計算機代数入門

野呂 正行

February 27, 2001

Copyright ©2000 by Masayuki Noro. All rights reserved.

本書の LaTeX ソースファイルは OpenXM (http://www.math.sci.kobe-u.ac.jp/OpenXM/) の一部として公開されています. 著作権に関しては, OpenXM パッケージに添付の Copyright.generic ファイルの内容に従います.

Contents

1 はじめに **7**

2 準備 **9**

2.1 CPU の構造および動作 9

2.1.1 CPU についてのあらまし 9

2.1.2 整数演算命令 10

2.2 C 言語について 11

2.2.1 変数 11

2.2.2 式 12

2.2.3 文 12

2.2.4 関数 13

2.2.5 配列, ポインタ 13

2.2.6 構造体 14

3 整数 **17**

3.1 整数の表現 17

3.2 加算 18

3.3 減算 19

3.4 乗算 20

3.5 除算 21

3.6 冪 22

4 多項式 **25**

4.1 多項式の表現 25

4.1.1 再帰表現 25

4.1.2 分散表現 26

4.2 加減算 26

4.3 乗算 26
4.4 冪 27
4.5 除算, 剰余演算 27
4.6 Karatsuba アルゴリズム 27
4.7 除算を乗算で行うには? 29

5 多項式の GCD **31**
5.1 Euclid の互除法 31
5.2 拡張 Euclid 互除法 32
5.3 終結式 35
5.4 部分終結式 36
5.5 Modular アルゴリズム 38

6 多項式の因数分解 **41**
6.1 有限体 41
6.2 無平方分解 43
6.3 Berlekamp アルゴリズム 47
6.4 Cantor-Zassenhaus アルゴリズム 49
6.5 DDF (distinct degree factorization) 52
6.6 次数別因子の分解 53
6.7 整数係数一変数多項式の因数分解 55
6.8 多変数多項式の因数分解, GCD, 無平方分解 59
6.8.1 一般化された Hensel 構成 59
6.8.2 多変数多項式の因数分解 62
6.8.3 多変数多項式の無平方分解及び GCD 64
6.9 代数体上の因数分解 65
6.10 応用 — 有理関数の不定積分 70
6.10.1 有理部分の計算 70
6.10.2 対数部分の計算 72

7 グレブナ基底 **75**
7.1 代数方程式の解とイデアル 75
7.2 項順序, モノイデアル, グレブナ基底 76
7.3 Buchberger アルゴリズム 83

7.4 Term order の例 84

8 グレブナ基底の応用 87

8.1 イデアルに関する演算 87
8.2 剰余環, 次元 89
8.3 消去法 91
8.4 加群のグレブナ基底 92
8.5 例 : 双対曲線の計算 92

9 イデアルの分解 95

9.1 素イデアル, 準素イデアル, 準素イデアル分解 95
9.2 準素分解の概略 97
9.3 0 次元イデアルの準素分解 99
9.4 準素分解の例 102

10 グレブナ基底計算の効率化 103

10.1 不必要対の検出 104
10.1.1 冗長な基底の除去 104
10.1.2 0 に正規化される対の除去 105
10.2 正規化対の選択戦略 106
10.3 Trace lifting 107
10.4 斉次化 と trace lifting の組合せ 110
10.5 F_4 アルゴリズム 112
10.5.1 Symbolic preprocessing 112
10.5.2 F_4 アルゴリズム 114
10.5.3 Modular 計算による線形方程式の求解 116
10.5.4 例 117
10.5.5 まとめ 119

11 Change of ordering 121

11.1 FGLM アルゴリズム 121
11.2 Modular change of ordering 124
11.2.1 Modular 計算と線形代数によるグレブナ基底候補生成 124
11.2.2 グレブナ基底候補がグレブナ基底となる条件 125
11.2.3 candidate_by_linear_algebra() 129

11.3 タイミングデータ 133
11.3.1 Change of ordering 135
11.3.2 RUR 136

Chapter 1

はじめに

計算機代数は, 計算機上で, 代数的計算を正確に行なうことを目標としている.

計算機が実用化されて以来, 計算機上での計算とは, 固定長, すなわち計算機が提供するサイズの整数あるいは浮動小数を用いた計算を意味する場合がほとんどであった. 特に, 浮動小数を用いる, いわゆる数値計算の場合, 計算誤差は避けられず, 一つのアルゴリズムに対して, その結果の誤差の解析は必須であった. 実際, 通常の数値計算では誤差の累積が大き過ぎて, 実用には適さないとされるアルゴリズムも存在する. また, 数値計算は, 代数的計算に限らず, 代数方程式, 微分方程式などの近似解の計算などに広く用いられているが, それらの場合における, 代数方程式, 微分方程式は, 多くの場合, その数学的構造は重視されず, ある点における値を生成するブラックボックスとして用いられる. このような方法は, 広い範囲の入力に対してなんらかの近似解を生成することができる反面, 得られた解から, もとの方程式が本来持っていた数学的な性質を得ることは多くの場合困難である. 計算機代数は, 数学的対象を, 忠実に計算機上に表現し, 誤差のない計算を行なうことにより, 数学的に明確な意味をもつ結果を得ることを主眼としている. 当然, 数値計算に比較して, 扱える対象は限られる. 今のところ, 整数, 有理数およびそれらを係数とする多項式, 有理式が主な対象であるため, 計算機代数と呼ばれる. しかし, その及ぶ範囲は, 多項式の因数分解を初めとして, 多項式イデアル論の応用としての代数幾何学をカバーするまでになり, イデアルの準素イデアル分解が構成的に可能となっている. この分野における基本的なツールは, Buchberger により考案されたグレブナ基底である. これを含む概念が, standard basis として, 広中により Buchberger 以前に考案されていたが, 多項式環の場合に構成的な生成アルゴリズムを Buchberger が与えたことにより, 以上のことが計算機上で可能となったわけである.

以上のように, 計算機代数においては, 正確な計算ということが基本となっている. 最も基本的な単位である整数も, 必要なだけの桁数をすべて表現できる任意多倍長整

数 (bignum) という形で表現されなければならない. 当然ながら, 桁数が増えれば増えるほど, それらの間の演算にかかる時間は増える. 実際, bignum の計算時間が, 全体の計算時間の主要な部分を占める場合もしばしば生ずる. よって, 計算機代数においては, 計算機が提供する整数演算の性能が, 計算全体の効率を大きく左右する. 残念ながら, つい最近まで, 計算性能の向上は, いわゆる High Performace Computing のための浮動小数演算に対してのみ計られてきた. しかし, Pentium, SPARC などの CPU では整数演算性能の向上も図られており, 計算機代数システムの性能もそれに応じて飛躍的に向上してきた. さらに, 最近大きく注目されている暗号技術は, RSA 暗号, 楕円曲線暗号など, bignum あるいはその上の多項式演算をベースとしているものが多く, 計算機代数で培われたさまざな手法がそのまま役立つケースも多い. 計算機代数では, 多項式, 有理式など, 木構造をもつ対象が数多く用いられるため, 数の演算性能だけでなく, データ構造の表現方法, さまざまなアルゴリズムの効率の向上, あるいはメモリ管理などを深く考慮する必要がある.

本講では,

- 整数, 多項式の表現および四則演算アルゴリズム
- 多項式 GCD, 終結式, 中国剰余定理
- 多項式の因数分解, 有理関数の不定積分
- グレブナ基底, 代数方程式の求解, イデアルの準素分解

について述べる. ここで述べられるアルゴリズムのほとんどは, 筆者らが開発, 配布中の計算機代数システム Risa/Asir [31] 上に実装され, 用いられている.

Notation 1.1

$\mathbb{Z}$ 有理整数環

$\mathbb{N}$ 非負整数全体

$\mathbb{Q}$ 有理数体

$\mathbb{C}$ 複素数

$|F|$ F が数のとき絶対値, F が集合のとき元の個数 (濃度)

Chapter 2

準備

本章では, 以下で述べるさまざまなアルゴリズムを理解し, 計算機上に実装するために必要なことがらについて説明する.

2.1 CPU の構造および動作

どのような言語でアルゴリズムが記述されるにせよ, 最終的にはそのアルゴリズムは CPU 固有の命令として実行される. ここでは, CPU の基本的な構造, 動作について解説する.

2.1.1 CPU についてのあらまし

CPU に対する命令は, 32bit あるいは 64bit というある決まった大きさの bit 列で表現されている. CPU は, メモリ中に置かれた一連の命令列を順に読み出し, 対応する操作を実行する. CPU は, レジスタと呼ばれる特殊なメモリを持っている. レジスタは, CPU の構成に応じて 32bit あるいは 64 bit という大きさを持つ. その本数は通常数本から数十本程度と少ないが, 主記憶と比較して高速に読み書きでき, CPU のデータ操作の対象となる. CPU が行う基本操作には, 次のようなものがある.

- メモリに対する命令

  メモリからレジスタへのデータの読み込み, レジスタからメモリへのデータの書き出しを行う.

- 演算命令

  幾つかのレジスタに置かれたデータを用いて種々の演算を行う. 演算命令には, 整数演算命令, 浮動小数演算命令などがある.

- 分岐命令

  レジスタの値をテストして, その結果に応じてプログラム中の指定された場所にジャンプする.

2.1.2 整数演算命令

計算機代数においては, 誤差のない演算を実現するため整数演算を多用する. ここでは, 整数演算命令について簡単に解説する.

レジスタの bit 長 n に対し 2^n を B と書く. レジスタ上のデータは, 0 以上 $B-1$ 以下の非負整数を表すことができる. すなわち, a_i を 0 または 1 とすると, bit 列 $(a_{n-1}a_{n-2}\cdots a_1a_0)$ は

$$a = \sum_{i=0}^{n-1} a_i 2^i$$

なる整数 a を表す (二進数). 一方で, 負数は 2 の補数表示と呼ばれる方法で, やはり n bit のデータで表す. この場合 bit 列 $(a_{n-1}a_{n-2}\cdots a_1a_0)$ は

$$a = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

なる整数 a を表す. 例えば, -1 は $(11\cdots1)$ なる bit 列で表される. この方法によれば, n bit のデータで $-B/2 \le a \le B/2-1$ なる整数を表すことができる. 注意すべき点は, 加減乗算を法 B で行う場合, どちらで解釈しても結果は同一となることである. これは, 上記二つの表現の違いが 法 B での代表元の取り方の違いであることからわかる. 以下で, $r = a \bmod B$ は $0 \le r < B$ なる剰余を表す. 通常, CPU には次のような整数演算命令が用意されている.

- 加算

  $(a+b) \bmod B$

- 減算

  $(a-b) \bmod B$

- 比較

  if $a > b$ then 1 else 0

- 左シフト

  a の k bit 左シフト $\bmod B$

- 右シフト

  a の k bit 右シフト (右 k bit は捨てられる)

- bit 毎の and, or, exclusive or, not

  整数を bit 列と見て, 各 bit 毎に論理演算を行う.

シフト演算は, 論理演算に用いられるだけでなく, 整数の 2^k 倍, あるいは 2^k で割った商の計算に用いられる. これら以外に, CPU によっては次の演算が用意されている場合がある.

- 乗算

  $(a \times b) \bmod B$

- 倍精度乗算

  $(a \times b) \bmod B^2$

- 除算

  $a = qd + r$ なる q, r $(0 \leq r < d)$

- 倍精度除算

  $aB + b = qd + r$ なる q, r $(0 \leq r < d)$

これらの基本操作を用いて, 次節で述べる, 多倍長整数の四則演算が実現される.

2.2 C 言語について

本講では, アルゴリズムおよびデータ構造の計算機上での実装例を C 言語により示す場合がある. ここでは, C 言語の文法に関して, ごく基本的なことについて解説する. 詳細は, [23] などを参照されたい.

