p \le B} \frac{\log p}{p-1}$$

が, 関数値 $v = f(x)$ または $v = F(x,y)$ に対応するレジストリには

$$\log v - \sum_{p \le B} \mathrm{cont}_p(f) \cdot \log p \quad or \quad \log v - \sum_{p \le B} \mathrm{cont}_p(F) \cdot \log p$$

が入っている. ここで両者の差を α とおいて, $v = r$ とすると

$$\begin{aligned}
\alpha(f) &= \left(\log v - \sum_{p \le B} \mathrm{cont}_p(f) \cdot \log p\right) - \left(\log r - \sum_{p \le B} \frac{\log p}{p-1}\right) \\
&= \sum_{p \le B} \left(\frac{1}{p-1} - \mathrm{cont}_p(f)\right) \log p, \\
\alpha(F) &= \left(\log v - \sum_{p \le B} \mathrm{cont}_p(F) \cdot \log p\right) - \left(\log r - \sum_{p \le B} \frac{\log p}{p-1}\right) \\
&= \sum_{p \le B} \left(\frac{1}{p-1} - \mathrm{cont}_p(F)\right) \log p
\end{aligned}$$

となる. これらの関数を α 関数と呼ぶことにする.

改めて v, r を別々に考えて, v が F 値をとるときを考える. ランダムな値の場合と F 値の場合でふるい後のレジストリの値が同じになるのは

$$\log r - \sum_{p \le B} \frac{\log p}{p-1} = \log F - \sum_{p \le B} \mathrm{cont}_p(F) \cdot \log p$$

すなわち

$$\begin{aligned}
\log r &= \log F + \sum_{p \le B} \left(\frac{1}{p-1} - \mathrm{cont}_p(F)\right) \log p \\
&= \log F + \alpha(F) = \log\left(F \cdot e^{\alpha(F)}\right) \qquad (2.2)
\end{aligned}$$

のときであることがわかる. この式が意味するのは, $F \cdot e^{\alpha(F)}$ と r が同程度の大きさのランダムな数になることである. よって $\alpha(F) < 0$ であれば, F は r よりも大きいにもかかわらず, r と同程度の smoothness を持つ. よって α 関数の値は小さい方が望ましい.

補足 2.2 Murphy の α 関数は, 連分数法や 2 次ふるい法と呼ばれる素因数分解法における, Knuth-Schroeppel 関数を数体ふるい法に適用したものであるので [19]. α 関数の導入は極めて自然と考えることができる.

2.3.3　収量の計算法

本節では多項式の良さを表す評価関数として, 収量を導入する. 収量は解の性質に大きく依存する (つまり α 関数値の関数として表される) が, それ以外の要素にも影響を受ける.

解の性質の関数としての収量

前節で定義した α 関数から収量を計算する方法を示す.

Boender の評価式　自然数 N の最大素因数を $P_1(N)$ とする. 与えられた自然数の組 $(x, y) \in \mathbb{N} \times \mathbb{N}$ に対し, x 以下の y-smooth な自然数の個数を $\psi(x, y)$ とおくと, $\psi(x, y)$ は

$$\psi(x, y) = \#\{\ N \in \mathbb{N} \mid N \leq x,\ P_1(N) \leq y\ \}$$

で与えられる. このとき漸近的に

$$\psi(x, y) \approx x\left(\rho(u) + (1-\gamma)\frac{\rho(u-1)}{\log x}\right) \tag{2.3}$$

となることが知られている. ここで $u = \log x / \log y$, γ は Euler 定数, ρ は Dickman 関数である.

一方で, 区間 $\left[x, x + \frac{x}{z}\right]$ 内の y-smooth な整数の個数は, ある実数 $\varepsilon \in (0, 1)$ を用いて

$$\begin{aligned}&\psi\left(x + \frac{x}{z}, y\right) - \psi(x, y)\\ &\quad = \frac{\log\left(1 + \frac{y}{\log x}\right)}{z \log y}\psi(x, y)\left[1 + O_\varepsilon\left(\frac{1}{z} + \frac{\log\log(1+y)}{\log y}\right)\right]\end{aligned} \tag{2.4}$$

で与えられることが知られている (Boender, [19]). ただし $x \geq 2$ であり, ある関数 $R(x, y)$ に対して

$$(\log\log x)^{\frac{2}{3}+\varepsilon} < \log y \leq (\log x)^{\frac{3}{5}}, \quad 1 \leq z \leq R(x, y)$$

を満たすものとする. よって式 (2.3) と式 (2.4) により, 区間 $\left[x, x + \frac{x}{z}\right]$ 内の整数が y-smooth になる割合は

$$\begin{aligned}&\frac{z}{x}\left[\psi\left(x + \frac{x}{z}, y\right) - \psi(x, y)\right]\\ &\qquad \approx \left(1 - \frac{\log\log x}{\log y}\right)\sigma(x, y)\left(1 + C_1\frac{1}{z} + C_2\frac{\log\log y}{\log y}\right)\end{aligned} \tag{2.5}$$

ただし

$$\sigma(x, y) = \rho(u) + (1-\gamma)\frac{\rho(u-1)}{\log x}$$

で, C_1, C_2 は ε によって定まる定数である.

収量の推定　x が区間 $[a_1, a_2]$ を動くときに, $|f(x)|$ が B-smooth となる割合を, 式 (2.5) を用いて評価したい (ここで $f(x)$ はこの区間内の連続関数と見なす). この区間内に $f(x)$ の極点や実解を持つことがありえるが, このような点は区間を segment に分割する際に除去されると考える. これら segment のうち最も長い segment を principal segment と呼ぶことにする. I を x の区間を定める principal segment, Γ を f によって定まる I 上の連続曲線とする. I は極点を持たないので, 任意の $x \in I$ に対して $f'(x) < 0$ または $f'(x) > 0$ が成立する. 以下 $f'(x) > 0$ の場合を考える. x が I を変化するときの Γ の最小値と最大値を S_1, S_2 として, 区間 $[S_1, S_2]$ を K 個の部分区間 $[y_i, y_{i+1}]$ $(i = 0, \ldots, K-1)$ に分割する. ただし

$$y_i = S_1 \times e^{ih}, \quad h = \frac{\log S_2 - \log S_1}{K}$$

とする. このとき $z = \frac{1}{e^h - 1}$ に対して $y_{i+1} = y_i + \frac{y_i}{z}$ となっている.

このような条件下で Γ の何個の整数点が B-smooth になるかを求めたい. 与えられた y_i に対し, f 値 $y \le y_i$ が整数でかつ B-smooth になるような y の個数を $t(y_i)$ と表す. このとき Γ 上にある B-smooth な整数点の個数 X_f は

$$X_f = \sum_{i=0}^{K-1} (t(y_{i+1}) - t(y_i)) \tag{2.6}$$

で与えられる. よって $t(y_{i+1}) - t(y_i)$ が評価できればよい.

$x_i = f^{-1}(y_i)$, $x_{i+1} = f^{-1}(y_{i+1})$ として, 区間 $[x_i, x_{i+1}]$ に対応する Γ の部分曲線を傾き

$$s_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

の直線と見なす. このときランダムに選んだ $y \in [y_i, y_{i+1}]$ に対して $x = f^{-1}(y) \in \mathbb{Z}$ となる確率は $1/s_i$ であるから,

$$t(y_{i+1}) - t(y_i) \approx \frac{y_{i+1} - y_i}{s_i} P(f_i, B) = (x_{i+1} - x_i) P(f_i, B)$$

と近似できる. ここで $P(f_i, B)$ は区間 $[y_i, y_{i+1}]$ の整数 f 値が B-smooth になる確率である.

前節で導出した式 (2.2) より, ランダムな値の対数値は $\log f + \alpha(f)$ と

近似できるので, 式 (2.5) より

$$\begin{aligned} P(f_i, B) &= \frac{z}{y_i}\left[\psi\left(y_i + \frac{y_i}{z}, B\right) - \psi(y_i, B)\right] \\ &\approx \left(1 - \frac{\log g_i(\alpha)}{\log B}\right)\left(\rho(v_i(\alpha)) + (1-\gamma)\frac{\rho(v_i(\alpha)-1)}{g_i(\alpha)}\right) \\ &\quad \times \left(1 + \frac{C_1}{z} + \frac{C_2 \log\log B}{\log B}\right) \end{aligned}$$

を得る. ただし

$$\begin{aligned} g_i(\alpha) &= \log y_i + \alpha = \log S_1 + ih + \alpha \\ v_i(\alpha) &= \frac{g_i(\alpha)}{\log B} \end{aligned}$$

とおいた. 以上のことから, $t(y_{i+1}) - t(y_i)$ の評価式

$$\begin{aligned} &t(y_{i+1}) - t(y_i) \\ &\approx (x_{i+1} - x_i)\left(1 - \frac{\log g_i(\alpha)}{\log B}\right)\left(\rho(v_i(\alpha)) + (1-\gamma)\frac{\rho(v_i(\alpha)-1)}{g_i(\alpha)}\right) \\ &\qquad \times \left(1 + \frac{C_1}{z} + \frac{C_2 \log\log B}{\log B}\right) \end{aligned}$$

を得る. この式を式 (2.6) に代入すれば, X_f の近似式が得られる.

2.3.4　収量

関数 $f(x) > 0$ に対して

$$u_f(x) = \frac{\log f(x) + \alpha(f)}{\log B}$$

とおく. また

$$P(f(x), B) \approx \rho(u_f(x)) + (1-\gamma)\frac{\rho(u_f(x)-1)}{\log f(x)}$$

であると仮定する. このときある区間 I に対して

$$\sum_{x \in I} P(f(x), B) \approx \sum_{x \in I}\left[\rho(u_f(x)) + (1-\gamma)\frac{\rho(u_f(x)-1)}{\log f(x)}\right]$$

となる. 実際の計算では区間 I が大きくなってしまうので, I を K 個に分割して, その部分区間を I_K とかくことにする. このとき I 上の関数 f の収量は

$$X_f \approx \frac{|I|}{K} \cdot \sum_{x \in I_K}\left[\rho(u_f(x)) + (1-\gamma)\frac{\rho(u_f(x)-1)}{\log f(x)}\right]$$

で与えられる.

しかし実際の計算によると, 収量計算は $\sum_{x \in I_K} \rho(u_f(x))$ だけを計算すれば十分である. よって多項式 F の収量関数 E を

$$E(F) = \sum_{x \in I_K} \rho(u_F(x)).$$

で定義する. この関数値が大きい方が良い多項式を表していると考えられる.

2.4　アルゴリズム

前節で定義した収量を用いて, 本節では多項式選択法のアルゴリズムについてまとめる. 多項式選択は 2 段階に分けて行われる. はじめの候補探索ステップでは, 改良 m-進展開法をもとに候補となる多項式を多数生成する. 次の絞り込みステップでは, 候補多項式の収量を求め, 候補の中から最良のものを選び出すことになる.

2.4.1　候補多項式の生成

与えられた合成数 n に対して, 多項式 $f(x)$ の次数 d を固定し, $m \approx \left\lfloor n^{\frac{1}{d+1}} \right\rfloor$ とおく. ここで n の m 進展開を

$$n = \sum_{i=0}^{d} c_i^{(m)} m^i, \quad 0 \le c_i^{(m)} < m$$

とする. 次に $c_i = c_i^{(m)}$ $(i = 0, 1, \ldots, d)$ とコピーしておいて, 再び $i = 0, 1, \ldots, d$ に対し $c_i > \left\lfloor \frac{m}{2} \right\rfloor$ ならば $c_i \leftarrow c_i - m$, $c_{i+1} \leftarrow c_{i+1} + 1$ と値を変更していき, 多項式 $f_m(x)$ を

$$f_m(x) = \sum_{i=0}^{d} c_i x^i, \quad |c_i| \le \left\lfloor \frac{m}{2} \right\rfloor$$

と定める. ここで多項式 $f_m(x)$ は m の関数として得られているので, m を変化させると, それに対応して多項式 $f_m(x)$ も変化することになる.

良い多項式を見つけるには, α 関数値が小さくなるような関数のみを生成する方が効率的である. そこで多項式 $f(x)$ の最高次の係数 c_d も入力値と考えることにする. 2.3.2 節の考察によると, c_d が smooth である方が α 値は小さくなったので, c_d としては小さな素数のべき積を用いることにする.

例 2.3 RSA-140 に対する多項式生成を考える．係数 c_d として区間 $[10^{20.3}, 10^{21.3}]$ から選ぶことにする．特に c_d の因数 c が以下の 5 つの条件を満たすように多項式をランダムに 100 個ずつ選んでいき，α 関数値の平均値を調べる．

1. c_d が素数 (最悪の場合)
2. c_d がランダムになっている (平均的な場合)
3. $c|c_d,\ c = 2 \times 3 \times 5 \times 7$ のとき (良い場合)
4. $c|c_d,\ c = 2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 = 10^{9.8}$ のとき (より良い場合)
5. $c|c_d,\ c = 2^5 \times 3^4 \times 5^3 \times 7^3 \times 11^2 \times 13 \times 17 \times 19 \times 23 \times 29 = 10^{16.6}$ のとき (最良の場合)

このときの $\alpha(F)$ の値の様子を表 2.2 にまとめる．

	$\min \alpha(F)$	mean $\alpha(F)$	$\max \alpha(F)$
1.	-0.67	1.16	2.63
2.	-1.62	0.44	2.33
3.	-3.02	-0.77	0.77
4.	-3.25	-1.40	0.15
5.	-3.54	-1.88	-0.47

表 2.2: RSA-140 の場合の $\alpha(F)$ の変化

結果から c が smooth であるほど α 関数値は小さくなることが分かる．

次のステップとして，多項式 $f_m(x)$ を適当に変換して α 値が小さい多項式 $f(x)$ を構成したい．ここで変換には以下の 2 種類の方法を使用する:

- Translation by t

  ある整数 $t \in \mathbb{Z}$ に対し $f_t(x) = f(x - t)$ とする．$m_t = m + t$ とすれば $f_t(m_t) \equiv 0 \pmod{n}$ となり，解の性質は変わらない．しかし係数のサイズを小さくすることができる．

- Rotation by $p(x)$

  ある多項式 $p(x) \in \mathbb{Z}[x]$ に対し $f_p(x) = f(x) + p(x)(x - m)$ とする (ただし $p(x)$ の次数は $d = \deg f$ よりも小さいとする)．このとき $f_p(m) \equiv 0 \pmod{n}$ となり，解の性質は変わらない．しかし係数のサイズを変えることができる．

多項式選択では最終的に傾斜多項式を求めることを目標にしているので, 傾斜多項式から遠い多項式はこの段階で候補から除外した方が良い. 実際, $f_m(x)$ から傾斜多項式

$$f(x) = f_m(x_t) + (c_1 x_t + c_0)(x_t - m_t), \quad x_t = x + t, m_t = m + t$$

を求めるのだが, 3 次以上の係数は変化することがない. そこでこの段階で

$$\frac{n - c_d m^d}{m^{d-1}} = c_{d-1} + \frac{c_{d-2}}{m} + \ldots$$

を計算し, 右辺の整数部分と小数部分が小さくない多項式は除外することにする.

例 2.4 RSA-140 の場合の多項式の組を 2 つ例示する. (F_1, F_2) は RSA-140 の素因数分解に実際に用いた傾斜多項式である. これに対し (G_1, G_2) は傾斜でない多項式であり, その中では最良のものである.

$$\begin{aligned}
F_1(x,y) &= 439682082840x^5 \\
&\quad +390315678538960x^4y \\
&\quad -7387325293892994572x^3y^2 \\
&\quad -190271532437429887148 24x^2y^3 \\
&\quad -6344102569446461791393061 3xy^4 \\
&\quad +3185539170714743503922235074 94y^5 \\
F_2(x,y) &= x - 3443565780924253695177900 7y
\end{aligned}$$

$$\begin{aligned}
G_1(x,y) &= 2378666111034213000 00x^5 \\
&\quad -5148567155828225103 04x^4y \\
&\quad -4722668925346720843884x^3y^2 \\
&\quad +6545365626333869758617x^2y^3 \\
&\quad -3356924353646091366162xy^4 \\
&\quad -5142225622472630020004y^5 \\
G_2(x,y) &= x - 6171197423044469387519 13y
\end{aligned}$$

このとき $\alpha(F_1) = -7.0$, $\alpha(G_1) = -4.2$ となっており, F_1 の方がはるかに好ましいといえる.