2.2.1 変数

変数とはデータを格納しておく場所である. データの種類には整数, 浮動小数をはじめとしてさまざまなもの (型) がある. 変数を使用するためには, 型および名前を宣言しなければならない. 例えば次は, `a`, `b`, `xyz` という名前の符号つき整数型変数, および `p`, `q` という名前の符号なし整数型変数の宣言である. 整数型変数のサイズが 32bit の

場合, int 型の変数は $-2^{31} \leq x \leq 2^{31}-1$, unsigned int 型の変数は$0 \leq x \leq 2^{32}-1$ なる整数 x を表現することができる.

```
int a,b,xyz;
unsigned int p,q;
```

2.2.2 式

変数および定数に対する一連の四則演算, 論理演算などをまとめて一つの式として記述することができる. 加, 減, 乗はそれぞれ +, -, * で表す. / は除算における商 (整数部分), % は剰余を表す. 式の中で用いることができる演算子には, 他に, << (左シフト), >> (右シフト), | (bit ごとの or), & (bit ごとの and), ^ (bit ごとの exclusive or) などがある. また, <, <=, >, >=, == (等しい), !=(等しくない) なども, 正しければ 1, そうでなければ 0 という値を返す 2 項演算子として用いることができる. 符号なし整数に対する加減乗算, 左シフト演算は, 整数型のサイズを n bit とするとき, 法 2^n で計算される. 例えば, 整数型が 32bit のとき,

```
unsigned int a,r;
a = 1<<31;
r = a+a;
```

を実行すると r は 0 となる.

2.2.3 文

文は C 言語の実行単位である. 文には, 式にセミコロン ; を付けた単文の他に, 幾つかの文を {, } で括った複文, if 文, for 文, while 文, switch 文などの制御を伴う文などがある. 次の例は, 小さな n に対して階乗を計算する文である.

```
unsigned int f;
for ( f = 1; n >= 1; n-- ) f *= n;
```

ここで, n-- は n = n-1, f *= n は f = f*n をそれぞれ意味する. この文は, 整数型が 32bit の場合, n が 12 以下でのみ正しい値を与える. これは, 13 以上では結果が 2^{32} 以上となるためであり, このような場合にも正しい結果を得るためには, 次節で述べるような任意多倍長整数の表現およびそれに対するアルゴリズムが必要となる.

2.2.4 関数

C 言語においては, プログラムは関数として記述される. 関数は, 引数 (入力) に対する操作をいくつかの文により記述し, 必要があれば結果を返す, という操作をまとめたものである. 先に述べた階乗計算を関数にすると次のようになる.

```
unsigned int factorial(unsigned int n)
{
    unsigned int f;
    for ( f = 1; n >= 1; n-- ) f *= n;
    return f;
}
```

この例では, 符号なし整数 n を受け取り, その階乗を符号なし整数として返す factorial という関数が宣言されている.

2.2.5 配列, ポインタ

任意多倍長整数は, メモリ上の十分な長さの連続領域として表現できる. このような領域は, C 言語では配列として表現される.

```
unsigned int a[10];
```

これは, 長さ 10 の符号なし整数配列 a を宣言している. 配列の各要素は, a[0], ···, a[9] により得られる. (添字が 0 から始まることに注意.) 次のプログラムは, 長さ 10 の配列の i 番目の要素に i を代入する.

```
unsigned int a[10];
int i;
for ( i = 0; i < 10; i++ ) a[i] = i;
```

このプログラムは次のようにも書ける.

```
unsigned int a[10];
int i;
unsigned int *p;
for ( i = 0, p = a; i < 10; i++, p++ ) *p = i;
```

p はポインタと呼ばれる型の変数である. ポインタは, メモリのアドレスを抽象化したものである. この例の場合には, p = a により最初 p は配列 a の先頭を指している. そして, i が 1 増えるたびに, p の指す位置も, 配列中で 1 要素ずつ先を指すように変化していく. すなわち, p++ による p のアドレスとしての変化量は unsigned int *p なる宣言により規定されるのである. また, *p = i は, p の指す領域の内容を i という値に書き換えることを意味する.

ポインタを導入する必要性にはさまざまなものがある. ここでは次の 3 つについて説明する.

- 動的なメモリ割り当て

  我々の演算対象である整数は, 計算の過程でいくらでも大きな値となっていく可能性がある. そのため, 例えば加算を行う関数は, 任意の大きさの引数に対して, 結果を保持するだけの領域を確保してそこに結果を書くという形が自然である. このとき, 記憶割り当てルーチンが返す値は, メモリ中のある領域の先頭アドレスであり, 受け取る側は必然的にポインタとして扱う必要がある.

- 木構造の表現

  多項式は, 後に述べるように木構造により表現される. その名の通り, 木は根から枝わかれしながらつながるデータを表現する. これは, メモリ上では, 他の領域を指すポインタを保持するデータ構造により表現するのが自然である.

- 関数呼び出し側の変数の書き換え

  C 言語では, 関数呼び出しは値による呼び出し (call by value) と呼ばれる方法で行われる. すなわち, 呼び出し側の変数を, 関数呼び出しの引数として渡しても, 呼び出された関数にはその値のみが渡されるため, 呼び出された関数は, その変数の中身を書き換えることができない. このような場合に, 引数として変数を指すポインタを与えることにより, その変数の中身を書き換えることが可能になる. 変数 a に対し, a を指すポインタの値は&a で得られる. & と * は互いに逆の操作となっている.

2.2.6 構造体

例えば多倍長整数を計算機上で表現する場合, 各桁を表す整数配列および符号, 桁数の情報は最低限必要である. これらはそれぞれ個別の変数に割り当てることも可能であるが, 一つの対象が複数の変数に別れて保持されるのは, その取扱いを複雑にし, ま

たプログラムの可読性の点からも好ましくない. このような場合に, 複数の要素を一つにまとめた構造体を用いることができる. 例えば, 整数は次のような構造体で表すことができる.

```
struct Integer {
  int n; /* 桁数 = |n|; n の符号がこの整数の符号 */
  unsigned int *digits; /* 各桁を表す配列へのポインタ */
};
```

もちろん, ポインタ `digits` は長さ `|n|` 以上の符号なし整数配列を指していなければならない.

Chapter 3

整数

3.1 整数の表現

計算機においては高々レジスタのビット長 (ワード長) の数に対する演算しか提供されず, それ以上の大きさの整数 (多倍長整数 あるいは bignum) に対する演算は

1. 整数を, ある整数 B により B 進表示する.
2. 各桁どうしの演算を CPU の整数演算機能により行なう.
3. これらの組合せにより元の整数どうしの演算結果を得る.

という形で行われる. B の値は, ワード長に近い大きさの 2 ベキとする場合が多い. これは,

- B を大きくすると B 進表現の桁数が少なくで済む
- B 以上の整数の上位, 下位の桁への分解が, ビットシフト演算で済む

という理由による. しかし, B を, 半ワード長より大きくとる場合, 乗除算において,

1. 単精度整数 $\times$ 単精度整数 $\rightarrow$ 倍精度整数
2. 倍精度整数 $\rightarrow$ 単精度整数 $\times$ 単精度整数 $+$ 単精度整数

という二つの演算が必要になる.

標準的な C 言語においては上記の演算は通常提供されておらず, アセンブリ言語によりこれらを記述する必要も生ずる. ただし, 最近広く使用されている `gcc` では `long long` あるいは `unsigned long long` 宣言により 64 bit 整数演算を用いることができ, それにより C 言語のみでこれらの演算が可能となる. 例えば 1. の演算は

```
typedef unsigned int UL;
typedef unsigned long long ULL;
UL a,b,ch,cl;
ULL c;
...
c = (ULL)a*(ULL)b;
ch = (UL)(c>>32);
cl = (UL)(c&(((ULL)1)<<32));
```

により, ch, cl に各々 64bit の乗算結果の上位, 下位 32bit が格納される. しかし, gcc の CPU 毎の実装の仕方によっては, 32 bit 乗除算が常に関数呼び出しになったりする場合もあるので, 確実な高速化のためにはやはりアセンブラあるいはインラインアセンブラを使うのも止むを得ない.

以下, B 進表現された二つの自然数

$$u = u_n B^n + u_{n-1} B^{n-1} + \cdots + u_0$$

$$v = v_m B^m + v_{m-1} B^{m-1} + \cdots + v_0$$

$(0 \leq u_i, v_j \leq B-1)$ に対し, 加減乗除演算を行うことを考える.

3.2 加算

加算は, 筆算を行うように下位の桁から繰り上がりとともに足していけばよい. しかし, 実際の計算機上での実現はそれほど自明ではない. B として計算機のワード長一杯 (32bit CPU なら $B = 2^{32}$) を選んだ場合, 次のようなアルゴリズムとなる.

アルゴリズム 3.1

Input : $u = \sum_{i=0}^{n-1} u_i B^i$, $v = \sum_{i=0}^{n-1} v_i B^i$

Output : $w = u + v$

$c \leftarrow 0$

for $i = 0$ to $n-1$

  $t \leftarrow u_i + v_i \bmod B$

  if $t < u_i$

    $t \leftarrow t + c \bmod B$

    $c \leftarrow 1$

else

$t \leftarrow t + c \bmod B$

if $t < c$ then $c \leftarrow 1$ else $c \leftarrow 0$

$w_i \leftarrow t$

$w_n \leftarrow c$

return $\sum_{i=0}^{n} w_i B^i$

このアルゴリズムに現れる $c \leftarrow a + b \bmod B$ は,

$$a + b \geq B \Rightarrow c \leftarrow a + b - B$$

を意味する. B がワード長一杯の場合, Section 2.1.2 で述べたように加算は B を法として行われる. これは, C 言語では,

```
unsigned int a,b,c;
...
c = a+b;
```

により実行できる. 繰り上がりは, $a + b \bmod B$ が a または b より小さくなっているかどうかで検出できるため, ここで述べたアルゴリズムにより加算が正しく実行できるのである.

3.3 減算

減算も, 筆算と同様の仕方で行う.

アルゴリズム 3.2

Input : $u = \sum_{i=0}^{n-1} u_i B^i$, $v = \sum_{i=0}^{n-1} v_i B^i$ $(u \geq v)$

Output : $w = u - v$

$b \leftarrow 0$

for $i = 0$ to $n - 1$

$t \leftarrow u_i - v_i \bmod B$

if $t > u_i$

$t \leftarrow t - b$

$b \leftarrow 1$

else

$t \leftarrow t - b \bmod B$

(B) if $t = B - 1$ then $b \leftarrow 1$ else $b \leftarrow 0$

$w_i \leftarrow t$

return $\sum_{i=0}^{n-1} w_i B^i$

このアルゴリズムに現れる $c \leftarrow a - b \bmod B$ は,

$$a - b < 0 \Rightarrow c \leftarrow a - b + B$$

を意味する. この操作も, C 言語では

```
unsigned int a,b,c;
...
c = a-b;
```

により実行される. 繰り下がりは, $a - b \bmod B$ が a より大きいかどうかで検出できる. (B) における $B - 1$ との比較は, ここでの繰り下がりが, $t = 0$ かつ $b = 1$ の場合にのみ生ずるためである.

3.4 乗算

乗算も筆算と同様の方法で計算できる.

アルゴリズム 3.3

Input : $u = \sum_{i=0}^{n-1} u_i B^i$, $v = \sum_{i=0}^{m-1} v_i B^i$

Output : $w = uv$

for $i = 0$ to $n - 1$

$w_i \leftarrow 0$

for $j = 0$ to $m - 1$

$c \leftarrow 0$

for $i = 0$ to $n - 1$

$t \leftarrow w_{i+j} + u_i v_j + c$

(M) $t = t_h B + t_l$ $(0 \le t_h, t_l < B)$ と分解

$w_{i+j} \leftarrow t_l$

$c \leftarrow t_h$

$w_{n+j} \leftarrow c$

(M) で計算される t は, $0 \leq, w_{i+j}, c < B$ (B 進 1 桁) の元より

$$t = w_{i+j} + u_i v_j + c \leq (B-1) + (B-1)^2 + B - 1 = B^2 - 1$$

より B 進で高々2 桁の数である. よって t_h, t_l は B 進 1 桁となり, 帰納的に, w_{i+j}, c が常に B 進 1 桁となることが分かる. ただし, $u_i v_j$ の計算には倍精度整数乗算が必要となる.

このアルゴリズムでは, 入力の桁数の積 mn に比例する手間がかかる. 桁数の大きい数の積の手間を減らす工夫として Karatsuba 法および FFT 法がある. これらのうち, 多項式に関する Karatsuba 法については Section 4.6 で述べるが, 整数に対しても全く同様のアルゴリズムが適用できる. (B 進整数は, B に関する多項式とみなせることに注意.) 一般に, Karatsuba 法はいくらかの overhead を伴うため, 実際には, ある桁数 (しきい値) を設定して, しきい値以上の桁数では Karatsuba 法, それ以下ではここで述べたアルゴリズムを適用することで, さまざまな桁数の数に対して高速な実装が得られる.

3.5 除算

整数除算も基本は筆算と同様で, 商の上位の桁から求めていく. 簡単のため, 被除数, 除数をそれぞれ

$$u = u_{n+1} B^{n+1} + \cdots + u_0$$

$$v = v_n B^n + \cdots + v_0$$

($0 \leq u_i, v_i < B$, $u/v < B$) とする. 問題は, 商を如何に正確に見積もることができるかである. 次に紹介するのは Knuth [24, Section 4.3] に述べられているもので, 商の候補を, 除数の上位 2 桁, 被除数の上位 3 桁から求める. $\lfloor a \rfloor$ は, a を越えない最大の整数とする. $q = \lfloor u/v \rfloor$ である.

命題 3.4 $\hat{q} = \min(B-1, \lfloor (u_{n+1}B + u_n)/v_n \rfloor)$ とする. この時, $v_n \geq \lfloor B/2 \rfloor \Rightarrow q \leq \hat{q} \leq q+2$.

命題 3.4 により, $v_n \geq \lfloor B/2 \rfloor$ の元で, 真の値 q より高々 2 だけ大きい近似値 $\hat{q}$ が得られる.

命題 3.5 $\hat{q} \geq q$ かつ $\hat{q}(v_n B + v_{n-1}) > u_{n+1}B^2 + u_n B + u_{n-1} \Rightarrow q \leq \hat{q} - 1$.

$\hat{q} = q + 2$ ならば常に命題 3.5 の不等式が成立するから, このチェックを高々 2 回行なうことで, $\hat{q} = q$ または $\hat{q} = q + 1$ とできる. このチェックを行ったのち, なお $\hat{q} \neq q$ となる可能性については次の命題が成り立つ.

命題 3.6 $\hat{q} > q$ かつ $\hat{q}(v_n B + v_{n-1}) \leq u_{n+1}B^2 + u_n B + u_{n-1} \Rightarrow u \bmod v \geq (1 - 2/B)v$.

すなわち, ランダムに u, v を選んだとき, $\hat{q} \neq q$ となる確率は $2/B$ 程度となり, $B = 2^{32}$ の場合には極めて小さい確率でしか起こらないことが分かる. この $\hat{q}$ により, $\hat{r} = u - \hat{q}v$ を実際に計算して, もし $\hat{r}$ が負ならば $q = \hat{q} - 1$ であり, $r = u - qv = \hat{r} + v$ となる. 命題 3.5 の不等式は

$$\hat{q}v_{n-1} > (u_{n+1}B + u_n - \hat{q}v_n)B + u_{n-1}$$

と書けるから, このチェックで必要になる整数演算は高々倍精度で十分である. まとめると,

- $\hat{q} = q$ または $\hat{q} = q + 1$ で, 後者となる確率は極めて小さい.
- 整数乗除算は倍精度乗除算で十分である.

前者は, 実際に足し戻しが必要になる場合が極めて小さいことを意味し, 除算の効率向上につながるが, 反面, 足し戻しを行なう部分のバグとりに苦労することなる. また, 後者は, 倍精度整数乗除算さえ実現されていれば, このアルゴリズムは実現可能であることを示している. 上記の方法で, $v_n \geq \lfloor B/2 \rfloor$ なる制限が与えられているが, これは, あらかじめ u, v に $d = \lfloor B/(v_n + 1) \rfloor$ を掛けておけば満たされる. 商は変化せず, 剰余は d で割ればよい. また, B が 2 の冪の場合には, d と値として 適当な 2 の冪がとれる. この場合, d による乗除算は, シフト演算となり都合がよい.

3.6 冪

整数の冪乗の計算は, 次のような 2 通りのアルゴリズムで行うことができる.

アルゴリズム 3.7
Input : $u \in \mathbb{Z}, e \in \mathbb{N}$
Output : $w = u^e$
$(k_m k_{m-1} \cdots k_0)_2 \leftarrow e$ の 2 進表示 $(k_m = 1)$
$t \leftarrow u$

$w \leftarrow 1$
for $i = 0$ to $m\{$
(a) if ($k_i = 1$) then $w = wt$
    if ($i = m$) then return w
    $t \leftarrow t^2$
$\}$

命題 3.8 アルゴリズム 3.7 は u^e を出力する.

[証明] (a) において, $t = u^{2^i}$ より, このアルゴリズムは, $k_i = 1$ なる i に対する u^{2^i} の積を出力する. この値は u^e に等しい. □

アルゴリズム 3.9
Input : $u \in \mathbb{Z}, e \in \mathbb{N}$
Output : $w = u^e$
$(k_m k_{m-1} \cdots k_0)_2 \leftarrow e$ の 2 進表示 $(k_m = 1)$
$w \leftarrow 1$
for $i = m$ to $0\{$
(a) $w \leftarrow w^2$
    if ($k_i = 1$) then $w = wu$
$\}$
return w

命題 3.10 アルゴリズム 3.9 は u^e を出力する.

[証明] $m = 0$ の時 $w = u$ となり正しい. $m-1$ まで正しいとする. $e_1 = (k_m k_{m-1} \cdots k_1)$ を m bit 数とみなせば, 帰納法の仮定により, $i = 0$ で (a) において $w = u^{e_1}$. このとき, このアルゴリズムは $w^2 \cdot u^{k_0} = u^{2e_1 + k_0}$ を出力するが, これは u^e に等しい. □
これらのアルゴリズムは, e 乗を, それぞれ $\log_2 e$ 回程度の 2 乗算と乗算で実行できる. このアルゴリズムは, 整数の冪乗に限らず, あらゆる場合に用いられる汎用的なものである. 前者のアルゴリズムは,

- 指数 e の各ビットを右から左にスキャンする.
- 常に掛ける数は同じ.
- 保持すべき途中結果は w のみ.

後者は

- 指数 e の各ビットを左から右にスキャンする.
- 掛ける数が変化していく.
- 保持すべき途中結果は w, t の両方.

という特徴を持ち, それぞれ一長一短がある.

Chapter 4

多項式

4.1 多項式の表現

整数の場合その一部を取り出して用いることはほとんどないが, 多項式は, その一部 (例えば係数) を多項式あるいは数として取り出す必要がしばしばある. 従って, その表現も, 用途に応じてさまざまなものが考えられる. ここでは, それらの中で再帰表現と 分散表現 について述べる.

4.1.1 再帰表現

再帰表現とは, 多項式に現れる変数に順序をつけ, その順序に従って, 多項式を一変数多項式とみなして, その各係数を, この変数順序に従ってさらに分解していく表現である.

例 **4.1** $(x+y+z)^2 = 1\cdot x^2 + (2\cdot y + (2\cdot z))\cdot x + ((2\cdot z)\cdot y + (1\cdot z^2))$

この表現は, 実際のシステムにおいては最も一般的に用いられる. 特に, 多項式を, ある変数に関する一変数多項式とみなして進行するアルゴリズムに対して有効である. このようなアルゴリズムには, 多項式の四則演算を初めとして, 多項式の GCD, 因数分解など, 基本的で重要なものが多く, 従って, 内部表現として再帰表現を採用することは, システムの効率の点から見て自然である. ただし, 再帰表現においては, 主変数でない変数に関する係数を取り出す場合に, 主変数に関する場合に比較して多くの時間がかかるのが難点である. このため, 計算に先だって, 異なる変数順序による再帰表現に変換することがしばしば行なわれるが, 変数, 項の個数が多い場合には, 多くの時間がかかる場合もある.

再帰表現を計算機上で実現する場合, 文字通り再帰的な表現となる. ここで, 主変数を一つ決めれば, その多項式は, その主変数に関する指数と係数のペアの並びとし

て表現される.

4.1.2 分散表現

分散表現とは, 多項式を, 単項式の和として表現する方式である. ここで, 単項式とは, 変数の冪積と係数の積である.

例 **4.2** $(x+y+z)^2 = 1 \cdot x^2 + 2 \cdot xy + 2 \cdot xz + 1 \cdot y^2 + 2 \cdot yz + 1 \cdot z^2$

この場合, 各単項式の順序に不定性があるが, 各変数は平等に扱われる. この表現は, 後述する グレブナ基底計算において最も好都合な表現形式であり, そこでは単項式の間の順序が本質的な役割を果たす. これについては, グレブナ基底のところで詳しく述べる.

多項式は, 計算機上ではリスト (木構造) あるいは配列で表現される. 配列表現の多項式に関しては, 同一次数の係数の取り出しが容易のため, 四則演算のアルゴリズムは自明である. しかし, リスト表現の場合, 四則演算は, 項の順序を比較しながら, もし順序が等しい場合には係数に対する演算を行うというもので, ソーティングの変形と考えられる. ただし, 対象となる多項式は既にソートされているため, その性質を利用して効率よく演算を行うことが必要である.

4.2 加減算

配列表現の場合, 加減算は自明である.

リスト表現の場合, 演算結果リストを空に初期化し, 先頭の項の次数を比較し, 次数の大きい方の項をそのまま, 演算結果のリストに繋ぎ, もし次数が等しい場合には, 係数どうしを演算し, それが 0 でない場合に演算結果リストに繋ぐ, という操作を繰り返す.

4.3 乗算

乗算は, 分配法則により一方の多項式の各単項式ともう一方の多項式の積の和となるが, 単項式と多項式の積は, 係数は係数どうしの積で, 各項の次数は単なる平行移動となるため, 一方の多項式の項の個数だけの多項式の和となる.

4.4 冪

多項式の冪の計算は, 数の場合とは異なり演算の方法により大きく効率が異なる.

整数演算の項で述べた冪の 2 進計算法では, 見掛けの乗算の回数は冪指数の対数程度であるが, 実際には中間に現れる多項式の項の個数が増え, かつ途中の積における次数の等しい各項の間の演算回数も増えるため, 実際の演算時間は急激に増加する. このため, 一見効率が悪そうに見える, 二項定理を用いる方法の方が効率よく冪を計算できる. 即ち, 冪乗すべき多項式を 2 つに分け, 二項定理により冪を計算するのである. この際二項係数が必要になるが, ある程度の大きさまで予めテーブルにしておいても良いし, あるいはその都度計算しても, 素朴な方法に比較して充分高速である.

4.5 除算, 剰余演算

除算, 剰余演算は, ある変数 (主変数) を決めて, 主変数に関する 1 変数多項式として行う. 体 K 上の一変数多項式の除算は次のアルゴリズムで計算できる.

アルゴリズム 4.3

Input : $f, g \in K[x], g \neq 0$

Output : $f = qg + r$, $q, r \in K[x]$, $\deg(r) < \deg(g)$ なる q, r

$q \leftarrow 0, \quad r \leftarrow f$

while ($\deg(r) \geq \deg(g)$) {

$\quad t \leftarrow \mathrm{lc}(r)/\mathrm{lc}(g) \cdot x^{\deg(r)-\deg(g)}$

$\quad r \leftarrow r - tg, \quad q \leftarrow q + t$

}

return (q, r)

4.6 Karatsuba アルゴリズム

ここでは, 1 変数多項式の高速乗算法およびその応用について述べる. 既に述べた方法では, 2 つの n 次の密な多項式の積計算は $O(n^2)$ の計算量が必要である. これに対し, Karatsuba アルゴリズムは $O(n^{\log_2 3}) \simeq O(n^{1.58})$ の計算量で積を計算する. さらに計算量の小さい FFT アルゴリズムについては [24, Section 4.3] に詳しい解説がある.