こうして好ましい多項式 $f_m(x)$ が得られたらば, α 関数値が最小となるようなパラメータ t, c_1, c_0 を定める. まず関数の平均サイズ

$$I(F,S) = \log\left(\sqrt{\iint_S F^2(x,y)dxdy}\right)$$

が最小となるように t, c_1, c_0 を (仮に) 定め, この値が小さくない場合には除外する. ただし $S = \{\ (x,y) \mid |x| < \sqrt{s},\ |y| < 1/\sqrt{s}\ \}$ を矩形領域と呼び, s は大きさを定めるパラメータである. 次に $f(x)$ の近辺で $I(F_{j_1,j_0}) + \alpha(F_{j_1,j_0})$ 値が最小となる j_1, j_0 を求める. ここで

$$f_{j_1,j_0}(x) = f(x) + (j_1 x + j_0)(x - m)$$

である. このようにして α 関数値が小さい多項式 $f_{j_1,j_0}(x)$ が得られる.

以上をまとめると, 以下のアルゴリズム 2.5 が得られる.

アルゴリズム 2.5 (傾斜多項式の候補生成) 与えられた合成数 n と次数 d に対し, α 関数値が小さな傾斜多項式 $f_{j_1,j_0}(x)$ を以下のようにして求める.

1. 係数 c_d と m を定める. 実際, c_d としては小さな素数のべき積で割り切れるものを選び,

   $$m = \left\lfloor \left(\frac{n}{c_d}\right)^{\frac{1}{d}} \right\rfloor$$

   とする. さらに

   $$\frac{n - c_d m}{m^{d-1}} = c_{d-1} + \frac{c_{d-2}}{m} + O(m^{-2})$$

   の整数部分と小数部分が十分小さくなることを確認する.
2. 多項式 $f_m(x) = \sum_{i=0}^{d} c_i x^i$ を求める. 次に Translation by t と Rotation by $p(x) = c_1 x + c_0$ を用いて, 多項式

   $$f(x) = f_m(x_t) + (c_1 x_t + c_0)(x_t - m_t)$$

   を求める. ただし $f(x)$ は矩形領域

   $$S = \{\ (x,y) \mid |x| < \sqrt{s},\ |y| < 1/\sqrt{s}|\ \}$$

   において小さい値を持つとする. ここで t, c_1, c_0, s はパラメータである. 次に

   $$\iint_S F^2(x,y)dxdy$$

が最小になるような t, c_1, c_0, s を求め, t, c_1, c_0 を整数にまるめた後に s は再計算する. 最後に

$$I(F,S) = \log\left(\sqrt{\iint_S F^2(x,y)dxdy}\right)$$

を計算し, 十分に小さいことを確認する.

3. 解の性質を調べる. 2 つの自然数 $J_1 \ll J_0$ が与えられているとき, $|j_1| < J_1$, $|j_0| < J_0$ となる整数 j_1, j_0 に対する多項式 $f_{j_1,j_0}(x)$ を以下で定める.

$$f_{j_1,j_0}(x) = f(x) + (j_1 x + j_0)(x - m).$$

この多項式が良い解を持つことを「ふるい法」によって調べる.

4. $I(F_{j_1,j_0},S) \approx I(F,S)$ だから, $I(F_{j_1,j_0},S)+\alpha(F_{j_1,j_0})$ を initial rating とおく. initial rating が小さければ

$$f(x) + (j_1 x + j_0)(x - m)$$

の係数を求める.

補足 2.6 上の Step 3. で用いる「ふるい法」の手順は以下の通りである：小さな素数のべき p^k と j_1 を固定して, j_0, ℓ を変数とする. 各 $\ell = 0, 1, \ldots, p^k$ に対し $f_{j_1,j_0}(\ell) \bmod p^k$ を計算し,

$$f_{j_1,j_0}(\ell) \equiv 0 \pmod{p^k}$$

となる $j_0 \in \mathbb{Z}/p^k\mathbb{Z}$ を求める. この j_0 に対応する $\mathrm{cont}_{p^k}(F_{j_1,j_0})$ を計算し記録する. この作業を各 p, j_1 に対して行うと, 小さい素数に対する $\alpha(F_{j_1,j_0})$ が全て計算されたことになる.

2.4.2　Dickman 関数の計算方法

収量計算で必要となる Dickman 関数 $\rho(u)$ についてまとめる. $\rho(u)$ は以下で定義される:

$$\rho(u) = \lim_{r\to\infty} \frac{\psi(r, r^{1/u})}{r} \quad \text{for } u > 1$$

また $u > 1$ でない場合には $\rho(u) = 1$ とする. $\rho(u)$ は, 直感的には r 以下の B-smooth な整数の割合を表している ($B = r^{1/u}$).

$u \in [\ell - 1, \ell]$ とする $(\ell \geq 0)$. このときこの区間で等しい値をとる解析関数 $\rho^{(\ell)}(u)$ が存在することが知られているので, $\rho^{(\ell)}(u)$ が計算できればよい. $\rho^{(\ell)}(u) = \rho^{(\ell)}(\ell - \xi)$ を計算するには, 関係式

$$\rho^{(\ell)}(\ell - \xi) = \sum_{i=0}^{\infty} d_i^{(\ell)} \xi^i$$

を用いる. ここで

$$d_0^{(2)} = 1 - \log 2, \quad d_i^{(2)} = \frac{1}{i 2^i}$$

$$d_0^{(\ell)} = \frac{1}{\ell - 1} \sum_{i=1}^{\infty} \frac{d_i^{(\ell)}}{i+1}, \quad d_i^{(\ell)} = \sum_{j=1}^{i-1} \frac{d_j^{(\ell-1)}}{i \ell^{i-j}}$$

である.

2.4.3　多項式の絞り込み

2.4.1 節の候補生成によって良い性質を持った多項式が多数得られる. 本節ではこれら多項式の中から最良のものを選び出す方法を紹介する.

傾斜でない多項式の場合

簡単のため, まず傾斜でない多項式の場合で説明する. 始めに $\alpha(F)$ を正確に計算する必要がある. 小さな素数 $p(< 100)$ に対しては $\mathrm{cont}_p(F)$ を直接計算し, $100 < p < 2000$ となる素数 p に対しては平均値を用いて $\mathrm{cont}_p(F)$ を推測する.

次に多項式 $F(x, y)$ の収量を計算する. $F(x, y)$ は斉次だから, 極座標への変換

$$F(x, y) = r^d F(\cos\theta, \sin\theta)$$

を考えることができる. 固定された $\theta = \theta_i$ に対して d 次の多項式の値は θ_i にそって r^d 倍されていくので, $F(\cos\theta_i, \sin\theta_i)$ を考察すれば良い. F をパラメータとする関数

$$u_F(\theta_i) = \frac{\log |F(\cos\theta_i, \sin\theta_i)| + \alpha(F)}{\log B}$$

を考える. 区間 $[0, \pi]$ を K 個の部分区間に分割し,

$$\theta_i = \frac{\pi}{K} \left(i - \frac{1}{2} \right) \qquad (i = 1, 2, \ldots, K)$$

とおく (i 番目の部分区間の中間値). 次に

$$E(F_1) = \sum_{i=1} K\rho(u_F(\theta_i))$$

を計算し, この値を F の rating と見なす. 同様にして候補多項式の rating を計算していき, 最も数値の大きいものを選択する.

傾斜多項式の場合

次に傾斜多項式 $F_1(x, y)$ の場合で考える. 矩形領域 S の縦横比を s とおく. このとき $s_1 = \sqrt{s}$, $s_2 = 1/\sqrt{s}$ として

$$x = s_1 \cos\theta, \quad y = s_2 \sin\theta$$

とおく. 傾斜でない多項式の場合と同様に区間 $[0, \pi]$ を K 個に分割し, $j = 1, 2$ に対し

$$u_{F_j}(\theta_i) = \frac{\log|F_j(s_1\cos\theta_i, s_2\sin\theta_i)| + \alpha(F_j)}{\log B_{F_j}}$$

を計算する. そして多項式ペア (F_1, F_2) の rating として

$$E(F_1, F_2) = \sum_{i=1}^{K} \rho(u_{F_1}(\theta_i))\rho(u_{F_2}(\theta_i))$$

を求める. 同様にして候補多項式ペアの rating を計算していき, 最も数値の大きいものを選択する.

2.5　RSA-130 に対する計算例

Murphy の多項式選択法の効果を示すため, 本節では RSA-130 に対する多項式について考察する. RSA-130 は数体ふるい法によって素因数分解された初めての大規模実験例であるが, その多項式生成法は明らかになっていない [6]. そこで分解に用いられた多項式と Murphy の選択法によって得られる多項式を比較し, Murphy の選択法が優れていることを示すのが本節の目的である.

RSA-130 の分解実験において使用された多項式 $f(x)$ と解 m は以下の

P_i	f_1 の収量		Initial	収量	$\alpha(F)$		E
	ratio	ranking	ranking	ranking	α 値	ranking	ranking
P_{14}	1.000	(1)	(14)	(2)	−0.40	(1)	(1)
P_4	0.991	(2)	(4)	(1)	0.58	(5)	(4)
P_1	0.937	(3)	(1)	(5)	1.09	(9)	(2)
P_{12}	0.875	(4)	(12)	(7)	−0.01	(2)	(3)
P_8	0.822	(5)	(8)	(10)	0.62	(6)	(11)
P_3	0.800	(6)	(3)	(3)	1.96	(13)	(6)
P_{10}	0.778	(7)	(10)	(4)	0.78	(7)	(8)
P_2	0.768	(8)	(2)	(6)	1.15	(10)	(5)
P_{11}	0.766	(9)	(11)	(11)	1.04	(8)	(10)
P_{15}	0.759	(10)	(15)	(8)	0.44	(4)	(7)
P_9	0.754	(11)	(9)	(9)	0.20	(3)	(9)
P_5	0.701	(12)	(5)	(13)	1.50	(11)	(12)
P_7	0.649	(13)	(7)	(14)	2.60	(15)	(14)
P_{13}	0.642	(14)	(13)	(12)	1.88	(12)	(13)
P_6	0.578	(15)	(6)	(15)	2.15	(14)	(15)
相関関係		基準	0.10	0.86		0.62	0.88

表 2.3: RSA-130 の候補多項式の収量値

通りである [15, 6]:

$$\begin{aligned} f(x) &= 574830224\ 8738405200\ x^5 \\ &\quad +988226191\ 7482286102\ x^4 \\ &\quad -1339249938\ 9128176685\ x^3 \\ &\quad +1687525245\ 8877684989\ x^2 \\ &\quad +375990017\ 4855208738\ x \\ &\quad -4676993055\ 3931905995, \\ m &= 125\ 7441116841\ 8005980468. \end{aligned}$$

明らかに $f(x)$ は傾斜多項式ではない.

2.5.1　生成法

多項式 $f(x)$ の生成手順を以下に示す [15, 6, 19].

1. $d = 5,\ m_0 \le m \le m_1$ と定める. ただし

$$\begin{aligned} m_0 &= \left\lfloor N^{\frac{1}{d+1}} \right\rfloor + 1 = 34\ 9003391158\ 4824772400 \\ m_1 &= \left\lfloor N^{\frac{1}{d}} \right\rfloor = 710224\ 3798957686\ 6574479042 \end{aligned}$$

P_i	m	χ	c_5
P_1	10519776768693341771145	0.00133	$811 \cdot 5479 \cdot 5483 \cdot 575711441$
P_2	12112464325781598662255	0.00057	$33469 \cdot 207096419302307$
P_3	12175183789358781924382	0.00150	$2 \cdot 3 \cdot 283 \cdot 3977979735853873$
P_4	12922982589397980905651	0.00094	$19 \cdot 23 \cdot 103 \cdot 111389799839027$
P_5	10056778742160802578928	0.00175	$2 \cdot 3^2 \cdot 83 \cdot 311 \cdot 37807017464459$
P_6	12893568754859383127665	0.00088	$2 \cdot 97588501 \cdot 25982655563$
P_7	13239320351370744041131	0.00132	$29 \cdot 1163 \cdot 24077 \cdot 5471033089$
P_8	12506435569527239916746	0.00129	$2^4 \cdot 43 \cdot 1019 \cdot 27581 \cdot 305448151$
P_9	12666132133378233425814	0.00044	$2^3 \cdot 13 \cdot 53299663918756421$
P_{10}	10844346817052874470999	0.00168	$281 \cdot 653 \cdot 2797073 \cdot 23476729$
P_{11}	13139341559800540682218	0.00403	$3 \cdot 5 \cdot 307622863517626499$
P_{12}	12857394860965184611325	0.00081	$2^2 \cdot 3 \cdot 428579828826125071$
P_{13}	11856745579968929283390	0.00109	$2 \cdot 1627 \cdot 522883 \cdot 4532400781$
P_{14}	12574411168418005980468	0.00134	$2^4 \cdot 3^4 \cdot 5^2 \cdot 13 \cdot 19^2 \cdot 331 \cdot 114213131$
P_{15}	11507478393662235457656	0.00088	$2^2 \cdot 2297 \cdot 13121 \cdot 74282945903$

表 2.4: RSA-130 の候補多項式のパラメータ

とする.

2. 固定された矩形領域 S 上で, ノルムの平均の大きさが小さい多項式 15 個を選び, initial rating の良い順に $P_1, ..., P_{15}$ とする[7] .
3. 15 個の多項式の詳細な収量を, 実際のふるいを通して求める (この収量は Murphy の収量とは異なる).
4. 収量が最大の多項式を出力する. 結果として P_{14} が出力された.

評価で用いた多項式 $P_1, \ldots, P_{15}$ の (f_1 の) 収量値を表 2.3 に示す [6]. また比較のため, 実際の (f_1 と f_2 の) 収量値による ranking, α 関数値と ranking, Murphy の収量 E による ranking も同じ表に示す [19].

2.5.2　考察

まず気づくのは, initial ranking と, f_1 の収量の ranking が大きく異なっていることである. 実際これら 2 つの ranking の相関関係は 0.10 となり, Huddlestone の ranking と f_1 の収量の ranking は無関係といえる. 従って Huddlestone が捨てた多項式の中に良いものが含まれているかもしれない.

多項式 $P_1, \ldots, P_{15}$ で使用される共通解 m と最高次の係数 c_5 の値を表 2.4 に示す. これら 15 個の多項式の χ 値はとても小さくなっていることから ($\chi \leq 0.004$), 生成段階で係数のサイズに注意が払われていたことが容易に予想できる[8] . しかし各多項式の α 関数値を計算すると, ほとん

[7] Scott Huddleston による計算と書かれているが, 計算法の詳細は不明である.
[8] ただし $\chi(f) = \min c_i/m$ と定める.

どの多項式の α 関数値は正であり, また最高次の係数 c_5 の smoothness から考えても, 多項式の生成時では解の性質は考慮されなかったと思われる. ただし最終的に絞り込まれた多項式 P_{14} は, 候補多項式の中で α 関数値が最小で, また係数 c_5 の smoothness も十分である.

他方で f_1 の収量の ranking と α 関数値の ranking の相関関係はまずまずであるが, f_1, f_2 の収量の ranking の相関関係よりは小さい. しかし Murphy 収量 E による ranking との相関関係は 0.88 であり, E の値が (f_1 の) 収量の良い近似になっていることがわかる. よって E 値の正当性が (少なくともこれら多項式に対しては) 示される.

2.5.3　他の候補多項式の生成

RSA-130 の分解実験 [6] で使用された多項式 P_{14} よりも良い多項式を求めてみよう. 方針として, 係数 c_d, c_{d-1} が小さくなるようなものを選び, E 値によって絞り込みを行う. ただし c_d の smoothness はあまり気にしないことにする. また解の性質もあまり考慮しない. ふるいを行った範囲は $-10^4 \le a \le 10^4$, $1 \le b \le 10^4$ で, $B = 11380951$ とした (このとき P_{14} から得られた relation は 15990 組であった).

実験によって得られた多項式 $Q_1, \ldots, Q_{18}$ の性能を表 2.5 に, パラメータを表 2.6 に示す. 表から分かる通り, 多項式 Q_1 は多項式 P_{14} に比べて 1.47 倍の relation を見つけることができる. なおこれら実験では f_1, f_2 の双方を用いた収量の相関関係は低くなっているが, やはり E 値の相関関係は高く, 良い近似になっていることがわかる.

次に解の性質まで考慮したアルゴリズムを用いた場合の多項式を生成してみよう. 得られた多項式 $R_1, \ldots, R_6$ の性能を表 2.7 に, パラメータを表 2.8 に示す. ふるいの範囲は $-10^6 \le a \le 10^6$, $1 \le b \le 10^6$, $B_1 = 11380951$, $B_2 = 120000000$ とした (ただし B_2 は large prime の上限値). 表からわかる通り, 多項式 R_1 は多項式 P_{14} に比べて 2.00 倍の relation を見つけることができる.

Q_i	f_1 の収量		収量	$\alpha(F)$		E
	ratio	ranking	ranking	α 値	ranking	ranking
Q_1	1.47	(1)	(5)	−3.62	(1)	(1)
Q_2	1.37	(2)	(6)	−2.45	(4)	(3)
Q_3	1.34	(3)	(2)	−2.15	(8)	(6)
Q_4	1.33	(4)	(4)	−2.70	(2)	(7)
Q_5	1.31	(5)	(8)	−2.15	(8)	(2)
Q_6	1.30	(6)	(12)	−2.01	(13)	(8)
Q_7	1.28	(7)	(17)	−2.19	(7)	(4)
Q_8	1.27	(8)	(13)	−2.68	(3)	(10)
Q_9	1.26	(9)	(11)	−1.93	(14)	(5)
Q_{10}	1.23	(10)	(1)	−2.03	(12)	(9)
Q_{11}	1.23	(11)	(18)	−2.43	(5)	(15)
Q_{12}	1.23	(12)	(9)	−2.24	(6)	(11)
Q_{13}	1.22	(13)	(14)	−2.08	(11)	(18)
Q_{14}	1.16	(14)	(10)	−1.82	(16)	(13)
Q_{15}	1.16	(15)	(3)	−1.71	(17)	(12)
Q_{16}	1.16	(16)	(15)	−0.95	(18)	(14)
Q_{17}	1.15	(17)	(7)	−2.10	(10)	(17)
Q_{18}	1.14	(18)	(16)	−1.87	(15)	(16)
相関関係		基準	0.53		0.56	0.91

表 2.5: RSA-130 の他の候補多項式の収量値

Q_i	m	χ	c_5
Q_1	13892376347633905755115	0.00473	$5^3 \cdot 54877 \cdot 509090282329$
Q_2	12453346471414472759941	0.01515	$3 \cdot 86209 \cdot 300229 \cdot 77700001$
Q_3	12189668945503746069685	0.01860	$174360559 \cdot 38509694549$
Q_4	11837358189073863960965	0.01275	$3^3 \cdot 5 \cdot 59 \cdot 571 \cdot 1709550942811$
Q_5	14227836633450858685725	0.01415	$2 \cdot 7^2 \cdot 53 \cdot 977 \cdot 18517 \cdot 32984993$
Q_6	12846317334855496412374	0.00660	$2^3 \cdot 7 \cdot 13 \cdot 23 \cdot 73 \cdot 83 \cdot 617 \cdot 2237 \cdot 36887$
Q_7	12485318267855789022719	0.01808	$3^2 \cdot 43 \cdot 73 \cdot 449 \cdot 272141 \cdot 1725463$
Q_8	13664023713239125138661	0.00950	$181 \cdot 270631 \cdot 77451303079$
Q_9	14262547698921937056113	0.01975	$2 \cdot 251513 \cdot 6086980338053$
Q_{10}	13755004021960592085464	0.01470	$13 \cdot 73 \cdot 11329 \cdot 341363029039$
Q_{11}	14214149085376118983291	0.00507	$3 \cdot 67^2 \cdot 231261154828021$
Q_{12}	15151852662623823374781	0.00874	$3^4 \cdot 5^2 \cdot 21841 \cdot 51162486377$
Q_{13}	14185394352093247029946	0.01247	$2 \cdot 13 \cdot 673 \cdot 1367 \cdot 131526688669$
Q_{14}	12351139031991610954191	0.00568	6286990862744556017
Q_{15}	14601881988167170300659	0.01173	$3 \cdot 1759 \cdot 1907 \cdot 270517847591$
Q_{16}	13603479675779569518553	0.01811	$5 \cdot 7 \cdot 41 \cdot 191 \cdot 8167 \cdot 1732925917$
Q_{17}	13809622636367237837331	0.01598	$2 \cdot 3^4 \cdot 13 \cdot 29 \cdot 37^2 \cdot 197 \cdot 218445341$
Q_{18}	12464197256082744853511	0.01164	$2 \cdot 3 \cdot 11 \cdot 19 \cdot 4790256596280107$

表 2.6: RSA-130 の他の候補多項式のパラメータ

R_i	f_1 の収量		$\alpha(F)$	
	ratio	ranking	α 値	ranking
R_1	2.00	(1)	-3.88	(5)
R_2	1.98	(2)	-3.92	(4)
R_3	1.89	(3)	-3.94	(3)
R_4	1.66	(4)	-4.00	(2)
R_5	1.63	(5)	-4.30	(1)
R_6	1.48	(6)	-3.20	(6)
相関関係		基準		

表 2.7: RSA-130 のさらに他の候補多項式の収量値

R_i	m	χ	c_5
R_1	12429620102099690356862	0.01142	$2^4 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 41 \cdot 12757$
R_2	12429620102099690356861	0.01526	$2^4 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 41 \cdot 12757$
R_3	12429620102099690356863	0.01727	$2^4 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 41 \cdot 12757$
R_4	13451029676646753000757	0.01842	$419 \cdot 5801 \cdot 1688431082621$
R_5	12400786914908592973618	0.01570	$31 \cdot 1361 \cdot 37199 \cdot 40759 \cdot 96329$
R_6	12664454168907537623814	0.01071	$2^3 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 43 \cdot 332309$

表 2.8: RSA-130 のさらに他の候補多項式のパラメータ

第3章　線形代数部

3.1　線形代数部の概要

3.1.1　背景

一般数体ふるい法は、任意に与えられた大きな素因子を持つ合成数 N に対する効率的な素因数分解分解アルゴリズムとして知られている。数体ふるい法の処理では、ふるいによって得られたデータから、有理数体並びに代数体から求められた、N を法として等しい非自明な 2 乗因子を利用することで N を分解する。このアルゴリズムにおいて、ふるいによって出力されたデータから必要な関係式を得る部分では、$GF(2)$ を係数とする行列 M に対して $Mx = 0$ となる x を求める必要がある。この処理部分を線形代数部と呼ぶ。線形代数部で扱う行列は巨大であることに加え、要素が 0 以外となる成分の割合が極端に小さい、いわゆる疎行列であるため、ガウスの消去法等の通常の線形方程式に対する解法では、効率が悪いことが判る。このような行列に対する効率的なアルゴリズムとしては、さらに行列の成分が $GF(2)$ で構成されていることから Block Lanczos 法 [17] が有効であることが知られている。

Block Lanczos 法は、$GF(2)$ 上の対称疎行列で与えられた連立 1 次方程式に対する求解法であり、アルゴリズム内部は大きく分けて、行列とベクトルとの乗算、ベクトル同士の内積、の 2 種類の演算を組み合わせ、Gram–Schmidt の直交化を可能な限り実施していくことによって、非自明解を求めていく。本章は、その高速化に関する方法の一つとして、並列処理の利用に関する考察を行う。

並列処理とは、ネットワーク等で結合した複数の計算機資源を利用して処理を高速化するものであるが、並列処理を効率化する際には、CPU 内部の処理時間に加え、計算機間の通信についても最適化を図る必要がある。本章では Block Lanczos 法を、並列処理ツールとして知られている MPI (Message Passing Interface) を用いて実装し、ネットワークによって密につながった汎用 PC 上で高速演算する方法について述べ、さらに、本実装を Cunningham number 2,1826L の因数である 164 桁の未分解合成数に対して数体ふるいを用いた素因数分解 [1] に適用した結果について

述べる。

3.1.2　数体ふるい法における Block Lanczos 法

Block Lanczos 法は対称行列 A で与えられる線形方程式の解を求めるアルゴリズムである。一方で、数体ふるい法で扱う行列 M は対称行列とは限らないため、Block Lanczos 法を適用するには少し工夫が必要である。

まず、行列 M を $GF(2)$ 上の $n' \times n$ とする。$A = M^T M$ は $n \times n$ の対称行列となる。任意に選んだ y に対して $b = Ay$ とし、y とは異なる y' で、$Ay' = b$ となるものが得られれば $x = y - y'$ が $Ax = 0$ $(x \neq 0)$ の解となることを利用する。具体的な演算方法は以下の通り。

まず t 個の ランダムなベクトル $c_i \in GF(2)^n$ $(i = 1, ..., t)$ を選択する。$c_1, ...c_t$ を並べて、$n \times t$ のベクトル列 C を生成する。なお今後、紛れが無い限りベクトル列をベクトルと呼ぶことにする。t は ブロック長を表す変数であり、理論上は何でも構わないが、実質的にはマシンワード長の倍数 (今回の実装では 128 bit) とする。ベクトル C に対して行列 A を作用させて B を生成する。

$$B = AC$$

Block Lanczos 法によって、次を満たすベクトル X が求められたとする。

$$\dim(AX - B) \ll t$$

上の式に $B = AC$ を代入すると、$\dim(A(X - C)) \ll t$ が成り立つことより、三角化等によって、$X - C$ の部分空間から、$Ax = 0$ を満たす $x \neq 0$ を求めることが出来る。一般には $Ax = 0$ であったとしても $Mx = 0$ となる保証はないが、行列 M の rank が n' に近く、さらに n が n' に対して十分大きければ、$Ax = 0$ の解が $Mx = 0$ の解となる可能性が高くなる。

Block Lanczos 法をアルゴリズム 1 に示す。$n \times t_1$ 行列 W と $n \times t_2$ 行列 W' に対し、$t_1 \times t_2$ 行列 $W^T A W'$ を $(W, W')_A$ と略記する。また二つのベクトル A, B を並べたものを $[A|B]$ と書くことにする。

アルゴリズム 3.1 (Block Lanczos 法)
$GF(2)$ 上対称行列 A に対し、$\dim(AX-B) \ll t$ を満たす X を求める。

1. 初期値の設定。$V_1 = B$ とし U_0, W_0 を共に 0 行列とする。
2. 下記の操作を $i = 1$ から繰り返す。
   (a) W_i を V_i の部分集合で $W_i \supset U_{i-1}$ かつ $\det((W_i, W_i)_A) \neq 0$ を満たす最大のものとする。
   (b) もし $W_i = \emptyset$ であればループを抜ける。
   (c) $U_i = V_i \setminus W_i$ とする。
   (d) $(W_j, W_j)_A{}^{-1}(W_j, [U_i|AW_i])_A$ を $C_{i,j}$ として、次を計算する。

$$V_{i+1}=[U_i|AW_i] + W_iC_{i,i} + W_{i-1}C_{i,i-1} + W_{i-2}C_{i,i-2}$$

3. 終了条件が成立する i に対し、$i = m$ とし、下記を出力する。

$$X = \sum_{j=0}^{m-1} W_j(W_j, W_j)_A{}^{-1}(W_j, B)$$

3.1.3　Block Lanczos 法の計算量

本アルゴリズムにおいて、1 回のループ内で必要な計算は、

1. 行列とベクトルの積 AW_i
2. ベクトルの内積 $(W_i, W_i)_A$, $(W_i, AW_i)_A$, (W_j, B)
3. ベクトルと小行列の積 $W_i(W_j, W_j)_A{}^{-1}$, $W_iC_{i,i}$, $W_{i-1}C_{i,i-1}$, $W_{i-2}C_{i,i-2}$, $W_j(W_j, W_j)_A{}^{-1}(W_j, B)$

である。各部分の計算量を見積もったものを表 3.1 にまとめた。対称行列 $A = M^TM$ の行数を n, 行列 M の各行における 1 の数の個数 (ウェイトと呼ぶ) の平均値を d, ベクトル B のブロック長を t とする。

表 3.1 より 1 回のループにおける計算量はメモリ参照 $O(nd) + O(nt)$ 論理演算 $O(nd) + O(nt^2)$ となる。一方で行列の rank を m (行列の行数、列数の小さい方で押えられる) とした時、Block Lanczos 法が終了するまでのループ回数はおよそ m/t で見積もることができるから、Block Lanczos 法全体の計算量は、メモリ参照 $O(mnd) + O(mnt)$ 論理演算 $O(mnd) + O(mnt^2)$ となる。以下本章では、この Block Lanczos 法で必要な演算部品、特に行列とベクトルの積の演算を、並列分散処理を行い

	メモリ参照	論理演算
行列×ベクトル	$O(nd) + O(nt)$	$O(dn)$
ベクトル内積	$O(nt)$	$O(nt^2)$
ベクトル×小行列	$O(nt)$	$O(nt^2)$

表 3.1: Block Lanczos 法の 1 回のループ処理計算量

て高速化する方法について述べる。

3.2　並列計算機環境におけるデータ通信

本章では、Block Lanczos 法の並列計算機上への実装について扱うが、まず、並列計算機による高速化を図る上で重要と思われる、各ノード上のデータ通信に関する考察を行う。ここでは各ノードで CPU, メモリ, ハードディスク、ならびに計算機ネットワークへのポートをそれぞれ個別に持つ分散メモリ型並列計算機を考える。

並列計算機上でのデータ通信を含むプログラム作成を容易に実現するツールとして、MPI (Message Passing Interface) がある。MPI は、アルゴリズムで必要とされる主要なデータ通信があらかじめ準備されており、TCP/IP 上の通信を行う上での繁雑な手続きを軽減してくれる。MPI には複数のライブラリが存在しているが、有名なところでは LAM, MPICH, SCore 等がある。各ライブラリにはそれぞれ特色があるが、今回は LAM verison 6.5.9 を利用した [12]。

3.2.1　2 ノード間のデータ交換

まず、データ通信の基本である 2 ノード間でのデータ交換について述べる。今回の実装では、全二重通信が可能なネットワークインターフェイス、ならびにスイッチング HUB で構成されたシステムを用いている。この全二重通信の特性を利用すれば 2 ノード間でのデータ交換は同時に行え、理論上はシーケンシャルに行った場合に比べて通信時間が半分になる。(図 1 参照。) 我々の実装では、`MPI_Barrier()` と `MPI_Sendrecv()` を組み合わせて実装した。なお `MPI_Barrier()` は、各ノードで同期を取るための関数であり、通信を開始する前に行い、通信効率を安定させるために重要である。

続いて 2 ノード間のデータ交換を利用して、全ノードでのデータ共有方

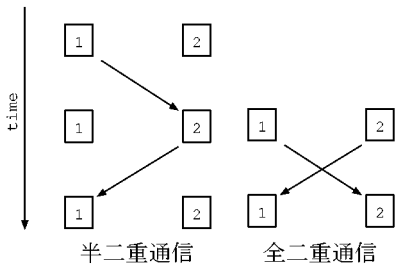
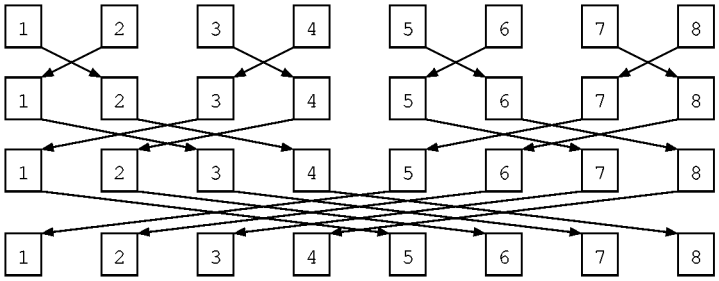


図 3.1: 2 ノード間通信

図 3.2: バタフライ構造



法について述べる。このようなデータ通信には、バタフライ構造 (Butterfly Structure) が有効である。バタフライ構造を実現するには、どの 2 ノード間の通信も独立に行えることが条件となり、ネットワークトポロジーとして完全グラフを構成している必要がある。今回用いた環境では、16 ポートのスイッチング HUB を利用し、いずれの 2 ノード間も独立に全二重通信可能であることを確認している。

なお、MPI ライブラリにはこのような用途のために `MPI_Allgather()` `MPI_Allreduce()` 関数等が用意されている。しかし、標準的な MPI ライブラリにおいてこれらの組み込み関数がネットワークが持つ双方向通信を自動的に認識し、通信時間を最短にしているとは限らないため、今回の実装では前節で述べた方法を使って、これらの機能を実現する関数を新たに記述し、通信量の最適化を図った[1] 。

[1] 実装当初は MPI ライブラリとして MPICH 1.2.5 を利用していたが、MPICH 1.2.5 では、ネットワークが持つ全二重通信の機能を利用できなかった。

3.3　行列とベクトルの積

本節では Block Lanczos 法を用いた時の計算量の多くを占める行列とベクトルの乗算について考察する。各行の平均非零要素数 が d であるような $GF(2)$ 上 $n' \times n$ 疎行列 M と $GF(2)$ 上 n 次元 ベクトル X に対して、$Y = M^T MX$ を考える。