まず, 1 次式 $f = ax + b$, $g = cx + d$ の積は, 係数環における積 3 回で実行できることに注意する. すなわち,

$$fg = (ax + b)(cx + d) = acx^2 + ((a - b)(d - c) + ac + bd)x + bd$$

で, ここに現れる係数環の積演算は, ac, bd, $(a-b)(d-c)$ の 3 回である. これを, 高次の場合にも再帰的に適用する.

命題 **4.4** 2^m-1 次式どうしの積は, $O(3^m)$ の計算量で計算できる.

[証明] 2^m-1 次式どうしの計算コストを $T(m)$ とする. 係数環における和, 積のコストをそれぞれ A, M とする. f, g を 2^m-1 次式とする.

$$f = f_1x^m + f_2, \quad g = g_1x^m + g_2 \quad (\deg(f_1), \deg(f_2), \deg(g_1), \deg(g_2) < m)$$

と書くと,

$$fg = (f_1g_1x^{2m} + ((f_1-f_2)(g_2-g_1) + f_1g_1 + f_2g_2)x^m + f_2g_2$$

ここで fg の計算コストは $3T(m-1) + 4\cdot 2^mA$, すなわち $T(m) = 3T(m-1) + 4\cdot 2^mA \quad (m \le 1)$. さらに, $T(0) = M$ から $T(m) = (M+8A)3^m - 8A\cdot 2^m$. □

この命題より, n 次式どうしの積は, $O(n^{\log_2 3})$ の計算量で計算できることが分かる. 更に詳しく, 通常の $O(n^2)$ アルゴリズムと比較してみよう. 2^m-1 次式どうしの通常のアルゴリズムによる計算コストを $T_0(m)$ とすれば, $T_0(m) = M2^{2m} + A(2^m-1)^2$ である.

$$T_0(m) - T(m) \ge M2^{2m} + A(2^m-1)^2 - ((M+8A)3^m - 8A\cdot 2^m)$$

で, 右辺の $m = 0, \cdots, 6$ に対する値は次のようになる.

m	0	1	2	3	4	5	6
右辺	0	M-7A	7M-31A	37M-103A	175M-195A	781M-727A	3367M-1351A

一般に, $M > A$ だから $m \ge 5$ すなわち 31 次式以上の積に対しては, Karatsuba 法は常に高速であり, M が A に比べて大きい場合ほど, 低い次数から Karatsuba 法が効果的であることが分かる. 更に, 計算量のオーダの違い ($O(n^2)$ と $O(n^{\log_2 3})$) により, 高次程 Karatsuba 法が高速になり, 例えば $M/A = 5$ の時, 2^m-1 次式の積の通常法と Karatsuba 法の計算コストの比は次のようになる.

m	0	1	2	3	4	5	6	7	8	9	10
計算コスト比	1	1.1	0.96	0.78	0.61	0.48	0.37	0.28	0.21	0.16	0.12

すなわち, 100 次で 1.5 倍, 1000 次式で 8 倍程度の差がつくことになる. Karatsuba 法における, 多項式の分割, 関数呼び出しの手間などがかかるため, 実際にはこの数字を達成することは難しいが, 多項式の積に関しては, Karatsuba 法は十分実用的であると言える.

4.7 除算を乗算で行うには?

既に述べた多項式除算は, 次数 n について $O(n^2)$ の計算量を必要とする. これを, より少ない手間で計算するために, 乗算を用いて除算を行うことを考える.

命題 4.5 $f = \sum_{i=0}^{n} a_i x^i$ を, $a_0 \neq 0$ なる多項式とする. $g_0 = 1/a_0, \quad g_i = 2g_{i-1} - g_{i-1}^2 f \bmod x^{2^i}$ とすれば, $g_i f \equiv 1 \bmod x^{2^i}$.

定義 4.6 n 次多項式 f に対し, $f^* = x^n f(1/x)$ と定義する.

命題 4.7 f, g を各々 n, m 次 $(n \geq m)$ 多項式とし, $f = gq + r \quad (\deg(r) < m)$ とする. この時 $tg^* \equiv 1 \bmod x^{n-m+1}$ なる t により $q = (tf^* \bmod x^{n-m+1})^*$.

[証明] $f = gq + r$ より $x^n f(1/x) = x^m g(1/x) x^{n-m} q(1/x) + x^n r(1/x)$. すなわち $f^* = g^* q^* + x^{n-\deg(r)} r^*$. ここで, $\deg(r) < m$ より $n - \deg(r) \leq n - m + 1$. よって $f^* \equiv g^* q^* \bmod x^{n-m+1}$. よって $tf^* \equiv q^* \bmod x^{n-m+1}$ となるが, $\deg(q) = n - m$ より $q = (tf^* \bmod x^{n-m+1})^*$. □
$r = f - gq$ より, q, r が何回かの乗算で計算できることがわかる.

- t を求める際に必要な乗算の回数は $\log_2 (n - m)$ 程度,
- t が求まっていれば, q, r の計算には乗算 2 回で済む.
- 乗算に Karatsuba 法 (あるいは FFT 法) を用いることにより $O(n^2)$ より真に少ない手間で除算が計算できる.
- g を法とする演算のように, g が固定の場合, t を 1 回計算しておけばよいから, n, m が大きい時には有効. (50 次程度から効いてくる.)

Chapter 5

多項式の GCD

5.1 Euclid の互除法

R を UFD (Unique Factorization Domain; 一意分解整域) とし, R における GCD の計算が可能であるとする. このとき, 多項式環 $R[x]$ においても GCD の計算が可能となる. R の商体を $Q(R)$ と書く.

定義 5.1

$$f = \sum_{i=0}^{n} (a_i/b_i)x^i \in Q(R)[x] \quad (a_i, b_i \in R; \mathrm{GCD}(a_i, b_i) = 1)$$

に対し f の容量 $\mathrm{cont}(f)$ を

$$\mathrm{cont}(f) = \mathrm{GCD}(a_0, \cdots, a_n)/\mathrm{LCM}(b_0, \cdots, b_n),$$

f の **primitive part** $\mathrm{pp}(f)$ を

$$\mathrm{pp}(f) = f/\mathrm{cont}(f)$$

で定義する. $\mathrm{cont}(f)$ が R の単元となるような f を原始多項式と呼ぶ.

命題 5.2 $\mathrm{pp}(f)$ は原始的.

[証明] $G = \mathrm{GCD}(a_0, \cdots, a_n), L = \mathrm{LCM}(b_0, \cdots, b_n)$ と書く. $\mathrm{pp}(f) = \sum_i L/b_i \cdot a_i/G \cdot x^i$ より, $\mathrm{cont}(\mathrm{pp}(f)) = \mathrm{GCD}(L/b_0 \cdot a_0/G, \cdots) \in R$. 素元 p が右辺を割るとする. もし, ある j に対し $p|(a_j/G)$ ならば, $i \neq j$ なる任意の i に対し $p \not| (a_i/G)$ より $i \neq j$ のとき $p|(L/b_i)$. 一方, 任意の p に対し $p \not| (L/b_k)$ なる k は存在するから $k = j$. この時, $p|b_j$ となり $\mathrm{GCD}(a_j, b_j) = 1$ に反する. よって全ての i に対し, $p|(L/b_i)$ これは不可能. よって $\mathrm{cont}(\mathrm{pp}(f)) = 1$. □

補題 5.3 [Gauss] $f, g \in Q(R)[x]$ に対し, $\mathrm{cont}(fg) = \mathrm{cont}(f)\mathrm{cont}(g)$.

命題 5.4 $f, g \in R[x]$ に対し, $\mathrm{GCD}(f,g) = \mathrm{GCD}(\mathrm{cont}(f), \mathrm{cont}(g))\mathrm{GCD}(\mathrm{pp}(f), \mathrm{pp}(g))$ また, $f, g \in R[x]$ を原始多項式とすると, $\mathrm{GCD}(f,g) = \mathrm{pp}(\mathrm{GCD}_{Q(R)}(f,g))$ ここで, $\mathrm{GCD}_{Q(R)}(f,g)$ は, 体 $Q(R)$ 上の一変数多項式環における GCD を意味する.

以上により, 多項式の GCD は, 係数環の GCD および, 係数環の商体での GCD により計算できることがわかった. しかし, 実際に商体での演算を用いることは, 無駄な GCD 計算を増やすだけとなる. 従って, まず, 多項式の除算で, 商体の除算が現れないものを考える.

定義 5.5 $f, g \in R[x], \deg(f) \geq \deg(g)$ に対し f の g による擬剰余 (pseudo-remainder) $\mathrm{prem}(f,g)$ を

$$\mathrm{prem}(f,g) = \mathrm{remainder}(\mathrm{lc}(g)^{\deg(f)-\deg(g)+1} f, g)$$

と定義する. ここで, $\mathrm{remainder}(f,g)$ は, 通常の多項式除算である.

容易にわかるように, $\mathrm{prem}(f,g)$ の計算の途中に現れる除算は, 全て R 上の整除である. これを用いて, Euclid の互除法を記述すると次のようになる.

アルゴリズム 5.6
Input : 原始的多項式 $f_1, f_2 \in R[x]$, $\deg(f_1) \geq \deg(f_2)$
Output : $\mathrm{GCD}(f_1, f_2)$
$i \leftarrow 1$
do {
  $f_{i+2} \leftarrow \mathrm{prem}(f_i, f_{i+1})$
  if ($f_{i+2} = 0$) then return $\mathrm{pp}(f_{i+1})$
  $i \leftarrow i + 1$
}

定義 5.7 f_1, f_2 に対し, $\beta_i f_{i+1} = \mathrm{remainder}(\alpha_i f_{i-1}, f_i)$ なる除算により生成される列 $\{f_i\}$ を多項式剰余列と呼ぶ.

5.2 拡張 Euclid 互除法

体 K 上の一変数多項式環 $K[x]$ は PID (Principal Ideal Domain; 単項イデアル整域) である. この場合, $f, g \in K[x]$ に対し, イデアル (f,g) の生成元が, $\mathrm{GCD}(f,g)$ となる.

すなわち $a, b \in K[x]$ が存在して,

$$af + bg = \mathrm{GCD}(f, g)$$

と書ける. この係数 a, b は, Euclid の互除法の変形により求めることができる.

アルゴリズム 5.8
Input : 原始的多項式 $f_1, f_2 \in K[x], \deg(f_1) \geq \deg(f_2)$
Output : $\mathrm{GCD}(f_1, f_2)$ および $af_1 + bf_2 = \mathrm{GCD}(f_1, f_2)$ なる $a, b \in K[x]$
$a_1 \leftarrow 1$, $a_2 \leftarrow 0$, $b_1 \leftarrow 0$, $b_2 \leftarrow 1$
$i \leftarrow 1$
do {
  $\deg(f_i - q_i f_{i+1}) < \deg(f_{i+1})$ なる q_i を求める
  $f_{i+2} \leftarrow f_i - q f_{i+1}$; $a_{i+2} \leftarrow a_i - q a_{i+1}$; $b_{i+2} \leftarrow b_i - q b_{i+1}$
  if ($f_{i+2} = 0$) then return $\{f_{i+1}, a_{i+1}, b_{i+1}\}$
  $i \leftarrow i + 1$
}

ここで現れた a_i, b_i に対し,

$$\deg(a_i) \leq \deg(f_2) - \deg(f_{i-1}), \quad \deg(b_i) \leq \deg(f_1) - \deg(f_{i-1})$$

であることわかる. 特に, $f = f_i = \mathrm{GCD}(f_1, f_2)$ ならば

$$\deg(a_i) < \deg(f_2) - \deg(f), \quad \deg(b_i) < \deg(f_1) - \deg(f).$$

命題 5.9 $f, g \in K[x]$ に対し, $h = \mathrm{GCD}(f, g)$ とすると, $af + bg = \mathrm{GCD}(f, g)$ かつ $\deg(a) < \deg(g) - \deg(h), \quad \deg(b) < \deg(f) - \deg(h)$ なる $a, b \in K[x]$ は存在して一意的.