3.3.1　行列のデータ構造

行列がランダムな場合、すべての座標の要素を並べる表現方法が一般的に良く用いられるが、この方法は疎行列の場合には明らかに効率が悪い。今回扱う行列は疎行列であること、さらに成分として 0 あるいは 1 しか持たないことを利用し、非零成分の位置のみを記憶することとする。今回用いた行列の計算機内部での表現形式は、次のようなものである。なお m_i は i 行目の非零要素数であり、$a_{i,j}$ は、i 行目で非零要素の位置を表す。さらにこの表現では行に関する情報は省略している点に注意する。

$$
\begin{aligned}
&m_1, a_{1,1}, ..., a_{1,m_1}, \\
&m_2, a_{2,1}, ..., a_{2,m_2}, \\
&..., \\
&m_n, a_{n,1}, ..., a_{n,m_n}
\end{aligned}
$$

各データ m_i, a_{ij} の格納には、行列の行数列数に依存したデータ構造が必要であるが、今回扱う行列の列数は 2^{32} 未満であると仮定し、各データの格納には各々 32 ビット= 4 バイトを利用すればよいとする。よって、この表現で必要なメモリサイズは、$4n(1+d)$ バイトである。要素を全て並べる方法では $n' \times n$ 行列の表現に、$n'n$ 個のメモリが必要であるのに比べると、$d \ll n'$ である疎行列の場合には、必要となるメモリー量の大幅な削減効果がある。(実際に用いる行列の n, n' d の大きさについては、第 8 節に記述されている行列データの行数、列数ならびに平均ウェイト数を参照。) さらに、詳しくは述べないが、m_i や a_{ij} の大きさに制限がある場合などは、一つの情報を格納するエリアを縮小したり、データ圧縮を用いるなどによって、より少ないメモリで保持できる可能性がある。

3.4　分散計算機環境における行列乗算

まずは 1 CPU 上での演算について考察する。演算 $AX = M^T MX$ には M の各非零要素毎に、ベクトル X への 2 回の連続メモリアクセスと

2 回のランダムメモリアクセス、計 4 回のメモリアクセスが発生し、さらに行列要素へのアクセスを加え、合計 4 回の連続アクセス、2 回のランダムアクセスが必要となる。

続いて、行列とベクトルの積の並列処理を試みる。考えやすくするため、行列の非零成分の密度は均一であると仮定する。また並列計算機のノード数は $p = q^2$ 個 (平方数個) とする。まず行列の分割方法について考察する。各ノード毎で持つ部分行列への分割方法として、次の三種類の表現方法が考えられる。

1. 行で分割
2. 列で分割
3. 行と列で分割

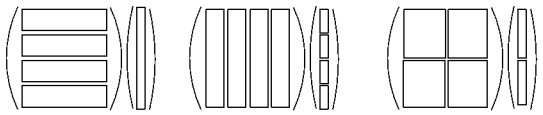
表 3.2 は、これら 3 種類の行列分割法について、行列とベクトルの乗算を並列演算処理するために必要なメモリ量、メモリアクセス回数、論理演算回数、通信時間をまとめたものである。行と列の両方で分割する場合、乗算 MX 1 回を計算する場合は $2n/q + 2n$ ビットのデータを送信する通信時間が必要であるが、一方で M^TMX を繰り返し計算する際には、n 次元ベクトルに引き戻す必要がないため、$2n/q$ ビットの通信時間で演算できることに 加え、さらに行列データを転置行列として使い回すことも可能である。行列が大きくなるに従い、全体に占める通信時間の割合が相当量にのぼることから、我々の実装では行と列で分割する方法を選択した。

この方法の場合、ノード数が 1, 4, 16, 64 である時に必要な各通信時間および行列演算量 (論理演算回数+メモリ参照回数) は次の通り。ただし、これらの値は、全ての 2 ノードが互いに妨げることなく独立に通信できることを仮定している点に注意する。

ノード数	通信	行列演算
1	0	$4nd$
4	n	nd
16	n	$nd/4$
64	$3n/4$	$nd/16$

今回の実装で利用する switching HUB は 16 ポート であり、それ以上のノード数に対する完全グラフネットワークを構築できないため、16 ノードで行列演算を行った。なお通信時間の面では、4 ノードと 16 ノードでは、時間に差はないが、分割された行列のサイズ、ならびに行列演算時間に差が現れる。

図 3.3: 行列の分割の模式図



1. 行で分割　2. 列で分割　3. 行と列で分割

	メモリ量	メモリ参照	論理演算	通信時間
行分割	$dn/p + 2n/p + n$	$3dn/p$	dn/p	$n\lceil \log_2(p) \rceil + n$
列分割	$dn/p + 2n/p + n$	$3dn/p$	dn/p	$n\lceil \log_2(p) \rceil + n$
行列分割	$dn/p + 3n/q$	$3dn/p$	dn/p	$2n/q$

表 3.2: 行列とベクトルの乗算の並列計算量見積もり

3.4.1　行と列で分割した場合の乗算アルゴリズム

$n \times n$ 行列 M を $p = q^2$ 個の $n'/q \times n/q$ 部分行列 U_{ij} $(i, j = 0, ..., q-1)$ に分割し各ノードに保持する。入力ベクトル X は、上位から始めて $\lceil n/q \rceil / q$ 成分づつ切り出したベクトルを $X_0, ..., X_{q-1}$ とした時、各 X_i をさらに q 個に均等分割したベクトル X_{ij} が $qi + j$ 番目のノードに格納されているものとする。出力ベクトル Y についても同様に各ノードに格納させるものとする。

アルゴリズム 3.2 (行と列で分割)

1. 各 $i = 0, ..., q-1$ について q 個のノード qi, ..., $qi+q-1$ が各々が持つベクトルをバタフライ構造を利用して集め、各ノードで X_i を共有する。
2. 各ノードで部分行列 U_{ij} とベクトル X_i の乗算を実行し n'/q 次元ベクトル $Z_{ij} = U_{ij}X_i$ を算出する。
3. 計算された Z_{ij} $(i = 0, ..., q-1)$ をノード $j, q+j$ ..., $(q-1)q+j$ 間で、バタフライ構造によるデータ交換をしながら排他的論理和を取り、$Z_j = \oplus_{i=0}^{q-1} Z_{ij}$ を共有する。
4. 各ノードで部分行列 U_{ij}^T とベクトル Z_j の乗算を実行し n/q 次元ベクトル $Y_{ij} = U_{ij}^T Z_j$ を算出する。
5. q 個のノード qi, ..., $qi+q-1$ で各々計算された Y_{ij} を q 分割し、各々を最終的に格納すべきノードへと、バタフライ構造を利用して分配しながら排他的論理和を取る。

3.5　ベクトル演算

Block Lanczos 法では、t ビット (今回の実装では 128 bit) 成分を 1 word として持つ n 次元のベクトルに対する内積、及びベクトルと t 次正方行列 (以降、小行列と呼ぶ) の積を Lanczos のメインループ内で複数回演算する必要がある。内積の計算は 1 word に対して t 回の 1 bit 巡回シフトを繰り返し、相対するベクトルの要素との排他的論理和をとることで、実現される。

アルゴリズム 3.3 (ベクトルの内積)
t ビットを成分とする n 次元ベクトル X と Y の内積 $smat = X^T Y$ (t 次正方行列) を求める

1. t ビット t 次元ベクトル w の成分を 0 に初期化
2. i を 1 から n まで次を繰り返す
   (a) $a = X[i], b = Y[i]$
   (b) $j = 0, ..., t-1$ について
   $w[j] = w[j] \oplus ((a \lll j) \& b)$
3. 各 $i, j = 0, ..., t-1$ について
   $mat[i][j] = w[(-i-j-2)_t][j]$

但し、$w[j]$ は行列 w の j 行目を表し、各ベクトル並びに行列は第 0 成分、第 (0,0) 成分からはじまるものとする。また $()_t$ は t を法とした値とする。ベクトルと小行列との積も、行列の内積とほぼ同じである。

アルゴリズム 3.4 (ベクトルと小行列の積)
t ビットを成分とする n 次元ベクトル X と t 次正方行列 $smat$ との積 $Y = X \cdot smat$ を求める。

1. 各 $i, j = 0, ..., t-1$ について
   $w[i][j] = smat[(-i-j-2)_t][j]$
2. ベクトル Y の成分を 0 に初期化
3. i を 1 から n まで次を繰り返す
   (a) $a = X[i]$
   (b) $j = 0, ..., t-1$ について
   $Y[i] = Y[i] \oplus ((a \lll j) \& w[j])$

さて、アルゴリズム 3, 4 では 1 ワード長の要素に対する 1 ビット巡回シフト演算が支配的であり、普通に実装したのでは、Block Lanczos アルゴリズム全体に対するボトルネックとなりうる。そこで、この 1 ビット巡回シフトを高速化する方法を考える。アイデアは次の通りである。

ここでは、Pentium 4 の 128 ビット XMM レジスタおよび SSE2 命令を利用する。128 ビットレジスタ a にはターゲットとなる値が、また 128 ビットレジスタ b には a を 64 ビット単位で左右入れ替えた値が入っているものとする。a および b を、命令 `psrlq` で 63 ビット右シフトさせると、0 ビット目および 64 ビット目には、元の値の 63 ビット目および 127 ビット目が配置されている。ここで、`psrlq` 命令は 128 ビットデー

```
; word_rol1(xmm1) and word_rol1(xmm5)
; assumption: xmm5 = xmm1 <<< 64
; MSB of xmm2 and xmm6 is cleared
por xmm2, xmm1   ; MSB(xmm2) <- MSB(xmm1)
psrlq xmm2, 63
por xmm6, xmm5   ; MSB(xmm6) <- MSB(xmm5)
psrlq xmm6, 63
paddq xmm1, xmm1 ; xmm1 <<= 1
paddq xmm5, xmm5 ; xmm5 <<= 1
paddq xmm1, xmm6 ; fill LSB with crossing
por xmm5, xmm2
```

表 3.3: 128 ビットレジスタの 1 ビットローテーション

タとしてのシフトではなく、64 ビットデータとして左右 2 つに区切り、その区切り単位でシフトさせることに注意せよ。その結果が、それぞれ 128 ビットレジスタ c, d に格納されたとすれば、a, b を paddq 命令で 1 ビット左シフトさせ (ここでも、64 ビット単位でのシフトであることに注意)、それに d, c を por することで、128 ビットデータのローテーション結果が一度に二つ得られることになる。このようなアイデアで実装したところ、ローテーション一つ当たり 5 〜 6 cycles で求めることができた。本アルゴリズムを実現するアセンブリコードを表 3.3 に記す。

3.5.1　ベクトルの内積、ベクトルと小行列の積の並列化

ベクトル X, Y の内積を並列計算機上で実装する方法については、ノード数を p とした場合、n 次元ベクトルを各ノードに $\lceil n/p \rceil$ 次元ずつ等分に分配しておき、あとは各ノードで持つベクトルで計算した内積結果を排他的論理和すればよい。内積結果は t 次正方行列であり、n 次元ベクトルや行列と比較すると遥かに小さく、通信時間はほぼ無視できる。ベクトルと小行列の積も同様である。

3.6　Block Lanczos 計算結果

前節までに述べた Block Lanczos アルゴリズムを、2,1826L に含まれる 164 桁の未分解合成数に対して実施した一般数体ふるい法を用いた素

因数分解において、線形代数演算に適用した結果を述べる。合成数の分解に関する概要については [1] を参照されたい。今回、線形代数部で利用した計算機環境は次の通り。
Northwood Pentium 4 (2.53GHz 等を含む 16 台)
1GB main memory, Gigabit Ethernet
LAM-6.5.9 (MPI Library)

また、ふるいならびに filtering によって生成された行列のデータは、次の通りである。

ファイルサイズ	4.661Gbytes
行数	7501898
列数	7500801
最大ウェイト数	1279
平均ウェイト数	165.78

この行列を $4 \times 4 = 16$ 個の部分行列に分割した。なお、分割には各部分行列のファイルサイズがなるべく均一になるように工夫している。重みの分配についての具体的な話は、数体ふるい法で線形代数部があつかう行列を生成する filtering パートの具体的な出力と深い関係があり、簡単には記述できないため、本章では省略する。各分割行列 M_{ij} のファイルサイズ $sizeof(M_{ij})$ は次の通りとなった。単位は Byte である。

$$sizeof(M_{ij}) = \begin{pmatrix} 79422532 & 79513089 & 79681767 & 79535670 \\ 79532686 & 79617818 & 79812467 & 79636906 \\ 79479662 & 79575870 & 79752208 & 79597753 \\ 79514674 & 79595534 & 79763598 & 79611965 \end{pmatrix}$$

今回の計算は数日に渡って行われることから、システムエラー等により途中で計算が止まる可能性も考慮して、100 ループ毎に 1 回、内部データをファイルに保存し、途中からでも再開できるようにした。本行列に対し、Block Lanczos アルゴリズムを適用した結果、12 日 4 時間 18 分かけて、128 次元の解が得られた。各演算時間の内訳は次の通りである。

総ループ数	58952 回
ベクトル演算	47 時間 24 分
行列演算	171 時間 54 分
通信時間	72 時間 12 分
その他	48 分
合計	292 時間 18 分

第4章　代数的数の平方根

4.1　数体篩法における代数的数の平方根

合成数 N に対して $A^2 - B^2 = 0 \mod N$ を満たす自然数 $A \neq \pm B \mod N$ を求めることによって、因子 $\mathrm{GCD}(A-B, N)$ を得られるが、数体篩法は、この A, B を代数体の演算を経由することで効率的に求める方法である。

合成数 N と $f(M) = 0 \mod N$ を満たす自然数 M と多項式 $f(x)$、を用意する。ここで、$f(x)$ は 非 monic な $\mathbb{Z}$ 上既約多項式である。$f(x)$ の根 θ を添加した $\mathbb{Q}$ 上の代数体を $\mathbb{K} = \mathbb{Q}(\theta)$ とする。$\phi(\alpha) = M \mod N$ で定まる準同型を $\phi : K \to \mathbb{Z}/N\mathbb{Z}$ とする。一般数体ふるい法を用いた素因数分解アルゴリズムでは、ふるい演算、ならびに線形代数演算から、数多くの一次多項式 $\{a + bx | a, b \in \mathbb{Z}\}$ の組合せによって、$\prod_{i \in S}(a_i + b_i M)$ が、$\mathbb{Z}/N\mathbb{Z}$ 上で平方数となり、さらに代数的数 $\prod_{i \in S}(a_i + b_i \theta)$ が生成する $\mathbb{K}$ 上の単項イデアルが、平方イデアルとなるようにできる。(S は有限集合。) このような一次式の集合 $\{a_i + b_i x | i \in S\}$ を relation と呼ぶ。更に次節で述べる方法によって、$\prod_{i \in S}(a_i + b_i \theta)$ が $\mathbb{K}$ 上の平方数となるような relation の組合せを求める事が出来る。この時次の式が成り立つ。

$$\left[\phi\left(\sqrt{\prod_{i \in S}(a_i + b_i\theta)}\right)\right]^2 = \left[\sqrt{\prod_{i \in S}(a_i + b_i M)}\right]^2 \mod N$$

この両辺の計算によって、合成数 N の分解をする。右辺の平方根の計算は、$\mathbb{Z}/N\mathbb{Z}$ 上の平方根計算は直接的に求める事が出来るが、代数的数の平方根については、数論に関する様々な知識を動員して求めなければならない。本文章は本章では、一般数体ふるい法を用いる素因数分解アルゴリズムにおける最終部分である、上式の左辺に含まれる代数的平方根を効率的に求める方法について述べるものである。

4.1.1　平方イデアルから平方数へ

篩演算部、線形代数部によって、relation の積 $\prod_{i\in S}(a_i+b_i\theta)$ が生成する単項イデアルが、平方イデアルとなるものを求める事ができる。一般数体篩法を用いた素因数分解には、代数体の要素として平方数であるものを得る必要があるが、一般的には代数的数を生成元とする単項イデアルが平方イデアルであったとしても、その代数的数そのものが平方数であるとは限らない。そこで、何らかの処理を行なって、平方イデアルとなる単項イデアルから平方数を導く必要がある。代数的数 α が平方数であるためには、任意の素イデアル $\mathfrak{p}$ に対して $\alpha \mod \mathfrak{p}$ が平方数である必要があるが、代数体の次数 d に対し、α を割らない $2d$ 個程度の素イデアル $\mathfrak{p}$ を法として平方数であれば、α も平方数となっている可能性が十分高いことが知られている [11]。この方法を用いて、$\prod_{i\in S}(a_i\theta+b_i)$ が代数的数として平方数となる組合せを見つけることが出来る。

命題 4.1 $\alpha(\theta)$ が一次の素イデアル $\mathfrak{p}=(p,\theta-s)$ を法として平方数であることと、$\alpha(s)$ が p を法として平方数であることは同値

定義 4.2 ((Legendre-Jakobi-Kronecker Symbol))
整数 a および素数 p に対して平方剰余記号 $\left(\frac{a}{p}\right)$ を次で定義する。
$a=0$ のとき $\left(\frac{a}{p}\right)=0$、$a$ が p を法として平方数であるとき $\left(\frac{a}{p}\right)=1$、平方数でない時 $\left(\frac{a}{p}\right)=-1$。

次の式より、平方剰余記号を具体的に計算することが可能となる。

$$\left(\frac{ab}{p}\right)=\left(\frac{a}{p}\right)\left(\frac{b}{p}\right),\quad \left(\frac{a}{p}\right)=a^{(p-1)/2} \mod p$$

さらに平方剰余記号を高速に計算する方法が知られているが、その方法については、[5] を参照。これらの命題から、素イデアル $\mathfrak{p}=(p,\theta-s)$ 上で $\prod_{i\in S}(a_i\theta+b_i)$ の平方剰余記号を求めるには、各 $(a_i\theta+b_i)$ の素イデアルに対する平方剰余記号の値を求め、その各平方剰余の積を求めれば良い。

以上を用いて単項イデアルが平方イデアルとなる代数的数から、平方数である代数的数を求められる。具体的には Block Lanczos アルゴリズムで求められる複数個の解から、用意された全ての一次素イデアル上での平方剰余記号が 1 となる組合せを求めればよい。実際の計算では、平方剰余記号の値を -1 の巾として与え、その指数を並べれば、高々 Lanczos 解の個数×用意された素イデアルの $GF(2)$ 上の行列 (実際には 32×32 程度) で与えられた線形関係式の非自明解を求めることと等しくなる。こ

の解を求めるのは簡単であるから、以上によって一次式 $\prod_{i\in S}(a_i\theta+b_i)$ の組合せで、代数的数として平方数となるものを求めることが出来る。

この節以降、一次式の積として与えられる代数的数 $\prod_{i\in S}(a_i\theta+b_i)$ は、代数的平方数であるとする。

4.1.2　代数体上の計算

ここでは、今後使用する記号の定義を行う。整数係数 d 次既約多項式を考える。

$$f(x)=c_dx^d+\cdot+c_1x+c_0$$

$f(x)$ の複素根を $\theta_1,...,\theta_d$ とし、根の一つを θ とする。θ を添加した $\mathbb{Q}$ 上の拡大体を代数体とよび $\mathbb{K}=\mathbb{Q}(\theta)$ で表す。$f(x)$ を $\mathbb{K}$ の定義多項式とよぶ。$\sigma_j(\theta)=\theta_j$ $(j=1,...,d)$ から得られる準同型写像を $\sigma_j:K\to\mathbb{C}$ とする。f の多項式としての判別式を次で計算される値とする。$\mathrm{disc}(f)=res_x(f(x),f'(x))$ ただし、$res_x(f,g)$ は多項式 f, g の変数 x に関する終結式、$f'(x)$ は、$f(x)$ の x に関する微分とする。定義多項式 $f(x)$ を monic 化したもの $\tilde{f}(x)=c_d^{d-1}f(x/c_d)$ の根を $\tilde{\theta}$ とすると $\mathbb{K}=\mathbb{Q}(\tilde{\theta})$ である。代数的数 $\alpha\in K$ に対し、$N(\alpha)=\prod_{j=1}^{d}\sigma_j(\alpha)$ を α のノルムと言う。$\psi_x(\theta)=x$ から得られる準同型写像を $\psi_x:K\to\mathbb{Q}[x]$ とすると、$N(\alpha)$ は次で計算される。

$$N(\alpha)=res_x(f(x),\psi_x(\alpha))/{c_d}^e$$

ただし e は $\psi_x(\alpha)$ の x に関する次数である。特に、一次要素 $a+b\theta$ のノルムは次の式で計算される。

$$N(a+b\theta)=(-b)^df(-a/b)/c_d$$

4.1.3　整数環の整数基底計算

代数体 $\mathbb{K}$ の要素で、最小多項式の最高次係数が 1 となる要素全体を整数環 (Integral Domain) と呼び $\mathcal{O}$ と書く。代数体 $\mathbb{K}$ の整数環 O は d 個の要素を基底とする $\mathbb{Z}$ 上の加群として表現される。代数体上の平方根を計算するに当たって、整数環 $\mathcal{O}$ の $\mathbb{Z}$ 上の基底 (Integral Basis) を求める必要があるが、その手段としては、Zassenhaus の Round2 アルゴリズム ([5] Algorithm 6.1.8) が知られている。このアルゴリズムには $\tilde{f}(x)$ の判別式 $\mathrm{disc}(\tilde{f})$ に対し、二次因子を持たない s を用いて、$\mathrm{disc}(\tilde{f})=st^2$ となる因数分解が得られ、かつ t の素因数分解が求められる必要がある。こ

の表現を得ることは、一般的には難しい問題であり、一般的な合成数に対する素因数分解アルゴリズムを用いた場合には、目標としている合成数 N の素因数分解に、同程度の大きさを持つ別の素因数分解を行わなければならないという矛盾が含まれることになる。ただし、実際に一般数体篩法による素因数分解に用いる多項式の場合、判別式 $\mathrm{disc}(\tilde{f})$ の 2 次以上の素因子は、ほとんどが $f(x)$ の最高次係数 c_d の素因子であり、それ以外の値もあまり巨大な 2 次因子が含まれる可能性は少ない。よって、$\mathrm{disc}(\tilde{f}) = st^2$ には厳密な素因数分解ではなく、rho 法、楕円曲線法等で、小さい素因子のみを取り出した疑似素因数分解で十分な場合が多い。この疑似素因数分解から得られる基底による $\mathbb{Z}$ 加群 $\mathcal{R}$ は、厳密な意味では整数環 $\mathcal{O}$ とは異なるが、ほとんどの場合、$\mathcal{O}$ と同じ性質を持つことが期待できる。

整数環の基底は、代数体 $\mathbb{K}$ に対して一回だけ計算されればよい。この計算は平方根の演算とは独立かつ事前に計算しておけばよく、しかもほとんどの場合、疑似素因数分解さえ得られれば、汎用数式処理システムを用いても、比較的短時間で求められる。そこで今回の代数的平方根アルゴリズムの実装では、整数環の基底を求めるアルゴリズムを改めて C 言語等で実装することはせず、一般的な数式処理ライブラリとして有名な PARI-GP version 2.1.5 [2] に用意されている関数を用いて求められた整数基底を利用した。PARI-GP による整数基底の計算法は次の通りである。代数体 $\mathbb{K}$ の判別式 $\mathrm{disc}(\mathbb{K})$ の計算法と併せて記述する。この演算は数秒で終了する。

アルゴリズム 4.3 (PARI-GP による整数基底計算)

```
poly = “定義多項式”
d = poldegree(poly)                    ; 多項式次数
lc = polcoeff(poly,d)                  ; 最高次係数
mpoly = subst(poly,x,x/lc)*lc^(d-1)); monic 化多項式
pdisc = poldisc(mpoly)                 ; 多項式の判別式
pl = factor(pdisc,2^31-1)              ; 疑似素因数分解
disc = nfdisc(poly,0,pl)               ; 代数体 K の判別式
basis = nfbasis(poly,0,pl)             ; 整数環 O の基底
```

例 4.4 (PARI-GP による計算例 : $N = 5 \cdot 3^{45} + 2$ の場合)

合成数 $N = 5 \cdot 3^{45} + 2 = 14771563532754168493217$ を例として、数体篩法を用いた因数分解における代数体のパラメータ計算例を示す。

$$f(x) = 5x^5 + 2, M = 19683$$

とおけば $f(M) = N$ である。以下は、この $f(x) = 0$ の根で定まる数体の各種パラメータ計算である。

```
colsole% gp -q
? poly=5 * x^5 + 2
%1 = 5*x^5 + 2
? d = poldegree(poly)
%2 = 5
? lc = polcoeff(poly,d)
%3 = 5
? pdisc=poldisc(subst(poly,x,x/lc)*lc^(d-1))
%4 = 7629394531250000
? pl = factor(pdisc,2^31-1)
%5 =
[2 4]
[5 21]
? disc = nfdisc(poly,0,pl)
%6 = 31250000
? basis = nfbasis(poly,0,pl)
%7 = [1, 5*x, 5*x^2, 5*x^3, 5*x^4]
```

4.1.4　分数イデアルと演算

代数体 $\mathbb{K}$ の整数環 $\mathcal{O}$ の分数イデアル全体を $\mathcal{I}$ とする。任意の分数イデアル $I \in \mathcal{I}$ は、$\mathbb{K}$ の有限個の要素 $x_1, ..., x_m$ を生成元とする $\mathbb{Z}$ 加群として記述できる。この時、$I = \langle x_1, ..., x_m \rangle$ と書く。

$\mathcal{O}$ の任意のイデアルは整数環 $\mathcal{O}$ の基底 $\omega_1, ..., \omega_d$ に対し、整数係数の d 次正方行列として表現できる。イデアル $I = \langle a_1, ..., a_d \rangle$, $J = \langle b_1, ..., b_d \rangle$ の乗算は、$I \times J = \langle \{a_i b_j | i, j \in \{1, ..., d\}\} \rangle$ で求められる。行列表現されたイデアルについては、各生成元を整数基底を用いて代数的数に戻した上で、定義通り IJ の生成元を求め、この d^2 個の生成元からなるイデアルに対し、Hermit Normal Form (後述) を計算し、d 次正方行列に変換すればよい。

任意の $\mathcal{O}$ の素イデアル $\mathfrak{p}$ に対し、$v_{\mathfrak{p}}(I)$ を分数イデアル $I \in \mathcal{I}$ の $\mathfrak{p}$-進付値とする。分数イデアル I が平方であるとは、$J \in \mathcal{I}$ が存在し $J^2 = I$ を満たす時を言う。このような J は唯一であり $\sqrt{I}$ で記す。$I = \mathfrak{p}_1^{v_1} \cdots \mathfrak{p}_m^{v_m}$

を I の素イデアル分解とした時、I が平方である事と全ての v_i が偶数であることは同値であり、さらに $\sqrt{I} = \mathfrak{p}_1^{v_1/2} \cdots \mathfrak{p}_m^{v_m/2}$ である。$a \in K$ が平方数であれば、単項イデアル $\langle a \rangle$ は平方イデアルである。

4.1.5　Hermit Normal Form

イデアル $I = \langle a_1, ..., a_t \rangle$ の各生成元 a_i に対して、整数基底 $\omega_1, ..., \omega_d$ の各々の元と ω_j との積 $a_i\omega_j$, $(1 \leq i \leq t)$, $(\leq j \leq d)$ を考え、各 $a_i\omega_j$ を w_k を基底として $d \times dt$ 行列表示したものを M とする。Hermit Normal Form (以後 HNF と略す) の定義は次の通りである。

定義 4.5 $m \times n$ 整数行列 M が Hermit Normal Form (HNF) であるとは、$\{r+1, ..., n\}$ から $\{1, ..., m\}$ への強増加関数 f が存在し、次を満たす時を言う。

(1) $r+1 \leq j \leq n$ に対し $m_{f(j),j} \geq 1$
　　$i > f(j)$ の時は $m_{i,j} = 0$
　　$k < j$ の時は $0 \leq m_{f(k),j} < m_{f(k),k}$

(2) M の 1 列目から r 列目までは 0

一般に与えられた基底ベクトル列で与えられる格子に対し、HNF を求めるアルゴリズムは以下の通りである。

アルゴリズム 4.6 ((2.4.4 in [5])Hermit Normal Form)
n 個の m 次元ベクトル A_i $(i = 1, ..., n)$ の成分を並べた $m \times n$ 行列を $(a_{i,j})$ とする。Hermit Normal Form $W = (w_{i,j})$ を求める。

(1) $i = m, j = n, k = n$
　if $m \leq n$ then $l = 1$
　if $m > n$ then $l = m - n + 1$

(2) if $j = 1$ goto step (4)

(3) $j = j - 1$
　if $a_{i,j} = 0$ goto step (2)
　compute (u, v, d) such that
　　$ua_{i,k} + va_{i,j} = d = gcd(a_{i,k}, a_{i,j})$
　　(Extended Euclid's algorithm)
　$B = uA_k + vA_j$
　$A_j = (a_{i,k}/d)A_j - (a_{i,j}/d)A_k$
　$A_k = B$
　goto step (2)

(4) $b = a_{i,k}$
　if $b < 0$
　　$A_k = -A_k, b = -b$
　if $b = 0$
　　$k = k + 1$
　　goto step (5) else
　　for $j = k + 1$ to n
　　　$q = \lfloor a_{i,j}/b \rfloor$
　　　$A_j = A_j - qA_k$

(5) if $i = l$
　　for $j = 1$ to $n - k + 1$
　　　$W_j = A_{j+k-1}$
　　terminate the algorithm
　else
　　$i = i - 1$, $k = k - 1$, $j = k$
　　goto step (2)

$\lfloor a \rfloor$ は a を超えない最大の整数である。($a < 0$ の場合の計算には注意が

必要)

4.1.6　LLL 簡約アルゴリズム

LLL 簡約基底計算アルゴリズムは、与えられた格子の基底から、格子に含まれる短いベクトルを求める方法として考えられ、現在まで、さまざまな応用や演算の効率化、拡張等が研究されている。LLL 簡約基底の定義は次の通りである。

定義 4.7 一次独立な整数係数 n 次元ベクトル $b_1, ..., b_n$ に対し、Gram-Schmit の直交化により得られる直交基底を $b_1^*, ..., b_n^*$ とする。$i \neq j$ に対し

$$\mu_{i,j} = \frac{b_i \cdot b_j^*}{b_j^* \cdot b_j^*}$$

とおく。$b_1, ..., b_n$ が LLL 簡約基底 (LLL reduced basis) であるとは、$1 \leq j < i \leq n$ に対し

$$|\mu_{i,j}| \leq \frac{1}{2}$$

かつ $1 < i \leq n$ に対し

$$|b_i^*|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) |b_{i,i-1}^2|$$

が成り立つことを言う。

LLL 簡約基底の第一要素は格子に含まれる短ベクトルとみなされる。LLL 簡約基底を求める操作を LLL 簡約 (LLL reduction) と呼ぶ。整イデアル I の HNF は、向きがほぼ同じもので構成され、大きな数値が次数の低い項にあたる係数にあつまっているという、極端に偏ったベクトルが並べられた行列である。このような行列に対し、倍精度の浮動小数による近似を用いてベクトルの直交化を計算すると、計算途上での丸め誤差の蓄積が無視できなくなり、正しい LLL 簡約基底が得られないことがある。一つの解決法としては LLL 簡約アルゴリズムを複数回適用し、収束するまで繰り返すという方法があるが、これは極めて効率が悪い。今回の LLL 簡約実装では、出来るだけ誤差を含まない形で加減乗除演算を進めることが望ましく、LLL 簡約の直交基底演算には、直交化基底の計算にも任意多倍長整数演算を用いた Integral LLL Algorithm が有効であると考えた。Integral LLL Algorithm は、直交基底の計算に浮動小数を用いる一般的な LLL Algorithm 実装と比較すると、全ての計算を多倍長整数演算で計算するため 1 回の計算で比較した場合には、処理に時間が

かかるものの、演算結果に対する丸め誤差が発生しないため、出力は必ず LLL 簡約基底となり、LLL Algorithm を繰り返し適用する等の特別な処理を行う必要はなく、結果としては、浮動小数を用いた計算よりも、遥かに効率的であった。今回の実装では Integral LLL Algorithm を用いて LLL 簡約基底を計算した。

アルゴリズム 4.8 (Integral LLL Algorithm) 一次独立な整数係数 n 次元ベクトル $b_1, ...b_n$ を LLL 簡約基底に変換する。

1. $k = 2, k_{max} = 1, d_0 = 1, d_1 = b_1 \cdot b_1$
2. while $k < n$ do
    if $k > k_{max}$
        for $j = 1$ to k
            $u = b_k \cdot b_j$
            for $i = 1$ to j
                $u = \frac{d_i u - \lambda_{k,i}\lambda_{j,i}}{d_{i-1}} (\in \mathbb{Z})$
            if $j < k$ then $\lambda_{k,j} = u$
            if $j = k$ then $d_k = u$
        if $d_k = 0$ then output an error and terminate
    $redi(k, k-1)$
    while $4d_k d_{k-2} < |3d_{k-1}^2 - \lambda_{k,k-1}^2|$
        $swapi(k)$
        $k = \max(2, k-1)$
        $redi(k, k-1)$
    for $l = k-2$ down to 1
        $redi(k, l)$
3. output $b_1, ..., b_n$