[証明] 一意性のみ示せばよい. h で両辺を割って, $h = 1$ としてよい.

$$af + bh = 1, \quad a_1 f + b_1 g = 1,$$

$$\deg(a) < \deg(g), \quad \deg(b) < \deg(f), \quad \deg(a_1) < \deg(g), \quad \deg(b_1) < \deg(f)$$

とすると, $(a - a_1)f + (b - b_1)g = 0$ かつ $\mathrm{GCD}(f, g) = 1$ より $g | (a - a_1)$. $\deg(a - a_1) < \deg(g)$ より $a - a_1 = b - b_1 = 0$.□
$af + bg = 1$ ならば $af \equiv 1 \bmod g$ すなわち, 剰余環 $K[x]/(g)$ において $a \bmod g =$

$(f \bmod g)^{-1}$ である. 剰余環における逆元計算は, 計算機代数において頻繁に必要となり, 特に後で述べる modular アルゴリズムにおいて極めて重要な役割を演ずる.

この命題は, 次のように一般化できる.

命題 5.10 $f = \prod_{i=1}^{n} f_i \in K[x]$ に対し, f_i が互いに素であるとする. このとき,

$$\sum_{i=1}^{n} e_i(f/f_i) = 1, \quad \deg(e_i) < \deg(f_i)$$

なる $e_i \in K[x]$ が一意的に存在する.

[証明] 帰納法により, $\sum_{i=1}^{n} E_i(f/f_i) = 1$ なる E_i が存在することが分かる. $E_i = q_i f_i + e_i$ $(\deg(e_i) < \deg(f_i))$ と除算を行うと,

$$f \sum_{i=1}^{n} q_i + \sum_{i=1}^{n} e_i(f/f_i) = 1.$$

ここで, $\deg(\sum_{i=1}^{n} e_i(f/f_i)) < \deg(f)$ より $\sum_{i=1}^{n} q_i = 0$ が成り立つから,

$$\sum_{i=1}^{n} e_i(f/f_i) = 1, \quad \deg(e_i) < \deg(f_i).$$

このような e_i は $\bmod f_i$ で一意的だから, $\deg(e_i) < \deg(f_i)$ では一意に決まる. □

命題 5.11 $f = \prod_{i=1}^{n} f_i \in K[x]$ に対し, f_i が互いに素であるとする. $\deg(g) \leq \deg(f)$ ならば,

$$\sum_{i=1}^{n} e_i(f/f_i) = g \quad \deg(e_i) \leq \deg(f_i)$$

なる $e_i \in K[x]$ が存在する. 特に, $\deg(g) < \deg(f)$ ならば, $\deg(e_i) < \deg(f_i)$ とできて, e_i は一意的.

[証明] 前命題と同様に, $\sum_{i=1}^{n} E_i(f/f_i) = g$ なる E_i が存在する. $E_i = q_i f_i + e_i$ $(\deg(e_i) < \deg(f_i))$ と除算を行うと,

$$cf + \sum_{i=1}^{n} e_i(f/f_i) = (e_1 + cf_1)(f/f_1) + \sum_{i=2}^{n} e_i(f/f_i) = g, \quad c = \sum_{i=1}^{n} q_i$$

ここで, $\deg(\sum_{i=1}^{n} e_i(f/f_i)) < \deg(f)$, $\deg(g) \leq \deg(f)$ より $\deg(cf) \leq \deg(f)$ すなわち $\deg(c) \leq 0$. よって $\deg(e_1 + cf_1) \leq \deg(f_1)$. 特に, $\deg(g) < \deg(f)$ ならば $c = 0$ より $\deg(e_1 + cf_1) = \deg(e_1) < \deg(f_1)$. 一意性も前命題と同様. □

系 5.12 (部分分数分解)

$d, n \in K[x]$ $(\deg(n) < \deg(d))$ とし, $d = \prod_{i=1}^{n} d_i^i$ を d の無平方分解とすれば, $r_{ij} \in K[x]$ $(1 \leq j \leq i, \deg(r_{ij}) < \deg(d_i))$ が存在して,

$$n/d = \sum_{i=1}^{n} \sum_{j=1}^{i} r_{ij}/d_i^j.$$

[証明] $D_i = d_i^i$ とおけば, 前命題により $\deg(E_i) < \deg(D_i)$ なる E_i が存在して $n = \sum_{i=1}^{n} E_i(d/D_i)$. これから $n/d = \sum_{i=1}^{n} E_i/d_i^i$. E_i を d_i の冪で展開すれば, $\deg(E_i) < \deg(d_i^i)$ より $E_i = \sum_{j=1}^{i} r_{ij} d_i^{(i-j)}$, $\deg(r_{ij}) < \deg(d_i)$ と書ける. よって $E_i/d_i^i = \sum_{j=1}^{i} r_{ij}/d_i^j$. □

5.3 終結式

R を可換環とする.

定義 5.13

$$a(x) = \sum_{i=0}^{n} a_i x^i, \quad b(x) = \sum_{i=0}^{m} b_i x^i \quad (a_i, b_i \in R)$$

に対し, a, b の Sylvester 行列 $Syl(a, b)$ を次で定義する.

$$Syl(a,b) = \begin{bmatrix} a_n & a_{n-1} & \cdots & a_0 & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & a_n & a_{n-1} & \cdots & a_0 \\ b_m & b_{m-1} & \cdots & b_0 & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & b_m & b_{m-1} & \cdots & b_0 \end{bmatrix}$$

a, b の終結式 $\mathrm{res}_x(a, b)$ を $\mathrm{res}_x(a, b) = \det(Syl(a, b))$ で定義する.

補題 5.14 $\deg(s) < \deg(b), \deg(t) < \deg(a)$ なる $s, t \in R[x]$ が存在して $sa + tb = 0$
$\Leftrightarrow s(x) = \sum_{i=0}^{m-1} s_i x^i, \quad t(x) = \sum_{i=0}^{n-1} t_i x^i$ と書くとき

$$[s_{m-1}, \cdots, s_1, s_0, t_{n-1} \cdots, t_1, t_0] Syl(a, b) = [0, \cdots, 0] \tag{5.1}$$

系 5.15 R が整域とする. $\deg(s) < \deg(b), \deg(t) < \deg(a)$ なる $s, t \in R[x] \setminus \{0\}$ が存在して $sa + tb = 0 \Leftrightarrow \mathrm{res}_x(a, b) = 0$

命題 5.16 R が UFD とする. ($R[x]$ において GCD が well defined.) $a, b \in R[x]$ に対し,

$$\deg(\mathrm{GCD}(a, b)) > 0 \Leftrightarrow \mathrm{res}_x(a, b) = 0.$$

$\Rightarrow$
$g = \mathrm{GCD}(a,b)$, $\deg(g) > 0$ とすると, $(b/g)\cdot a + (a/g)\cdot b = 0$. b/g, a/g は (5.1) の 0 でない解を構成するから, 系より $\mathrm{res}_x(a,b) = 0$.
$\Leftarrow$
$\mathrm{res}_x(a,b) = 0$ とすると, 系より (5.1) の 0 でない解が存在する. すなわち, $\deg(s) < \deg(b), \deg(t) < \deg(a)$ なる s,t が存在して $sa+tb = 0$. $\mathrm{GCD}(a,b)$ が定数なら $a|t$ となるが, $\deg(t) < \deg(a)$ より不可能. よって $\mathrm{GCD}(a,b)$ は定数でない.□

5.4 部分終結式

アルゴリズム 5.6 は, 確かに商体上の演算を必要としないが, 係数の膨張が著しい. 次の例を考えてみる.

例 **5.17**

$$R = \mathbb{Z}, \quad f = x^8 + x^5 + 1, \quad g = 3x^6 + 1$$

この場合, 途中で生成される擬剰余は,

$$27x^5 - 9x^2 + 27$$
$$729x^3 - 2187x + 729$$
$$-13947137604x^2 + 94143178827x - 20920706406$$
$$582295034461169322002535 3x - 1293988965469265160005634$$
$$-233531910092827408511917946933862161420003868170076721 13424$$

となり, 結果は $\mathrm{GCD}(f,g) = 1$ だが係数の長さがステップ毎に約 2 倍ずづ増えていることが分かる.

実際には, 上の例では, 途中で生成される擬剰余は原始的でなく, 容量を除くと次のようになる.

$$3x^5 - x^2 + 3$$
$$x^3 - 3x + 1$$
$$-4x^2 + 27x - 6$$
$$9x - 2$$
$$-1$$

この場合, 擬剰余の容量を実際に GCD を用いて計算したが, 実はある程度の部分は, 自動的に取り除くことができる. このため, 部分終結式 (subresultant) を導入する.

定義 5.18 $f=\sum_{i=0}^{n}a_ix^i, g=\sum_{i=0}^{m}b_ix^i$ とするとき, f,g の j 次の部分終結式 $S_j(f,g)$ を次の行列式で定義する.

$$S_j(f,g)=\begin{vmatrix} a_n & a_{n-1} & \cdots & \cdots & \cdots & x^{m-j-1}f \\ & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & a_n & \cdots & a_{j+1} & f \\ b_m & b_{m-1} & \cdots & \cdots & \cdots & x^{n-j-1}g \\ & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & b_m & \cdots & b_{j+1} & g \end{vmatrix}$$

特に, $S_0(f,g)=\mathrm{res}_x(f,g)$ (終結式) となる.

定義 5.19 $f\sim g$ とは $\mathrm{pp}(f)=\mathrm{pp}(g)$ なること.

命題 5.20 [26] $\deg(f_1)\geq\deg(f_2)$ とする. f_1,f_2 から生成される多項式剰余列 $\{f_1,f_2,\cdots\}$ に対し, $i\geq 3$ ならば $f_i\sim S_{\deg(f_i)}$ かつ $f_i\sim S_{\deg(f_{i-1})-1}$.

部分終結式は, 定義により $R[x]$ に属する. よって, 多項式剰余列の定義に現れる α_i,β_i をうまく取って, f_i が適当な S_j と等しくなるようにできれば, 現れる除算は全て R における整除となる. これは実際に可能である.

命題 5.21 多項式剰余列を,

$$d_i=\deg(f_i)-\deg(f_{i+1}),\quad \alpha_i=1,$$

$$\beta_2=1,\quad \beta_i=\mathrm{lc}(f_{i-1})\zeta_i^{d_{i-1}},$$

$$\zeta_2=1,\quad \zeta_i=\mathrm{lc}(f_{i-1})^{d_{i-2}}\zeta_{i-1}^{(1-d_{i-2})},$$

$$f_{i+1}=\mathrm{prem}(f_{i-1},f_i)/\beta_i$$

で定めれば,

$$f_i=\pm S_{\deg(f_{i-1})-1}(f_1,f_2).$$

ここで, 係数を不定元としたとき, 終結式が, これら係数の既約多項式であることが知られている. よって, GCD が 1 になる場合を考えれば, この命題で定められる多項式剰余列は, 一般の多項式に対して最大の係数除去を行なっていると考えられる. 実際に, 前の例に適用してみると,

$$27x^5 - 9x^2 + 27$$
$$27x^3 - 81x + 27$$
$$-36x^2 + 243x - 54$$
$$1971x - 438$$
$$-5329$$

となり, 原始的多項式としたものには及ばないものの, かなりの係数を除去していることがわかる.

5.5 Modular アルゴリズム

前節で述べた互除法による GCD の計算は, GCD の次数が比較的高い場合には良好に動作するが, GCD の次数が低い場合, 特に GCD が 1 の場合などでは, 部分終結式法によっても係数の増大は避けられない. このような理由から, GCD は, **modular** アルゴリズムにより計算されることが多い. modular アルゴリズムとは, 係数環の剰余環における演算結果からもとの環上の結果を得るタイプのアルゴリズムの総称である. modular アルゴリズムには, 中国剰余定理を用いるものと, Hensel 構成を用いるものがある. 後者については因数分解の節で詳しく述べることとし, ここでは, 前者について述べる.