Sub-Algorithm $redi(k, l)$

if $|2\lambda_{k,l}| \leq |d_l|$ then terminate the sub-algorithm
$q = \lfloor \frac{2|\lambda_{k,l}| + |d_l|}{|2d_l|} \rfloor$ (= the integer nearlest to $|\lambda_{k,l}/d_l|$)
$b_k = b_k - qb_l$
$\lambda_{k,l} = \lambda_{k,l} - qd_l$
for $i = 1$ to $l-1$
    $\lambda_{k,i} = \lambda_{k,i} - q\lambda_{l,i}$

Sub-Algorithm *swapi*(k)

exchange b_k and b_{k-1}
if $k > 1$
　　for $j = 1$ to $k - 2$
　　　　exchange $\lambda_{k,j}$ and $\lambda_{k-1,j}$
$\lambda = \lambda_{k,k-1}$
$B = (d_{k-2}d_k + \lambda^2)/d_{k-1}$
for $i = k + 1$ to k_{max}
　　$t = \lambda_{i,k}$
　　$\lambda_{i,k} = \lfloor (d_k\lambda_{i,k-1})/d_{k-1} \rfloor$
　　$\lambda_{i,k-1} = (Bt + \lambda\lambda_{i,k})/d_k$
$d_{k-1} = B$

4.2　代数的数の平方根アルゴリズム – Montgomery, Nguyen の方法 –

代数体 $\mathbb{K}$ の要素で θ に関して一次式の積の形で与えられた平方数

$$\gamma = \prod_{i \in S}(a_i + b_i\theta)$$

の平方根を求める方法として、Montgomery, Nguyen によるアルゴリズムが知られている。本章以降では、このアルゴリズムの実装に関する詳細ならびに、実行結果について述べる。

4.2.1　アルゴリズムの概要

Montgomery, Ngyuyen による代数的数の平方根を求めるアルゴリズムは以下のような流れで行われる。

アルゴリズム 4.9 (Montgomery,Nguyen のアルゴリズム)

1. γ を二つの積 $\gamma = \gamma_{nm}\gamma_{dn}$ に分ける。γ_{mn}/γ_{dn} を改めて γ とおく。(単純化)
2. $\sqrt{\langle\gamma\rangle}$ を求める。(イデアル平方根)
3. $\sqrt{\gamma}$ の近似 $\mu = \prod_j \delta_j^{\pm 1}$ を LLL アルゴリズムを用いて求める。(イデアル近似)
4. $\theta = \gamma/\mu^2$ に対し $\sqrt{\theta}$ を Broute-force method と Chinese Remainder Theorem を用いて求める。(誤差の計算)
5. $\mu\sqrt{\theta}\gamma_{dn}$ を出力する。

本アルゴリズムを実行する上で仮定される条件は次の通り。

1. 定義多項式 $f(x)$ の判別式に関する square free 分解が得られている
2. 対象とする代数的数は 1 次式 $a_i + b_i\theta$ の積の形をした平方数である
3. 各 relation $a_i + b_i\theta$ に関するノルムの素因数分解が得られている

4.3　素イデアル分解

4.3.1　代数体 $\mathbb{K}$ の $\mathbb{Z}$ 加群 $\mathcal{A}$

環 $\mathcal{A} = \mathbb{Z}[\theta] \cap \mathbb{Z}[1/\theta]$ を考える。$\beta_0, ..., \beta_{d-2}$ を次のように定義する。

$$\begin{aligned}
\beta_0 &= c_d\theta^{d-1} + c_{d-1}\theta^{d-2} + \cdots + c_2\theta + c_1 \\
\beta_1 &= c_d\theta^{d-2} + c_{d-1}\theta^{d-3} + \cdots + c_2 \\
&\vdots \\
\beta_{d-2} &= c_d\theta + c_{d-1}
\end{aligned}$$

これを用いると、$\mathcal{A}$ は、$1, \beta_0, ..., \beta_{d-2}$ を基底とする $\mathbb{Z}$ 加群として表現できる。

$$\mathcal{A} = \mathbb{Z} + \beta_0\mathbb{Z} + \cdots + \beta_{d-2}\mathbb{Z}$$

代数的数 $x \in \mathbb{K}$ 並びに $\mathcal{A}$ の素イデアル $\mathfrak{p}$ ならびに $\mathfrak{p}$ のノルム $\mathcal{N}(\mathfrak{p})$ に対して定まる値 $l_{\mathfrak{p},\mathcal{A}}(x) \in \mathbb{Z}$ を $\prod_{\mathfrak{p}} \mathcal{N}(\mathfrak{p})^{l_{\mathfrak{p},\mathcal{A}}(x)} = |\mathcal{N}_{\mathbb{K}}(x)|$ となる値として定義する。但し $l_{\mathfrak{p},\mathcal{A}}(x)$ は有限個の $\mathfrak{p}$ を除き 0 である。

各素数 p に対して $f(s) = 0 \mod p$ を満たす 0 以上 p 未満の整数 s、の集合を $R(p)$ とする。ただし $p|c_d$ の時は、$R(p)$ には ∞ が含まれているとする。$\mathcal{A}$ 上の任意の一次素イデアルは、素数 p と $s \in R(p)$ の組を用い、$(p, \theta - s)$ ($s \neq \infty$ の時)、あるいは $(p, 1/\theta)$ ($s = \infty$ の時) と表現できる。

p を素数、s を $R(p)$ の要素とする。$a+bs=0 \mod p$ の時、$e_{p,s}(a+b\theta)$ を次で定義する。

$$e_{p,s}(a+b\theta) = v_p(c_d \mathcal{N}(a+b\theta))$$

ただし、$p|b$ の時は、$s=\infty$ とする。それ以外の p, s の組みに対しては、$e_{p,s}(a+b\theta)=0$ とする。

4.3.2　整数環 $\mathcal{O}$ 上の素イデアル分解

relation $a+b\theta$ のノルム $\mathcal{N}(a+b\theta)$ は、篩演算部より、比較的小さな素数で分解できるようなものとなっている事が保証され、さらにその具体的な素因数分解の形は判っていると仮定できる。この時分数イデアル $\langle a+b\theta\rangle$ の $\mathcal{A}$ 上の素イデアル分解は、$F(a,b)=c_dN(a+b\theta)=(-b)^d f(-a/b)$ の素因数分解から一意に得られる。$v_p(n)$ を整数 n の p 進付値とする。この時、単項分数イデアル $\langle a+b\theta\rangle$ の素イデアル $\mathfrak{p}=(p,\theta-s)$ 上の指数 $l_{\mathfrak{p},\mathcal{A}}$ は次で計算される。

$$l_{\mathfrak{p},\mathcal{A}} = \begin{cases} e_{p,r}(a+b\theta) & \text{if } b \neq 0 \mod p \\ e_{p,\infty}(a+b\theta) - v_p(c_d) & \text{if } b = 0 \mod p \end{cases}$$

これによって、$a+b\theta$ で生成される $\mathcal{A}$ の単項分数イデアルの素イデアル分解が求められた。一方、代数的平方根で必要とされる素イデアル分解は、整数環 $\mathcal{O}$ 上の分解であるが、$\mathcal{A}$ 上の素イデアルと $\mathcal{O}$ 上の素イデアルは異なるため、上記で求められる $\mathcal{A}$ における素イデアル分解を利用するには条件が必要である。

ここで、指数 $[\mathcal{O}:\mathcal{A}]$ を割る素数 p を exceptional prime (例外素数) と定義し、それ以外の素数を normal prime と定義する。指数 $[\mathcal{O}:\mathcal{A}]$ は、次のように具体的に計算できる。

$$[\mathcal{O}:\mathcal{A}] = \sqrt{\frac{\mathrm{disc}(\mathcal{A})}{\mathrm{disc}(\mathcal{O})}} = \sqrt{\frac{|\mathrm{disc}_x(f(x))|}{\mathrm{disc}(\mathbb{K})}}$$

ただし、$\mathrm{disc}_x(f(x))$ は代数体 $\mathbb{K}$ の定義多項式 $f(x)$ の判別式、$\mathrm{disc}(\mathbb{K})$ は、整数環 $\mathcal{O}$ と共に求められる $\mathbb{K}$ の判別式である。この時、次の定理が成り立つ。

定理 4.10 a,b を互いに素な整数の組、p を $p \nmid [\mathcal{O}:\mathcal{A}]$ が成り立つ素数、$\mathfrak{p}$ を p の上にある $\mathcal{O}$ の素イデアルで、$v_{\mathfrak{p}}(\langle a+b\theta\rangle) \neq 0$ であるとする。この時、次が成立する。

1. $r \in R(p)$ に対して、$\mathcal{O}$ の一次素イデアル $\mathfrak{p}_r$ で、$\mathcal{A}$ の p, r に対応する一次素イデアルで $\mathfrak{q}_r$ の上にあるものが唯一存在する。さらに

$$v_{\mathfrak{p}_r}(\langle a + b\theta \rangle) = l_{\mathfrak{p},\mathcal{A}}$$

   となる。
2. 少なくとも一つの $r \in R(p)$ に対し $e_{\mathfrak{p},r}(a, b) \neq 0$
3. $p \nmid c_d$ ならば $r \neq \infty$ となる r に対し $\mathfrak{p} = \mathfrak{p}_r$
4. $p | c_d$ ならば $r \neq \infty$ あるいは $r = \infty$ となる r に対し $\mathfrak{p} = \mathfrak{p}_r$
5. $p | F(a, b)$ または $p | c_d$

更に、順同型写像 $\psi_{p,s}$ を次のように定義する。

$$\begin{aligned} &\psi_{p,s} : \mathbb{Z}[\theta] \to \mathbb{F}_p, \qquad \psi_{p,s}(\theta) = s, \text{ if } s \neq \infty \\ &\psi_{p,\infty} : \mathbb{Z}[\theta^{-1}] \to \mathbb{F}_p, \quad \psi_{p,\infty}(\theta^{-1}) = 0 \end{aligned}$$

すると、定理上記定理に現れる $\mathcal{O}$ の素イデアル $\mathfrak{p}_s$ は次のように表現される。

$$\begin{aligned} &\mathfrak{p}_s = \langle p, \beta_0 - \psi_{p,s}(\beta), ..., \beta_{d-2} - \psi_{p,s}(\beta) \rangle \text{ if } s \neq \infty \\ &\mathfrak{p}_s = \langle p, \beta_0, ..., \beta_{d-2} \rangle \text{ if } s = \infty \end{aligned}$$

この定理によって、篩計算で得られた $c_d \mathcal{N}(a + b\theta)$ の素因数分解で、normal prime の上にあるにある素イデアルについては、素イデアルの生成元、ならびに 付値が具体的に計算できることになる。

一方で exceptional prime 上にある素イデアル $\mathfrak{p}$ (これも exceptional と呼ぶことにする) の生成元、および $\mathfrak{p}$ に関する $\langle a + b\theta \rangle$ の付値については、篩計算結果から導くことは出来ず、正確な素イデアル分解を導くには、任意の素数 p に対する $\mathcal{O}$ 上の素イデアル分解法である Buchmann-Lenstra アルゴリズム、ならびに一般的な素イデアルに対する付値計算アルゴリズムを用いて、別に計算しなければならない。

しかし実際には exceptional prime の個数は、多くても 3,4 個と少なく、値も 2 や 3, あるいは 11 といった小さい素数である場合が多い。そのため exceptional であることを無視したとしても、最終的な誤差はあまり大きくならないことが計算機実験によって判った。そこで今回の実装では、この exceptional prime についても normal prime と同様の扱いをすることとし、その結果として単項イデアル近似に生じる誤差については、代数的数の平方根アルゴリズムの最終部、すなわち誤差修正部で吸収させることにした。この方法によって、代数的平方根が正しく求められることを確認している。

4.3.3　イデアルの単純化

代数体 $\mathbb{K}$ の分数イデアル $I \in \mathcal{I}$ に対し I_{nm}, I_{dm} を次のように定義する。

$$I_{nm} = \prod_{v_{\mathfrak{p}}(I)>0} \mathfrak{p}^{v_{\mathfrak{p}}(I)}$$
$$I_{dn} = \prod_{v_{\mathfrak{p}}(I)<0} \mathfrak{p}^{-v_{\mathfrak{p}}(I)}$$

イデアル I のノルムを $\mathcal{N}(I)$ とした時、I の complexity $\mathcal{C}(\mathcal{I})$ を $\mathcal{C}(I) = \mathcal{N}(I_{nm})\mathcal{N}(I_{dn})$ で定義する。

ここで、relation の積 $\alpha = \prod_{i \in S}(a_i + b_i\theta)$ で生成されるイデアルは、前節までの方法を用いて素イデアル分解できるものの、このままでは因子となる素イデアルの個数もその巾指数も大きすぎ、演算効率がよくない。そこで、上記の分数イデアルの complexity $\mathcal{C}()$ を用い、イデアルを単純化することを試みる。任意の $i \in S$ について、e_i を $\{1, -1\}$ からランダムに選ぶ。これによって、上式の右辺の積が二つに分けられる。

$$\alpha_{nm} = \prod_{i \in S, e_i = 1} (a_i + b_i\theta), \alpha_{dn} = \prod_{i \in S, e_i = -1} (a_i + b_i\theta)$$

さらに、$\alpha = \alpha_{nm}\alpha_{dn}$ であることから

$$\frac{\alpha}{\alpha_{dn}^2} = \frac{\alpha_{nm}}{\alpha_{dn}}$$

となる。もし右辺の分数式の代数的平方根 β が求められれれば、α の平方根は $\alpha_{dn}\beta$ として計算される。

また、e_i をランダムに選んでいることから、$\langle \alpha_{nm} \rangle$ と $\langle \alpha_{dn} \rangle$ に含まれる素イデアルが相殺され、右辺の各素イデアルに対するべき指数の絶対値が小さくなり、$\mathcal{C}(\left\langle \frac{\alpha_{nm}}{\alpha_{dn}} \right\rangle) \ll \mathcal{C}(\langle \alpha \rangle)$ となることが予想される。

さらに、$\{e_i | i \in S\}$ を、$\{1, -1\}$ からうまく選び直すことによって、complexty $\mathcal{C}\left\langle \frac{\alpha_{nm}}{\alpha_{dn}} \right\rangle$ をさらに小さくできる可能性がある。このための手段として、今回の実装では、次のような単純な方法を取り入れた。まず $E = \{1, -1\}^{|S|}$ とし、関数 $U : E \to \mathbb{C}$ を次のように定義する。

$$U((e_1, ..., e_{|S|})) = \log_2 \mathcal{C}\left(\left\langle \prod_{i \in S}(a_i + b_i\theta)^{e_i}) \right\rangle\right)$$

アルゴリズム 4.11 (イデアル単純化)
τ を定めておく[a] 。

1. $e \in E$ をランダムに取り、$\Theta = U(e)$ とする。
2. $\Theta_{ini} = \Theta$ とする。
3. $i = 1$ から $|S|$ について、次を繰り返す。
   (a) $-e_i$ を e_i で置き換えたものを $\tilde{e}$ とする。
   (b) $U(\tilde{e}) < \Theta$ であれば e を $\tilde{e}$ で置きかえて、$\Theta = U(\tilde{e})$ とする。
4. $\Theta_{ini} - \Theta > \Theta \times \tau$ の時 2 に戻る。

[a] 今回の実装では $\tau = 0.001$ とした

文献 [20] では、焼きなまし法 (simulated anneling) を用いて complexity が小さくなるものを探索している。焼きなまし法は次のように記述される。

アルゴリズム 4.12 焼きなまし法
$\Theta_{ini}, \Theta_{fin}, \tau$ を定めておく。

1. ランダムに $e \in E$ を選び $\Theta = \Theta_{ini}$ とする。
2. ランダムに選んだ i に対して e から e_i の符号のみを変えたものを f とする。
3. $\Delta = U(f) - U(e)$ とおく。
4. $\Delta > 0$ の時、確率 p を $\exp(-\Delta/\Theta)$ とし、それ以外の時は $p = 1$ とする。
5. 確率 p で $e = f$ とし、$\Theta = \Theta \times \tau$ とする。
6. $\Theta > \Theta_{fin}$ ならば 2 に戻る。

焼きなまし法では、ランダムに選んだ relation の分子分母を入れ替える操作が必要となるため、relation データのランダムファイルアクセスが必須となるが、一般数体篩法で利用される relation データは巨大であり、メモリに保持可能とは限らないため、この操作は関数 U の減少率の向上よりも、ランダムデータアクセスに対する速度低下の影響の方が大きくなる可能性が高いため、今回の実装には焼きなまし法は利用せず、アルゴリズム 4.11 を使った。

例 4.13 (RSA100 の場合)

$$\alpha = (-4765231 + 41477\theta)(-9100546 + 66691\theta)\cdots[212190 \text{ 個の積}]$$

に対し、$\langle\alpha\rangle$ を素イデアル分解すると、次の通りである。

$$\langle\alpha\rangle = \mathfrak{p}_1^{239708}\mathfrak{p}_2^{251056}\mathfrak{p}_3^{104636}\mathfrak{p}_4^{52094}\mathfrak{p}_5^{52734}\mathfrak{p}_6^{94952}\ldots$$

$$\log\left(C(\langle\alpha\rangle)\right) = 35774945.32$$

各 relation をランダムに分子分母を振り分けると、次の通りとなった。

$$\left\langle\frac{\alpha_{nm}}{\alpha_{dn}}\right\rangle = \mathfrak{p}_1^{-898}\mathfrak{p}_2^{504}\mathfrak{p}_3^{568}\mathfrak{p}_4^{78}\mathfrak{p}_5^{-198}\mathfrak{p}_6^{256}\mathfrak{p}_7^{-240}\mathfrak{p}_8^{-222}\cdots,$$

$$\log\left(C\left(\left\langle\frac{\alpha_{nm}}{\alpha_{dn}}\right\rangle\right)\right) = 6924502.90,$$

$$(\text{分子} = 3481594.10, \text{分母} = 3442908.80)$$

さらに、本節で述べた方法で、イデアルの complexity を下げると次の通りとなった。

$$\left\langle\frac{\alpha_{nm}}{\alpha_{dn}}\right\rangle = \mathfrak{p}_1{}^{6}\mathfrak{p}_2{}^{4}\mathfrak{p}_3{}^{0}\mathfrak{p}_4{}^{0}\mathfrak{p}_5{}^{0}\mathfrak{p}_6{}^{0}\mathfrak{p}_7{}^{-2}\mathfrak{p}_8{}^{-10}\cdots$$

$$\log\left(C\left(\left\langle\frac{\alpha_{nm}}{\alpha_{dn}}\right\rangle\right)\right) = 2146584.77,$$

$$(\text{分子} = 1072949.51, \text{分母} = 1073635.25)$$

以上のように巾指数が大幅に小さくなるのが判る。

例 4.14 ($N = 5\cdot 3^{45}+2$ の場合)
$N = 5\cdot 3^{45}+2$ とし、N を数体篩法で素因数分解とする場合に用いる定義多項式として $f(x) = 5x^5+2$ を取り、$f(x) = 0$ の根を θ とする。篩部、線形代数部および平方剰余記号の計算より、次の平方数が得られる。

$$\alpha = (-201+\theta)(-131+\theta)(-3+\theta)(1+\theta)(2+\theta)(75+\theta)\cdots$$

イデアルの単純化を行いイデアル平方根 $\sqrt{\alpha_{nm}/\alpha_{dn}}$ の分子イデアル、分母イデアルに対し、素イデアル因子を決定する。
分子イデアル = { $\langle 2,\theta\rangle^6$, $\langle 7,\theta-1\rangle^3$, $\langle 37,\theta-12\rangle$, $\langle 47,\theta-8\rangle^2$,
$\langle 73,\theta-60\rangle^2$, $\langle 83,\theta-8\rangle$, $\langle 139,\theta-28\rangle$, $\langle 229,\theta-110\rangle$, $\langle 251,\theta-227\rangle$,
$\langle 307,\theta-73\rangle$, $\langle 317,\theta-57\rangle$, $\langle 331,\theta-121\rangle$, $\langle 409,\theta-75\rangle$, $\langle 433,\theta-124\rangle$,
$\langle 457,\theta-372\rangle$, $\langle 487,\theta-107\rangle$, $\langle 491,\theta-361\rangle$, $\langle 677,\theta-140\rangle$, $\langle 727,\theta-579\rangle$,
$\ldots$, $\langle 9433,\theta-5881\rangle$ }
分母イデアル = { $\langle 3,\theta-2\rangle$, $\langle 23,\theta-12\rangle$, $\langle 97,\theta-52\rangle$, $\langle 109,\theta-24\rangle$,
$\langle 113,\theta-67\rangle^2$, $\langle 163,\theta-108\rangle$, $\langle 173,\theta-160\rangle^2$, $\langle 181,\theta-103\rangle^2$, $\langle 199,\theta-179\rangle^2$,
$\langle 251,\theta-15\rangle$, $\langle 337,\theta-286\rangle^2$, $\langle 349,\theta-221\rangle$, $\langle 419,\theta-145\rangle$, $\langle 439,\theta-337\rangle^2$,
$\langle 547,\theta-262\rangle$, $\langle 719,\theta-657\rangle$, $\langle 739,\theta-478\rangle$, $\langle 751,\theta-427\rangle$, $\langle 937,\theta-539\rangle$,
$\ldots$, $\langle 9871,\theta-2303\rangle$ }

4.4　平方根イデアルから平方根の近似

4.4.1　アイデア

$\sqrt{\langle\gamma\rangle}$ から $\sqrt{\gamma}$ の近似 μ を LLL アルゴリズムを用いて求めるアルゴリズムについて説明する。本部分は、Montgomery, Nguyen による代数的平方根アルゴリズムのメインパートである。基本的な流れは次の通り。実質的に行っている操作は、与えられたイデアルに対する単項イデアル近似である。

アルゴリズム 4.15 平方根近似アイデア

1. $\sqrt{\langle\gamma\rangle}$ の分子または分母から整イデアル I を選ぶ。
2. イデアル I から LLL アルゴリズムを用いて、適切な $\delta \in I$ を求める。
3. $\mu = \mu \times \delta^{\pm 1}$
4. $\sqrt{\langle\gamma\rangle}$ の分子分母から全ての整イデアルを選択し終えてなかったら、 step 1 に戻る。
5. $\theta = \gamma \times \mu^{-2}$ として終了。

4.4.2　具体的なアルゴリズム

平方根近似アルゴリズムの各ステップ $l \geq 1$ で用いる変数についてまとめておく。

(1) 代数的数 γ_l：近似誤差の 2 乗。アルゴリズム中ではノルムの対数値 (複素数) 等の計算を行うのみ。仮想的な変数である。

(2) 符号 s_l：抽出したイデアル I が分子か $(= 1)$ 分母か $(= -1)$ を表す。

(3) 分数イデアル G_l：未選択イデアル。初期値は $\sqrt{\langle\gamma\rangle}$ であり、終了時は $\langle 1\rangle$。アルゴリズム中では素イデアルの積として保持する。

(4) 整イデアル I_l：G_l の分子又は分母から抽出されるイデアル。アルゴリズム中では Hermit Normal Form で表現する。

(5) 整イデアル H_l：ループ毎の誤差イデアル。G_l と $\langle\gamma_l\rangle$ の差と等しい。アルゴリズム中では Hermit Normal Form として持つ。

(6) 代数的整数 δ_l：イデアル I の近似単項イデアルの生成元。アルゴリズム中では θ の多項式で表現する。

アルゴリズム 4.16 (Montgomery,Nguyen アルゴリズム)

1. 事前検証
   $\mathcal{C}\sqrt{\langle\gamma\rangle} = 1$ の時は $\theta = \gamma$ として終了。
2. 初期設定
   $l = 1, \gamma_1 = \gamma, G_1 = \sqrt{\langle\gamma\rangle}, H_1 = \langle 1\rangle$
3. イデアル選択
   $s_l = 1$ の時は G_l の分子から、$s_l = -1$ の時は分母から整イデアルの一部を選択する。
4. 単項イデアル近似
   適切な $\delta_l \in I_l$ を LLL アルゴリズムを用いて選ぶ。
   $\gamma_{l+1} = \gamma_l \times \delta_l^{-2s_l}$.
   $G_{l+1} = \left(\frac{I_l}{H_l}\right)^{-s_l} G_l$.
   $\mathcal{C}(G_{l+1}) = \frac{1}{\mathcal{N}(I_l/H_l)}\mathcal{C}(G_l)$
5. 終了判定
   $\mathcal{C}(G_{l+1}) = 1$ ならば $\theta = \gamma_{l+1}, \mu = \prod_{i=1}^{l} \delta_i^{s_i}$ として終了。
6. 誤差イデアル
   $H_{l+1} = \frac{\langle\delta_l\rangle}{I_l}$
7. 次ループへの準備
   $s_{s+1} = s_1 \times (-1), l = l + 1$。
   step 3 に戻る。

4.4.3　整イデアル I の選択

処理開始時に定めた定数 LLL_{max} に対して、I_l のノルム $\mathcal{N}(I_l)$ が LLL_{max} になるべく近くなるように I_l を定める。具体的には、まず $I_l = H_l$ とし、$s_l = 1$ の時は G_l の分子から、$s_l = -1$ の時は分母から、素因子イデアルの積を抽出し、$\mathcal{N}(I_l)$ が LLL_{max} を越えない範囲で出来るだけ多く I_l とイデアル乗算を行う。なお処理が進み、$\mathcal{N}(G_l)$ が小さくなり、抽出する素イデアルが足りない場合には、$\mathcal{N}(I_l)$ が LLL_{max} よりかなり小さくなることや、$I_l = N_l$ となることもあり得る。イデアル I_l は Hermit Normal Form で表現する。

例 4.17 ($N = 5 \cdot 3^{45} + 2$ の場合)
まず、各素イデアルを HNF で表現する。例えば素イデアル $\langle 47, \theta - 8\rangle$

の HNF 表現は次の通りである。

$$\langle 47, \theta - 8 \rangle = \begin{pmatrix} 47 & 7 & 9 & 25 & 12 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$LLL_{max} = 2^{160}$ とする。LLL_{max} の決め方は後の節で述べる。分子イデアルから取り出した素イデアル積を作り、イデアルノルムが出来るだけ LLL_{max} に近くなるよう $\langle 2, \theta \rangle$ から $\langle 677, \theta - 140 \rangle$ までの 18 個、延べ 27 個の素イデアルの積 I を生成する。スペースの都合により、第一行を成分毎に改行して記述した。

$$\begin{aligned} I &= \langle 2, \theta \rangle \cdots \langle 677, \theta - 140 \rangle \\ &= \begin{pmatrix} 167244377328879100397750502732363140172960068 \\ \quad 75991348495299820844669113970277953951964768 \\ \quad\quad 10235250207239455847065182259362606951975584 4 \\ \quad\quad\quad 20373654549020750262411092710819131883540704 \\ \quad\quad\quad\quad 309702107287544228281262448177123083269349 04 \\ 0 \quad 2 \quad 0 \quad 0 \quad 0 \\ 0 \quad 0 \quad 2 \quad 0 \quad 0 \\ 0 \quad 0 \quad 0 \quad 2 \quad 0 \\ 0 \quad 0 \quad 0 \quad 0 \quad 2 \end{pmatrix} \end{aligned}$$

イデアル I のノルム (=HNF の行列式) は次の通り。

$$\begin{aligned} \mathcal{N}(I) &= 267591003726206560636400804371781024276736 1088 \\ &= 2^6 \cdot 7^3 \cdot 37 \cdot 47^2 \cdot 73^2 \cdot 83 \cdot 139 \cdot 229 \cdot 251 \cdot 307 \cdot 317 \cdot 331 \\ &\quad \cdot 409 \cdot 433 \cdot 457 \cdot 487 \cdot 491 \cdot 677 \end{aligned}$$

4.4.4　代数的整数の選択

ここでは、$\mathcal{N}(\langle \delta_l \rangle / I_l)$ がなるべく小さくなるような $\delta_l \in I_l$ を LLL アルゴリズムを 2 回適用する事で求める。

LLL 簡約 — 一回目　まず、Hermit Normal Form で表現された整イデアル I_l に対し、LLL アルゴリズムを用いて基底の簡約を行う。Hermit Normal Form は、上半三角行列であり、低次の基底に対応する係数に、絶対値が大きい要素が集中しているため、LLL 簡約を行うことによって、

係数の絶対値の大きさを小さく、しかもバランスさせることが出来る。これによって得られる LLL 簡約基底を $(v^{(j)})_{j=1}^{d} = ([v_1^{(j)}, ..., v_d^{(j)}]^t)_{j=1}^{d}$ とする。

LLL 簡約 — 二回目　定数 $c > 0$ ならびに λ_j を次で定義する。

$$c^d = \frac{LLL_{max}}{\mathcal{N}(I_l)} \sqrt{\frac{|\mathcal{N}(\gamma_l)|^{s_l}}{|\Delta(\mathbb{K})|}}$$
$$\lambda_i = \frac{c}{|\sigma_i(\gamma_l)^{s_l/2}|} \text{ for } 1 \leq i \leq d$$

これらの値、ならびに $v = \sum_{j=1}^{d} v_j \omega_j$, $\sigma_j(v)$ を用いて、次の $2d \times d$ 行列 Ω を生成する。

$$\Omega = \begin{pmatrix} v_1^{(1)} & \cdots & v_1^{(d)} \\ \vdots & & \vdots \\ v_d^{(1)} & \cdots & v_d^{(d)} \\ \lambda_1\sigma_1(v^{(1)}) & \cdots & \lambda_1\sigma_1(v^{(d)}) \\ \vdots & & \vdots \\ \lambda_d\sigma_d(v^{(1)}) & \cdots & \lambda_d\sigma_d(v^{(d)}) \end{pmatrix}$$

ただし、$f(x)$ に複素根がある場合には、複素共役根 $\sigma_i, \bar{\sigma}_i$ に対して、$\sigma_i(v), \bar{\sigma}_i(v)$ の代わりに、$\Re(\sigma_i(v))\sqrt{2}, \Im(\sigma_i(v))\sqrt{2}$ を利用する。また、$\sigma_j()$ の計算など各々の複素数は対数で計算することで演算効率を上げると共に[1] 、実数値については小数点以下を四捨五入し、整数値に丸める。この行列 Ω に対し二回目の LLL 簡約を適用する。

δ_l の抽出　二回目の LLL 簡約で得られた行列の上半行列で、第一列目の d 個の要素 (Ω の $(v_1^{(1)}, ..., v_d^{(1)})$ にあたる位置の要素) を係数とし整数基底を用いて生成される要素を δ_l とする。

例 4.18 ($N = 5 \cdot 3^{45} + 2$ の場合)

例 4.17 で計算されたイデアル I の HNF に対し、LLL 簡約 LLL() を適用する。これによって行列の各要素は、絶対値がほぼ均衡した行列となる。

[1] 複素数 $a + bi$ の対数は $\log(a + bi) = (\log(a^2 + b^2))/2 + i \arctan(b/a)$ で計算される。

$$\mathrm{LLL}(I) = \begin{pmatrix} 502277136 & 589486184 & -316607656 & -271370372 & 167578316 \\ 975003110 & -575431678 & 181218768 & 142646214 & -465928794 \\ -8268458 & -348810556 & -756650446 & 920251668 & 587620000 \\ 397867350 & -347106474 & 407839890 & -476483736 & 1321527182 \\ -107542952 & -596642536 & -754946364 & -436449478 & -1083098812 \end{pmatrix}$$

この HNF から 10×5 行列 Ω を生成する。

$$\Omega = \begin{pmatrix} 502277136 & 589486184 & -316607656 & -271370372 & 167578316 \\ 975003110 & -575431678 & 181218768 & 142646214 & -465928794 \\ -8268458 & -348810556 & -756650446 & 920251668 & 587620000 \\ 397867350 & -347106474 & 407839890 & -476483736 & 1321527182 \\ -107542952 & -596642536 & -754946364 & -436449478 & -1083098812 \\ -3134775401 & 3225153566 & 2929504035 & 106734687 & 2625804052 \\ 844229362 & -224656659 & -3865908654 & 936604570 & -2080117513 \\ -4197547488 & 521271833 & -1774020234 & 1743839786 & -140044400 \\ 1664626262 & 2191697916 & 1295344480 & -3971323238 & -1331309036 \\ -1426474883 & 137138122 & 2570340636 & 1243626555 & -6889968359 \end{pmatrix}$$