命題 5.22 (中国剰余定理) 可換環 A のイデアル $A_1, \cdots, A_r$ の任意の対 $(A_i, A_j)(i \neq j)$ に対し,$A_i + A_j = A$ ならば,

$$A/\prod A_i \simeq \bigoplus A/A_i.$$

[証明] $r = 2$ のときを示す. $A_1 + A_2 = A$ より $a_1 + a_2 = 1$ なる $a_i \in A_i$ が存在する. 任意の $x, y \in A$ に対し, $z = xa_2 + ya_1$ と置くと, $z \equiv x \pmod{A_1}$ かつ $z \equiv y \pmod{A_2}$ より, 標準的写像

$$\phi : A \to A/A_1 \oplus A/A_2$$

は全射かつ $\mathrm{Ker}(\phi) = A_1 \cap A_2$ より $A/A_1 \cap A_2 \simeq A/A_1 \oplus A/A_2$. ここで, $A_1A_2 \subset A_1 \cap A_2 = (A_1 \cap A_2) \cdot (A_1 + A_2) \subset A_1A_2$ より, $A_1A_2 = A_1 \cap A_2$. よって

$$A/A_1A_2 \simeq A/A_1 \oplus A/A_2.$$

一般の場合は, $A_1 + A_2 \cdots A_r \supset \prod_{i \geq 2}(A_1 + A_i) \ni 1$ より帰納法が使える. □

系 5.23 A を Euclid 環とする. $m_i \in A$ が互いに素ならば, $A/(\prod m_i) \simeq \bigoplus A/(m_i)$.

例 5.24 K を体とする. $f_i \in K[x]$ が互いに素ならば, $K[x]/(\prod f_i) \simeq \bigoplus K[x]/(f_i)$.

これは, 有限体上の因数分解に用いられる. また, f_i として $x - a_i$ なる形のものをとれば, 補間法をあらわすと考えられる.

例 5.25 $m_i \in \mathbb{Z}$ が互いに素ならば, $\mathbb{Z}/(\prod m_i) \simeq \bigoplus \mathbb{Z}/(m_i)$.

次の補題は, modular 演算によるイメージをもとの空間に引き戻す際に常に用いられる.

補題 5.26 $a \equiv a' \bmod A$, $|a| \leq B$, $|a'| \leq A/2$, $A > 2B$ ならば, $a = a'$.

[証明] $a - a' = kA$ とする. $|a - a'| \leq |a| + |a'| \leq B + A/2 < A$ より $k = 0$. □

中国剰余定理の応用として, $\mathbb{Z}[x]$ における GCD 計算を例にとる.

$$f, g \in \mathbb{Z}[x], \quad h = \mathrm{GCD}(f, g) \in \mathbb{Z}[x]$$

とする. この時,

$$\mathrm{GCD}(f \bmod p_i, g \bmod p_i) = h \bmod p_i$$

なる素数 p_i が与えられれば, 係数に対して中国剰余定理を適用して,

$$m = \prod p_i | (h - h_1)$$

なる h_1 を構成できる. ここで, m が十分大きければ, h_1 の法 m での一意性により, h_1 の係数に m を加減して係数の絶対値を $m/2$ 以下としたものは, h と一致する. 問題は, あらかじめ, 法 p_i での GCD が, 真の GCD の像となっている, すなわち妥当であるかどうか不明であるという点であるが, 妥当でない p は有限個しかないことがわかる (互いに素な場合の終結式を考えればよい). p の妥当性は法 p での GCD の次数が最低という条件で特徴づけられるため, さまざまな p での GCD 計算を繰り返すことにより判定できる.

このほか, 整数に対する中国剰余定理は, 拡張 Euclid 互除法, 行列式, 終結式の計算に用いられる. この方法が効力を発揮するのは, 結果の大きさに比較して, 途中の計算において大きな式が現れる場合 (中間式膨張) である. 現在主要なシステムにおいては, GCD は, 後で述べる Hensel 構成により計算される場合が多いが, 前処理として,

いくつかの数あるいは式を法として GCD を計算してみることは, GCD が 1 である場合のチェックとして有効である.

最後に, 剰余環での像から, 逆像を求める方法を 2 つ紹介する. 環 R は, 整数環または, 体上の一変数多項式環とし, $m_1, \cdots, m_r \in R$ は互いに素であるとする. この時, 与えられた $u_1, \cdots, u_r \in R$ に対し

$$u \equiv u_i \pmod{m_i}$$

なる $u \in R$ を求めることが目標である. $m = \prod m_i$ とおく.

方法 **5.27** (Lagrange 補間) $\mathrm{GCD}(m_i, m/m_i) = 1$ より, ある $v_i, w_i \in R$ が存在して $v_i m/m_i + w_i m_i = 1$. この時 $L_i = v_i m/m_i$ とおけば $L_i \equiv \delta_{ij} \pmod{m_j}$. これにより, $u = \sum u_i L_i$ とおけば $u \equiv u_i \pmod{m_i}$.

これは, 法の数が固定の時, 一度, 基底 L_i を計算してしまえば, 任意の u_i に対して線形結合を作るだけで解が得られるが, 法の数が増えた時に基底を計算し直す必要がある.

方法 **5.28** (Newton 補間) $i < j$ のとき $\mathrm{GCD}(m_i, m_j) = 1$ より, ある $v_{ij}, N_{ij} \in R$ が存在して $v_{ij} m_j + N_{ij} m_i = 1$. この時 $N_{ij} m_i \equiv 1 \pmod{m_j}, i < j$. これにより $u = v_r m_{r-1} m_{r-2} + \cdots + v_2 m_1 + v_1$ なる形で u を求めると

$$v_1 = u_1$$
$$v_j = (\cdots((u_j - v_1)N_{1j} - v_2)N_{2j} - \cdots - v_{j-1})N_{j-1,j} \pmod{m_j})$$
$$\cdots$$

これは, $m_1, \cdots, m_r$ に対して構成された解を用いて $m_1, \cdots, m_{r+1}$ での解を計算している. その際, 新たに付け加えた m_{r+1} を法とする, $m_1, \cdots, m_r$ の逆元の計算を行なっている. v_j の計算が複雑に見えるが, 実際には, $j-1$ に対する解 u' を用いて,

$$(u_j - u')(m_1 \cdots m_{j-1})^{-1} \bmod m_j$$

を計算しているに過ぎない. 構成法からわかるように, これは, 法の数を増やして精度を上げていくタイプのアルゴリズムに向く.

Chapter 6

多項式の因数分解

6.1 有限体

計算機代数においては, 整数に関する演算を効率よく行うことを目的として, 適当な素数 p を選んで, p を法とする演算 (modular 演算) を行ったのち, 整数に関する結果を得るという手法がしばしば用いられる. また, 暗号に関する計算のように, 有限体における計算そのものが対象となっている場合もある. 本節では, 有限体に関する基本的事項に関し, なるべく self contained な形で述べる.

定義 6.1 有限個の元からなる体を有限体と呼ぶ. q 個の元からなる体を $GF(q)$ と書く.

命題 6.2 $GF(q)$ の標数はある素数 p で, $q = p^n$. ここで, $n = [GF(q) : GF(p)]$.

[証明] 標数 0 ならば $\mathbb{Q} \subset GF(q)$ となるから標数は正. $p > 0$ を標数とすると, $GF(p) \subset GF(q)$ より $GF(q)/GF(p)$ は有限次代数拡大. その拡大次数を n とおけば, $\{w_1, \cdots, w_n\} \subset GF(q)$ なる $GF(p)$ 上線形独立な基底が存在して,

$$GF(q) = GF(p)w_1 \oplus \cdots \oplus GF(p)w_n$$

よって, $|GF(q)| = (|GF(p)|)^n = p^n$. □

命題 6.3 $GF(q) \quad (q = p^n)$ は $x^{q-1}-1$, $x^q - x$ の最小分解体で, $x^q - x = \prod_{\alpha \in GF(q)} (x - \alpha)$.

[証明] 乗法群 $GF(q)^\times$ は有限群で, その位数は $q-1$ より, 全ての元 $\alpha \in GF(q)^\times$ に対し $\alpha^{q-1} = 1$ すなわち α は $x^{q-1} - 1$ の根. $x^{q-1} - 1$ の根はこれで尽くされるから,

$$x^{q-1} - 1 = \prod_{\alpha \in GF(q)^\times} (x - \alpha), \quad x^q - x = \prod_{\alpha \in GF(q)} (x - \alpha).$$

□

系 6.4 $GF(q)$ の標数が奇数とする.

$$F_+ = \{\alpha \in GF(q) \mid \alpha^{(q-1)/2} = 1\}, \quad F_- = \{\alpha \in GF(q) \mid \alpha^{(q-1)/2} = -1\}$$

とおけば, $GF(q) = F_+ \cup F_- \cup \{0\}$ で $|F_+| = |F_-| = (q-1)/2$.

[証明] $x^q - x = x(x^{(q-1)/2} - 1)(x^{(q-1)/2} + 1)$ なる分解が成り立つから, $GF(q)$ が上のように分解されることがわかる. F_+ は $x^{(q-1)/2} - 1$ の根全体で, 次数から $|F_+| = (q-1)/2$ がわかる. F_- も同様. □

命題 6.5 一つの体 K に含まれる $GF(q)$ $(q = p^n)$, $GF(q')$ $(q' = p^{n'})$ に対し

$$GF(q) \subset GF(q') \Leftrightarrow n|n'.$$

[証明] $GF(q) \subset GF(q')$ とすれば,$[GF(q) : GF(p)]|[GF(q') : GF(p)]$ より $n|n'$.

逆に $n|n'$ とする. $GF(q')$ は K に含まれる, $x^{q'} - x$ の最小分解体だから,

$$GF(q') = \{\alpha \in K \mid \alpha^{q'} - \alpha = 0\}$$

同様に,

$$GF(q) = \{\alpha \in K \mid \alpha^{q} - \alpha = 0\}$$

$n|n'$ より $(q-1)|(q'-1)$ が言えるから, $x^q - x | x^{q'} - x$. よって, $\alpha \in GF(q) \Rightarrow \alpha \in GF(q')$. □

系 6.6 $GF(p)$ の代数的閉包 Ω を一つ定めれば, $GF(q) \subset \Omega$ は, Ω における $x^q - x$ の最小分解体として一意に定まる.

命題 6.7 $GF(q)^\times$ は位数 $q-1$ の巡回群.

[証明] $G = GF(q)^\times$ は有限アーベル群より, $1 < n_i \in \mathbb{Z}$ $(i = 1, \cdots, l)$, $n_i | n_j$ $(i < j)$ が存在して,

$$G \simeq \bigoplus \mathbb{Z}/(n_i).$$

$l > 1$ とすれば, 少なくとも n_1^2 個の元が $x^{n_1} - 1 = 0$ の根となるが, これは不可能. よって

$$l = 1, \quad G \simeq \mathbb{Z}/(n_1).$$

□

命題 6.8 f を $GF(q)$ 上既約とすると,

$$f|x^{q^n} - x \Leftrightarrow \deg(f)|n$$

[証明] $GF(q)$ の代数的閉包 Ω を一つ固定する. f が $GF(q)$ 上既約とする. f の Ω における根を α とする.
$\underline{\Rightarrow)}$ $f|x^{q^n}-x$ とする. このとき $f(\alpha)=0$ より $\alpha^{q^n}-\alpha=0$. よって, $\alpha \in GF(q^n) \subset \Omega$. これから $GF(q)(\alpha) \subset GF(q^n)$. ゆえに

$$\deg(f) = [GF(q)(\alpha) : GF(q)]|[GF(q^n) : GF(q)] = n.$$

$\underline{\Leftarrow)}$ $\deg(f)|n$ とする. $GF(q)(\alpha), GF(q^n)$ は一つの体 Ω に含まれるから, $GF(q)(\alpha) \subset GF(q^n)$ これより $\alpha^{q^n} - \alpha = 0$. f は α の最小多項式だから

$$f|x^{q^n} - x.$$

□

$x^{q^n} - x$ は重根を持たないから, 次が成り立つ.

系 6.9 $F = \{f|f : GF(q)$ 上既約, モニックで $\deg(f)|n\}$ とおけば $x^{q^n} - x = \prod_{f\in F} f$.