行列 Ω に対し、LLL 簡約を実施する。

$$\mathrm{LLL}(\Omega) = \begin{pmatrix} 318115812 & 589486184 & 1091763320 & -500768980 & 167578316 \\ -432785464 & -575431678 & 399571432 & -1226569806 & -465928794 \\ 571441112 & -348810556 & -357079014 & -176940876 & 587620000 \\ -823590210 & -347106474 & 50760876 & -813617670 & 1321527182 \\ -1033092014 & -596642536 & -704185488 & -1680495426 & -1083098812 \\ -3134775401 & 3225153566 & 2929504035 & 106734687 & 2625804052 \\ 844229362 & -224656659 & -3865908654 & 936604570 & -2080117513 \\ -4197547488 & 521271833 & -1774020234 & 1743839786 & -140044400 \\ 1664626262 & 2191697916 & 1295344480 & -3971323238 & -1331309036 \\ -1426474883 & 137138122 & 2570340636 & 1243626555 & -6889968359 \end{pmatrix}$$

$\mathrm{LLL}(\Omega)$ の第一列の先頭 5 要素

$$(318115812, -432785464, 571441112, -823590210, -1033092014)$$

と整数基底 $(1, 5x, 5x^2, 5x^3, 5x^4)$ との内積によって、イデアル I を近似する代数的数 $\delta \in I$ が求められる。

$$\begin{aligned}\delta &= -51654600700\theta^4 - 41179510500\theta^3 + 28572055600\theta^2 \\ &\quad -21639273200\theta + 318115812\end{aligned}$$

この δ のノルムと イデアル I のノルムの比を比べると

$$\mathcal{N}(\delta)/\mathcal{N}(I) = 524$$

であり、これは比較的小さい値であると言える。これは $\langle\delta\rangle$ はイデアル I のよい近似となっていることを示している。

4.4.5　アルゴリズムの停止性

アルゴリズムが有限回のステップで止まることは、次の定理から保証される。

定理 4.19
代数体 $\mathbb{K}$ のみに依存する定数 C が存在し、次が成立する。

$$|\mathcal{N}(\delta_l)| \leq C \times \mathcal{N}(I_l)$$

特に C は $\mathcal{N}(I_l), LLL_{max}$ とは独立であり、$\mathcal{N}(H_l) \leq C$ も成り立つ。

4.4.6　誤差イデアル

選択された代数的整数 $\delta_l \in I_l$ によって、I_l に近い単項イデアルが生成されるが、必ずしも $\langle\delta_l\rangle$ が I_l と完全に一致するとは限らない。そのため、平方根近似アルゴリズムの各ループの最後に、誤差イデアル $H_{l+1} = \langle\delta_l\rangle / I_l$ を計算しておき、次のループのイデアル I_{l+1} に含める必要がある。この誤差イデアル H_{l+1} を求める方法について述べる。

一つの方法としては、ノルム $\mathcal{N}_{\mathbb{K}}(\delta_l)$ の素因数分解が求められれば、$\langle\delta_l\rangle$ の素イデアル分解が得られ、これらから I_l の因子を取り除き、残った素イデアルの積を H_{l+1} とする方法がある。しかしこの方法は、ノルムの素因数分解を求めなければならず、常に簡単に求められるとは限らない。

文献 [20] には、H_{l+1} について "directly computed" と述べられているのみであり、具体的な計算方法に関する記述はない。

一方でイデアル I_l の逆イデアル I_l^{-1} が求められれば、それと $\langle\delta_l\rangle$ との積をとることで、誤差イデアルが求められる。この逆イデアルの求め方について [5] のアルゴリズムを用いることにする。

アルゴリズム 4.20 (逆イデアル) 代数体 $\mathbb{K}$ の整数環 $\mathcal{O}$ の整数基底 $(\omega_i)_{1\leq i\leq d}$ および、任意に与えられた整イデアル I で ω_i 上 I の $\mathbb{Z}$ 基底 γ_j で与えられる d 次正方行列表現 M に対し、逆イデアル I^{-1} の HNF を求める。

1. n 次正方行列 $T = (t_{ij}), t_{ij} = Tr(\omega_i\omega_j)$ を計算する。但し $Tr(a)$ は、代数的整数 a の d 次最小多項式の $d-1$ 次係数を表す。(T は different と呼ばれる。)
2. $d = det(T)$ とする。(d は $\mathbb{K}$ の判別式 $\mathrm{disc}(\mathbb{K})$ と等しい。)
3. HNF が dT^{-1} である整イデアルの $\mathbb{Z}$ 基底を $\{\delta_j\}$ とする。
4. n^2 個 の代数的数 $\gamma_i\delta_j$ の ω_j に対する $n\times n^2$ 行列表示の HNF (n 次正方行列) を N とする。
5. $P = \mathrm{disc}(\mathbb{K})(N^tT)^{-1}$ とし、P の全ての要素の分母の LCM を e とする。
6. W を eP の HNF とする。
7. I^{-1} の HNF、(W, e) を出力する。(e は分母)

例 4.21 ($N = 5\cdot 3^{45}+2$ の場合)
例 4.17 で求められたイデアル I の逆イデアルは次の通り。

$$I^{-1} = \begin{pmatrix} a & 0 & 0 & 0 & 681370833000623387848121289573254159230125 82 \\ 0 & a & 0 & 0 & 481359477437616550930840415486270288635339 50 \\ 0 & 0 & a & 0 & 399380629914242742722599685603463350999124 36 \\ 0 & 0 & 0 & a & 912530288335792795530813887620851862209953 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
$$\times 1/167244377328879100397750502732363140172960068$$
$$a = 836221886644395501988752513661815700864800 34$$

さらに抽出された代数的数 δ が生成するイデアルとの積によって求められる誤差イデアル H の HNF は次の通り。

$$H = \langle\delta\rangle \times I^{-1} = \begin{pmatrix} 262 & 196 & 18 & 14 & 40 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

4.4.7　LLL_{max} の決定

整イデアル I の選択と行列 Ω の生成で利用する定数 LLL_{max} について考察する。LLL_{max} は、アルゴリズムの停止性を保証するために、各ループの最後に計算する誤差イデアルのノルム以上でなければならない。誤差イデアルのノルムは、定理 4.19 の定数 C によって上限が押えられることが示されている。経験的には、LLL_{max} を C^2 程度にとれば十分である。この C の具体的な値は以下の通りである。(文献 [20] の付録参照。) 整数環の整数基底を $\omega_1, ..., \omega_d$, $\sigma_j(a)$ を $\sigma_j(\theta) = \theta_j$ (θ_j は定義多項式 $f(x) = 0$ の j 番目の根) で定まる準同型写像とする。

$$
\begin{aligned}
C_1 &= \max_{1 \le j \le d} \sqrt{\sum_{i=1}^{d} |\sigma_j(\omega_i)|^2} \\
C_2 &= 2^{d(d-1)/4} d^d 2^{d+1} \\
C_3 &= \sqrt{1 + dC_1} \\
C &= 2^{d(d-1)/2} \max(1, C_1^d) C_2 C_3^d
\end{aligned}
$$

実際には LLL_{max} を C よりずっと小さい値に設定しても十分動くだけでなく、処理が高速になることも少なくないため、演算効率の点からは、LLL_{max} をいろいろ変化させ、最適値を決める必要がある。

例 4.22 (計算例：RSA100 の場合)
100 桁の合成数

$$
\begin{aligned}
RSA100 =& 1522605027922533360535618378132637429718068114961\backslash \\
& 3806886579084945801229632589528976540003506920061 39
\end{aligned}
$$

に関する代数体の定義多項式、ω_i, σ_j は次の通り。

$$
\begin{aligned}
f(x) =& 162177120x^5 + 88959473314x^4 + 40889525457549x^3 \\
& -30210012886398365x^2 - 11144881792587126717x \\
& +11474234546362572 5398
\end{aligned}
$$

$$
\begin{aligned}
(\omega_i) =& \{1, \\
& 81088560x, \\
& 162177120x^2 + 5322994x, \\
& 162177120x^3 + 86411554x^2 + 53645077x, \\
& 18019680x^4 + 1059474274/9x^3 + 1301911441/9x^2 + 546259853/9x + 5/9 \\
& \}
\end{aligned}
$$

$$
(\sigma_j) = \{-297.42,\ 10.02,\ 461.72,\ -361.43 + 619.02i,\ -361.43 - 619.02i\}
$$

これより定数 C の値は次のように計算される。

$$\begin{aligned} C_1 &= 47361539534591102 98.111, \\ C_2 &= 6400000, \\ C_3 &= 4866289116.698, \\ C &= 2^{255.54}, \end{aligned}$$

例 4.23 ($N = 5 \cdot 3^{45} + 2$ の場合)
定義多項式 $f(x) = 5x^5 + 2$ について同様の計算を行うと C は次の通り。

$$C_1 = 44.436, C_2 = 6400000, C_3 = 14.939$$

より $C = 2^{79.482}$ であるから、LLL_{max} としては $2^{160}(\approx C^2)$ とすれば十分である。

4.5　誤差項の平方根計算

イデアルの平方根の近似計算より、$\theta = \gamma_{l+1}$ を得る。この θ の平方根を素数 p を法とする平方根と CRT、あるいは Hensel Lifting を組み合わせることで、求められる。さらに、代数体 $\mathbb{K}$ の定義多項式が奇数次であり、かつ代数的数のノルムの値が判っている場合には、Couveignes の方法を用いて、より効率的に代数的平方根を求めることができる。今回の実装ではこの Couveignes の方法を用いる事とした。

命題 4.24 (Couveignes)
代数体 K の次数が奇数ならば、代数的平方数 $\gamma \in \mathbb{K}$ に対し、

$$\delta = \gamma^{\frac{(1+p+p^2+\cdots+p^{d-1})(p-2)+1}{2}} \sqrt{\mathcal{N}(\gamma)} \mod p$$

とすると $\delta = \pm\sqrt{\gamma} \mod p$。

なお、代数的数の平方根のノルムは、最後に残る誤差イデアル H の行列式の値からおおよそ推測されるものの、exceptional prime に関する手続きを無視しているため、完全にノルムと一致する保証がない。しかし、ノルムの値は高々定数倍にしか影響していないことから次のように正しいノルムを推測する事ができる。

平方根ノルムの仮の値 $\tilde{b}$ を用いて計算された平方根 $\tilde{\delta}$ から $\sqrt{\tilde{\delta}^2/\gamma} \mod p$ を計算し、有理数近似[2] を行ったものを ε と置く。

$$\sqrt{\mathcal{N}(\gamma)} = b/\varepsilon$$

[2] $n \mod p$ が与えられた時 $bn + cp = a$ で a, b が共に $\sqrt{n}$ 以下になるように取れば、$n = a/b \mod p$ となる。これは拡張ユークリッド算法を途中まで行うことで得られる。

と推定される。ただし、$\tilde{\delta}^2/\gamma \mod p$ が平方数にならなかった場合は失敗であり、p をより大きい値に取り直す必要がある。

一般的には複数の素数を法とした平方根から、Chinese Remainder Theorem を使って持ち上げる操作が必要である。

命題 4.25 (Chinese Remainder Theorem)
$p_1, ..., p_s$ を互いに素な自然数とし、

$$x = x_i \mod p_i$$

とする。$P = \prod_{i=1}^{s} p_i$, $P_i = P/p_i$, $q_i = pP_i^{-1} \mod p_i$ とおけば、

$$y = \sum q_i x_i P_i$$

に対し

$$x = y \mod P$$

となる。

例 4.26 ($N = 5 \cdot 3^{45} + 2$ の場合)
$f(x) = 5x^5 + 2$ とする。平方根近似アルゴリズムで、最終的に残った誤差イデアル H は

$$H = \begin{pmatrix} 982 & 318 & 934 & 348 & 914 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

であった。これより $\mathcal{N}(\sqrt{\gamma}) = \det(H) = 1964$ である。

$p = 99981911$ に対し relation $a_i + b_i\theta$ と平方根近似で得られた δ_l を乗算すると、

$$\gamma \mod p = 4545\theta^4 + 99977421\theta^3 + 3180\theta^2 + 99979001\theta + 2456$$

が得られた。これに対して couveignes の方法を適用すると、

$$\sqrt{\gamma} \mod p = 65\theta^4 + 99981886\theta^3 + 35\theta^2 + 99981881\theta + 14$$

2 乗すると、上記の式に一致することが確かめられる。更に、各係数を p を法として有理数近似を行うと、次の式が得られる。

$$\beta = 65\theta^4 - 25\theta^3 + 35\theta^2 - 30\theta + 14$$

計算すると、確かに正しい代数的平方根であった。今回の場合、一つの素数のみで正しい解が得られている。

これらの値から代数的数の平方根に $M = 3^9$ を代入した法 N での値が得られる。

$$(\sqrt{\alpha})|_{\theta=M} = \alpha_{dn}(M)\delta(M)\beta(M) \mod N = 8076504220635373905778$$

更に、別に求められた有理数側の平方根

$$\sqrt{\prod(a_i + b_i M)} \mod N = 14197497622838308876303$$

より、次のように素因数分解が得られる。

$$N = 5 \cdot 3^{45} + 2 = 3670785863 \cdot 4024087507159$$

関連図書

[1] 青木和麻呂, 植田広樹, 木田 祐司, 下山武司, 園田裕貴, "一般数体篩法実装実験 (1)-概要", SCIS2004 予稿集, 2004.

[2] C. Batut, K. Belabas, D. Bernardi, H. Cohen, M. Olivier, "User's Guide to PARI / GP," for version 2.1.1, 2000 Available at `http://pari.math.u-bordeaux.fr/`

[3] R.P.Brent, "Recent Progress and Prospects for Integer Factorisation Algorithms," Proceedings of COCOON2000, LNCS 1858, pp.3-22, 2000.

[4] J.P. Buhler, H.W. Lenstra Jr., and C. Pomerance, "Factoring Integers with the Number Field Sieve", pp.50-94 in [14], 1993.

[5] Henri Cohen, "A Cource in Computational Algebraic Number Theory," Graduate Texts in Mathematics 138, Third, Corrected Printing 1996, Springer–Verlag, 1993.

[6] J. Cowie, B. Dodsonm, R.M. Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery, and J. Zayer, "A World Wide Number Field Sieve Factoring Record: On to 512 Bits", *Asiacrypt'96*, LNCS 1163, pp. 382-394, Springer-Verlag, 1997.

[7] M. Elkenbracht-Huizing, "A Multiple Polynomial General Number Field Sieve", *Experimental Mathematics*, 5(3), pp.231-253, 1996. Available from `http://citeseer.nj.nec.com/cache/papers/cs/5307/http:zSzzSzwww.expmath.comzSzrestrictedzSz5zSz5.3zSzhuizing.pdf/elkenbracht-huizing96implementation.pdf`

[8] J. Franke et al., "RSA576 (Email announcement)", December 3, 2003. Available at `http://www.crypto-world.com/announcements/rsa576.txt`

[9] J.E. Gower, "Rotations and Translations of Number Field Sieve Polynomials", *Asiacrypt 2003*, LNCS 2894, pp.302-310, Springer-Verlag, 2003.

[10] 伊豆 哲也, 木田 祐司, "素因数分解の現状について", 日本応用数理学会論文誌, Vol.13, No.2, pp.289-304, 2003.

[11] 木田 祐司, "暗号アルゴリズムの詳細評価に関する報告書," CRYPTREC 暗号アルゴリズム及び関連技術の評価報告 no.0022, 2002 `http://www.shiba.tao.go.jp/kenkyu/CRYPTREC/fy15/cryptrec20030424_outrep.html`

[12] Trustees of Indiana University, "LAM/MPI Parallel Computing" `http://www.lam-mpi.org`, 18-Dec-2003.

[13] A.K. Lenstra, H.W., Lenstra Jr., M. Manasse, and J. Pollard, "The Number Field Sieve", Proceedings of *STOC'90*, pp.564-572, 1990.

[14] A.K. Lenstra, H.W. Lenstra (Eds.), "The development of the number field sieve", Lecture Notes in Mathematics (LNM) 1554, Springer-Verlag, 1993.

[15] A.K. Lenstra, et al., Factorization of RSA-130, `http://www.crypto-world.com/announcements/RSA130.txt`

[16] A.K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes, P.C. Leyland, "Factoring Estimates for a 1024-Bit RSA Modulus", *Asiacrypt 2003*, LNCS 2894, pp.55-74, Springer-Verlag, 2003.

[17] P.L.Montgomery, "A Block Lanczos Algorithm for Finding Dependencies over $GF(2)$," Proceedings of Eurocrypt95, LNCS 921, pp.106-120, 1995.

[18] P.L.Montgomery, "Distributed Linear Algebra," Presentation material for RSA2000, 2000. `http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc2000/montgomery.ppt`

[19] B.A. Murphy, "Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm", a doctor thesis of Australian National University, 1999. Available at `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/students.html`

[20] Phong Nguyen, "A Montgomery-like Square Root for the Number Field Sieve," Proceedings of ANTS-III, LNCS1423, pp.151-168, Springer–Verlag, 1998.