6.2 無平方分解

以下で述べるさまざまな因数分解アルゴリズムは, 入力が重複因子をもつことを許さない. よって, 以下に述べる無平方分解を行なっておく必要がある.

定義 6.10 重複因子を持たない多項式を無平方 (**squarefree**) という. $f \in K[x]$ に対し,

$$f = \prod g_1^{n_i}$$

で各 g_i は無平方かつ互いに素な多項式で $n_1 < n_2 < \cdots$ となっている時, この因数分解を f の 無平方分解 と呼ぶ.

まず, 標数 0 の体 K の無平方分解アルゴリズムについて述べる.

補題 6.11 K を標数 0 の体とする. $f \in K[x]$ に対し, $f = \prod_i g_i^{n_i}$ を f の無平方分解とするとき,

$$\mathrm{GCD}(f, f') = \prod_i g_i^{n_i-1}.$$

[証明]

$$f' = \prod_i g_i^{n_i-1}(\sum_i n_i g_i' \prod_{k\neq i} g_k)$$

より

$$\mathrm{GCD}(f,f') = \prod_i g_i^{n_i-1}\mathrm{GCD}(\prod_k g_k, \sum_i n_i g_i' \prod_{k\neq i} g_k)$$

K の標数が 0 より $n_i \neq 0, g_i' \neq 0$ だから,

$$\mathrm{GCD}(\prod_k g_k, \sum_i n_i g_i' \prod_{k\neq i} g_k) = \prod_k \mathrm{GCD}(g_k, \sum_i n_i g_i' \prod_{k\neq i} g_k) = \prod_k \mathrm{GCD}(g_k, g_k')$$

ここで g_k が無平方より, g_k を既約分解して考えれば $\mathrm{GCD}(g_k, g_k') = 1$. よって,

$$\mathrm{GCD}(f,f') = \prod_i g_i^{n_i-1}.$$

□

アルゴリズム 6.12 (無平方分解)

Input : $f(x) \in K[x]$ (K は標数 0 の体)
Output : f の無平方分解 $f = g_1^{n_1} g_2^{n_2} \cdots$

$F \leftarrow 1$
$flat \leftarrow f/\mathrm{GCD}(f,f'), \quad m \leftarrow 0, \quad counter \leftarrow 1$
while($flat \neq constant$) {
(a) while ($flat | f$) {
$f \leftarrow f/flat, \quad m \leftarrow m+1$
}
(b) $flat_1 \leftarrow \mathrm{GCD}(flat, f)$
$g \leftarrow flat/flat_1, \quad flat \leftarrow flat_1$
(c) $F \leftarrow F \cdot g^m, \quad counter \leftarrow counter + 1$
}
return F

命題 6.13 アルゴリズム 6.12 は f の無平方分解を出力する.

[証明] (a) において $counter = k$ の時,

$$f = g_k^{n_k - n_{k-1}} g_{k+1}^{n_{k+1} - n_{k-1}} \cdots, \quad flat = g_k g_{k+1} \cdots, \quad m = n_{k-1} \quad (n_0 = 0)$$

あることを帰納法により示す. $k = 1$ の時は補題より正しい. k まで正しいとする. 仮定より, $flat^{n_k - n_{k-1}} | f$ かつ $flat^{n_k - n_{k-1}+1} \not| f$ より (b) において

$$m = n_{k-1} + (n_k - n_{k-1}) = n_k, \quad f = g_{k+1}^{n_{k+1} - n_k} \cdots$$

となる. よって, (c) においては

$$flat = flat_1 = g_{k+1} \cdots$$

となり $k+1$ でも正しい. また (c) において $g = g_k$, $m = n_k$ となっているからこのアルゴリズムは正しく無平方分解を出力する. □
標数 $p > 0$ の体 K 上では微分して 0 になる多項式が存在するため注意を要する.

補題 6.14 K を標数 $p > 0$ の有限体とする. K 上の一変数多項式 f に対し, $f' = 0 \Leftrightarrow$ ある多項式 g が存在して $f = g^p$.

補題 6.15 K を標数 $p > 0$ の有限体とする. h を, f の因子のうち, 重複度が p で割り切れるものの積とし, $g = f/h = \prod g_i^{n_i}$ とする. このとき,

$$\mathrm{GCD}(f, f') = h \,\mathrm{GCD}(g, g') = h \prod g_i^{n_i - 1}.$$

[証明] $h' = 0$ より $f' = g'h + gh' = g'h$. よって, $\mathrm{GCD}(f, f') = h \,\mathrm{GCD}(g, g')$.

$$\mathrm{GCD}(g, g') = \prod_i g_i^{n_i - 1} \mathrm{GCD}(\prod_k g_k, \sum_i n_i g_i' \prod_{k \neq i} g_k)$$

仮定により $n_i \neq 0$. また g_i は無平方だから, 前補題により $g_i' \neq 0$ だから,

$$\mathrm{GCD}(\prod_k g_k, \sum_i n_i g_i' \prod_{k \neq i} g_k) = \prod_k \mathrm{GCD}(g_k, \sum_i n_i g_i' \prod_{k \neq i} g_k) = \prod_k \mathrm{GCD}(g_k, g_k')$$

ここで g_k が無平方より, g_k を既約分解して考えれば $\mathrm{GCD}(g_k, g_k') = 1$. よって,

$$\mathrm{GCD}(g, g') = \prod_i g_i^{n_i - 1}.$$

□

アルゴリズム 6.16 (無平方分解)

Input : $f(x) \in K[x]$ (K は標数 $p > 0$ の有限体)
Output : f の無平方分解 $f = g_1^{n_1} g_2^{n_2} \cdots$
$F \leftarrow 1$
if $f' \neq 0$ {
  $flat \leftarrow f/\mathrm{GCD}(f, f'), \quad m \leftarrow 0, \quad counter \leftarrow 1$
  while($flat \neq constant$) {
(a) while ($flat | f$) {
    $f \leftarrow f/flat, \quad m \leftarrow m + 1$
  }
(b) $flat_1 \leftarrow \mathrm{GCD}(flat, f)$
  $g \leftarrow flat/flat_1, \quad flat \leftarrow flat_1$
(c) $F \leftarrow F \cdot g^m, \quad counter \leftarrow counter + 1$
  }
}
if $f \neq constant$ {
  $g \leftarrow f^{1/p}$
  $g = g_1^{n_1} g_2^{n_2} \cdots$ と無平方分解
  $F = F \cdot g_1^{pn_1} g_2^{pn_2} \cdots$
}
return F

この場合, $flat$ が, f の重複度が p で割り切れない因子の積となるから, $flat$ が定数となった時点で f の各因子の重複度は p で割り切れる. よって, $1/p$ 乗が計算でき, その無平方分解を p 乗すれば, f の無平方分解が計算できたことになる.

ここでは, 係数を体としたが, UFD 上では, あらかじめ f に対し $cont(f)$ を計算し, それで f を割って原始的多項式としてから行なう. 整数上の場合には, この操作は単に係数の GCD を計算することを意味するが, 多変数多項式の場合, 多項式の GCD が必要となる. さらに, 係数環上での無平方分解も要求されているならば, このアルゴリズムを再帰的に用いる必要がある.

このアルゴリズムの Yun による改良も知られている. また, この素朴な方法は, 重複度の大きな因子をもつ場合には, 最初の GCD 計算で多大な時間を必要とする場合が多く, 実用的に問題がある. これを避けるため, 重複度最大の因子から順に直接求めていく方法が Wang-Trager [46] により考案されている. この方法は, 後述する Hensel 構成との組合せにより大きな効果を発揮する. これについては 6.8.3 節で述べる.

6.3 Berlekamp アルゴリズム

p を素数とし, q 元体 $K = GF(q)$ $(q = p^n)$ を係数とする一変数多項式環 $R = K[x]$ における既約因子分解を考える. $f(x) \in R$ が無平方であるとする. $f = \prod_{i=1}^{r} f_i$ と既約分解すると, 中国剰余定理により,

$$R/(f) \simeq \bigoplus R/(f_i).$$

$R/(f), R/(f_i)$ 上に次の K-準同型 (Frobenius 写像) を考える.

$$\pi : f \mapsto f^q \bmod f$$

$$\pi_i : f \mapsto f^q \bmod f_i$$

すると,

$$\mathrm{Ker}(\pi - I) \simeq \bigoplus \mathrm{Ker}(\pi_i - I)$$

である. ここで, f_i は既約より R/f_i は体である. また $x^q - x$ は, K のすべての元を根にもつが, 体 R/f_i 上で考えれば, 根はすべてこれで尽くされている. よって $\mathrm{Ker}(\pi_i - I) = K$. 結局

$$\mathrm{Ker}(\pi - I) \simeq \bigoplus K$$

すなわち, f の既約多項式の個数を r とすると, $\mathrm{Ker}(\pi - I)$ は, r 個の K の直和となる. さらに, これは次のように言い替えられる.

命題 6.17 1. $f|(g^q - g) \Rightarrow$ ある $(s_1, \cdots, s_r) \in K^r$ が存在して $f_i|(g - s_i)$

2. すべての $(s_1, \cdots, s_r) \in K^r$ に対し, ある g が存在して $f|(g^q - g), f_i|(g - s_i)$

g は, $\deg(g) < \deg(f)$ としてよいから, $g \in \mathrm{Ker}(\pi - I)$ をとると, 適当な s に対し $\mathrm{GCD}(g - s, f)$ は自明でない f の因子を与える.

次のアルゴリズムは有限体上の一変数多項式の因数分解を行う.

アルゴリズム 6.18 (Berlekamp[5])

Input : $f(x) \in R$; f は無平方
Output : $\{f_1, f_2, \cdots\}$ $f = f_1 f_2 \cdots$ は f の既約分解
$F = \{f\}$
$Q \leftarrow \pi$ の行列表現
$\{e_1 = 1, e_2, \cdots, e_r\} \leftarrow \mathrm{Ker}(Q - I)$ の K-基底

if $(r = 1)$ then return F
$k \leftarrow 2$
do {
  $F_1 \leftarrow \emptyset$
  while $F \neq \emptyset$ do {
    $g \leftarrow F$ の一つの元
    $F \leftarrow F \backslash \{g\}$
    $F_g \leftarrow \{\mathrm{GCD}(g, e_k - s)(s \in GF(q))$ のうち, 定数でないもの $\}$
    $g \leftarrow g / \prod_{h \in F_g} h$
    $F_1 \leftarrow F_1 \cup F_g$
    if ($g \neq 1$) $F_1 \leftarrow F_1 \cup \{g\}$
    if $(|(F \cup F_1)| = r)$ return $F \cup F_1$
  }
  $k \leftarrow k + 1$
  $F \leftarrow F_1$
}
return F

基底により全ての既約因子が分離できることは次の命題よりわかる.

命題 **6.19** $\{e_1, \cdots, e_r\}$ を $\mathrm{Ker}(\pi - I)$ の K-基底とするとすべての $i \neq j$ なる i, j に対し, ある k, s が存在して $f_i | (e_k - s), f_j \not| (e_k - s)$

[証明] この主張が偽ならば

ある i, j が存在して, すべての e_k, s に対し $((f_i | (e_k - s) \Rightarrow f_j | (e_k - s)))$

となる. この時, 各 k に対し, $f_i | (e_k - s_k)$ なる s_k をとれば, $\{e_k\}$ が基底であることより

すべての $g \in \mathrm{Ker}(\pi - I)$ に対し, ある s が存在して $f_i | (g - s), f_j | (g - s)$.

ところが, $\mathrm{Ker}(\pi - I)$ の中には f_i, f_j を分離するものがあるから矛盾. □

以上述べたアルゴリズムは, q が小さい時に用いられる最も原始的な形である. このアルゴリズムでは, 最悪 q 回 GCD を繰り返す必要が生ずる. しかし, 実際には $\mathrm{GCD}(g, e - s)$ が 1 または g 以外の値を取るような s の値は限られていて, その値は s のある多項式の根となっている.

命題 6.20 $e \in \mathrm{Ker}(\pi - I)$ に対し,

$$S = \{s \in GF(q) \mid \mathrm{GCD}(e - s, f) \neq 1\}$$

とおく. このとき $m(x) = \prod_{s \in S}(x - s)$ とおけば, $m(x)$ は e の $R/(f)$ における最小多項式. すなわち, $f|m(e)$ となるような, 最小次数の多項式.

[証明] f の各因子 f_i に対し $e \equiv s_i \bmod f_i$ となるような s_i が存在する. このとき $s_i \in S$ で

$$f_i|(e - s_i)|m(e).$$

f は 無平方より $f|m(e)$. 逆に, $M(x)$ を, e の $R/(f)$ における最小多項式とする. もし $\deg(M) < \deg(m)$ ならば, ある $s \in S$ が存在して

$$m = (x - s)q + r, \quad r \in GF(q)\backslash\{0\}$$

と書ける. この時, f の因子 f_i が存在して, $f_i|(e - s)$ となるから, $f_i|r$. これは矛盾. □

e の最小多項式は, e^k が $\{1, e, e^2, \cdots, e^{k-1}\}$ で張られているか否かを $k = 1$ から順に調べることにより決定できる.

最小多項式 $m(x)$ を用いれば, m の根を求めさえすれば, それらに対しのみ GCD を実行すればよいことになる.

6.4 Cantor-Zassenhaus アルゴリズム

前節で述べた Berlekamp アルゴリズムおよび最小多項式を用いる改良によって q が小さい場合には十分効率よく因数分解できる. しかし, q が大きい場合には以下で述べるような確率的アルゴリズムを用いなければ, 効率よい因数分解は難しい. 以下, 前節と同様の記号を用いる.

命題 6.21 標数 p が奇数とする. $e \in \mathrm{Ker}(\pi - I)$ とすれば,

$$\mathrm{GCD}(e^{(q-1)/2} - 1, f) \neq 1, f$$

となる確率は, k を f の既約因子の個数とするとき

$$1 - (\frac{q-1}{2q})^k - (\frac{q+1}{2q})^k.$$

[証明] f_i $(i = 1, \cdots, k)$ を f の既約因子とし, $e \equiv s_i \bmod f_i$ $(s_i \in GF(q))$ とする.

$$e^{(q-1)/2} \equiv s_i^{(q-1)/2} \equiv 0, \pm 1 \bmod f_i$$

より,

$$\mathrm{GCD}(e^{(q-1)/2} - 1, f) = f \Leftrightarrow \text{すべての } i \text{ に対し } s_i^{(q-1)/2} \equiv 1 \bmod f_i$$

$$\mathrm{GCD}\,(e^{(q-1)/2} - 1, f) = 1 \Leftrightarrow \text{すべての } i \text{ に対し } s_i^{(q-1)/2} \equiv 0, -1 \bmod f_i$$

$s \in GF(q)$ とする時, $s^{(q-1)/2} \equiv 1$ なる元は $(q-1)/2$ 個, $s^{(q-1)/2} \equiv 0, -1$ なる元は $(q+1)/2$ 個. よって, $\mathrm{GCD}(e^{(q-1)/2} - 1, f) = f$, $\mathrm{GCD}\,(e^{(q-1)/2} - 1, f) = 1$ なる確率はそれぞれ

$$(\frac{q-1}{2q})^k, \quad (\frac{q+1}{2q})^k$$

となる. □

標数 2 の場合を扱うために, $GF(2^n)$ 上の多項式 f の trace $\mathrm{Tr}(f)$ を定義する.

定義 6.22 $GF(2^n)$ において, $\mathrm{Tr}(x) \in GF(2^n)[x]$ を

$$\mathrm{Tr}(x) = \sum_{i=0}^{n-1} x^{2^i}$$

と定義する.

補題 6.23 $x^{2^n} - x = \mathrm{Tr}(x)(\mathrm{Tr}(x) + 1)$.

系 6.24 1. $a \in GF(2^n) \Rightarrow \mathrm{Tr}(a) \in GF(2)$

2. $|\{s \in GF(2^n) \mid \mathrm{Tr}(s) = 0\}| = |\{s \in GF(2^n) \mid \mathrm{Tr}(s) = 1\}| = 2^{n-1}$

命題 6.25 標数 $p = 2$ とする. $e \in \mathrm{Ker}(\pi - I)$ とすれば,

$$\mathrm{GCD}\,(\mathrm{Tr}(e), f) \neq 1, f$$

となる確率は, k を f の既約因子の個数とするとき $1 - 1/2^{k-1}$.

[証明] f_i $(i = 1, \cdots, k)$ を f の既約因子とし, $e \equiv s_i \bmod f_i$ $(s_i \in GF(q))$ とする.

$$\mathrm{Tr}(e) \equiv \mathrm{Tr}(s_i) \equiv 0, 1 \bmod f_i$$

より,

$$\mathrm{GCD}(\mathrm{Tr}(e), f) = f \Leftrightarrow \text{すべての } i \text{ に対し } \mathrm{Tr}(s_i) \equiv 0 \bmod f_i$$

$$\mathrm{GCD}(\mathrm{Tr}(e), f) = 1 \Leftrightarrow \text{すべての } i \text{ に対し } \mathrm{Tr}(s_i) \equiv 1 \bmod f_i$$

$s \in GF(q)$ とする時, $\mathrm{Tr}(s) \equiv 0, 1$ なる元はそれぞれ $q/2$ 個. よって, $\mathrm{GCD}(\mathrm{Tr}(e), f) = f$, $\mathrm{GCD}(\mathrm{Tr}(e), f) = 1$ なる確率はそれぞれ $1/2^k$ となる. □
これらをもとに, 次のようなアルゴリズムを得る.

アルゴリズム 6.26 (Cantor-Zassenhaus[8])

Input : $f(x) \in GF(q)[x]$, $q = p^n$, f は無平方
Output: $\{f_1, f_2, \cdots\}$ $f = f_1 f_2 \cdots$ は f の既約分解
$F = \{f\}$
$Q \leftarrow \pi$ の行列表現
$\{e_1 = 1, e_2, \cdots, e_r\} \leftarrow \mathrm{Ker}(Q - I)$ の K-基底
if $(r = 1)$ then return F
while$(|F| < r)$ do {
    $(c_1, \cdots, c_r) \leftarrow$ 乱数ベクトル $(c_i \in GF(q))$
    $e \leftarrow \sum c_i e_i$
    if $p = 2$
        $E \leftarrow \mathrm{Tr}(e) \bmod f$
    else
        $E \leftarrow e^{(q-1)/2} - 1 \bmod f$
    $F_1 \leftarrow \emptyset$
    while $(F \neq \emptyset)$ do {
        $g \leftarrow F$ の元, $\quad F \leftarrow F \backslash \{g\}$
        $h \leftarrow \mathrm{GCD}(g, E)$
        if $h \neq 1, g$
            $F_1 \leftarrow F_1 \cup \{h, g/h\}$
        else
            $F_1 \leftarrow F_1 \cup \{g\}$
    }
    $F \leftarrow F_1$
}
return F

6.5 DDF (distinct degree factorization)

有限体上の多項式に対しては, GCD のみによる DDF (distinct degree factorization; 次数別因子分解) とよばれる予備的な分解が可能である. DDF で得られる各因子は, それぞれ同一次数の既約因子の積からなる. これにより, 各次数の既約因子が 1 つの場合には GCD のみにより既約因子が得られる. また, 分解成分の既約因子が同一次数であることを利用して, 特殊な手法により既約分解をすることも可能になる.

系 6.9 より, 次のアルゴリズムが得られる.

アルゴリズム 6.27

Input : $f(x) \in GF(q)[x]$, f は無平方

Output : $f(x) = \prod f_i$, f_i は f の i 次既約因子の積

$w \leftarrow x, \quad i \leftarrow 1$

while($i \leq \deg(f)/2$) do {

$\quad w \leftarrow w^q \bmod f$

$\quad f_i \leftarrow \mathrm{GCD}(f, w - x)$

$\quad$ if $f_i \neq 1$ {

$\quad\quad f \leftarrow f/f_i$

$\quad\quad w \leftarrow w \bmod f$

$\quad\quad E \leftarrow e^{(q-1)/2} - 1$

$\quad$ }

}

while ($i < \deg(f)$)

$\quad f_i \leftarrow 1$

$f_{\deg(f)} \leftarrow f$

return $\prod f_i$

i 回目においては, i の真の約数を次数に持つ因子は既に除かれているため, i 次の因子のみが取り出せる. また, $i \geq \deg(f)/2$ となった時点で, f が既約な $\deg(f)$ 次因子であることは明らかである.

このアルゴリズムにおいて, $w^q \bmod f$ の計算を繰り返す必要がある. しかし, 一般に

$$g = \sum_{i=0}^{m} a_i x^i \Rightarrow g^q \bmod f = \sum_{i=0}^{m} a_i x^{iq} \bmod f$$

より, $w_0 = x^q \bmod f$ の計算結果から $w_i = w_0^i \bmod f = w_{i-1} w_0 \bmod f$ の計算を $i = 1, \cdots, \deg(f) - 1$ に対して行っておけば, $w^q \bmod f$ の計算は効率よく計算できる.

6.6 次数別因子の分解

f は無平方で, d 次の既約因子の積であるとする. f はもちろん Berlekamp アルゴリズムにより既約分解できるが, 含まれる既約因子の次数が全て等しいことを利用して分解を行うことを考える.

命題 6.28 $q = p^n$ で p が奇素数とする. $f = f_1 f_2$ で f_1, f_2 が f の 2 つの d 次既約因子とする. $GF(q)$ 上の 高々 $2d-1$ 次式 g をランダムに選ぶとき, $\mathrm{GCD}(g^{(q^d-1)/2}-1, f) = f_1$ または f_2 となる確率は $1/2 - 1/(2q^d)$.

[証明] f_1, f_2 が d 次既約より,

$$GF(q)[x]/(f_1) \simeq GF(q)[x]/(f_2) \simeq GF(q^d).$$

よって, 任意の $g \in GF(q)[x]$ に対し,

$$s_1 = (g \bmod f_1)^{(q^d-1)/2} = 0, \pm 1, \quad s_2 = (g \bmod f_2)^{(q^d-1)/2} = 0, \pm 1.$$

$GF(q)$ の元のうち $(q^d-1)/2$ 乗して 1 になるものは $(q^d-1)/2$ 個, そうでないものは $(q^d+1)/2$ 個ある. $\mathrm{GCD}(g^{(q^d-1)/2}-1, f) = f_1$ または f_2 となるのは, s_1, s_2 の一方のみが 1 になる場合である. ここで, 中国剰余定理により,

$$GF(q)[x]/(f_1 f_2) \simeq GF(q)[x]/(f_1) \bigoplus GF(q)[x]/(f_2) \simeq GF(q^d) \bigoplus GF(q^d)$$

だから, 任意の $(a_1, a_2) \in GF(q^d) \bigoplus GF(q^d)$ の元に対し, 多項式の組に対し, $a_1 = g \equiv g \bmod f_1$, $a_2 = g \equiv g \bmod f_2$ なる $2d-1$ 次以下の多項式 g が一意的に対応する. よって, $2d-1$ 以下の多項式 q^{2d} 個のうち, $\mathrm{GCD}(g^{(q^d-1)/2}-1, f) = f_1$ または f_2 となるのは,

$$2(q^d-1)/2 \cdot (q^d+1)/2 = (q^{2d}-1)/2$$

個であり, 確率は $1/2 - 1/(2q^{2d})$. □

命題 6.29 $q = 2^n$ とする. $f = f_1 f_2$ で f_1, f_2 が f の 2 つの d 次既約因子とし,

$$\mathrm{Tr}(x) = \sum_{i=0}^{nd-1} x^{2^i}$$

とする. $GF(q)$ 上の 高々 $2d-1$ 次式 g をランダムに選ぶとき, $\mathrm{GCD}(\mathrm{Tr}(x), f) = f_1$ または f_2 となる確率は $1/2$.

[証明] $f_1,\ f_2$ が d 次既約より,

$$GF(q)[x]/(f_1) \simeq GF(q)[x]/(f_2) \simeq GF(2^{nd}).$$

よって, 任意の $g \in GF(q)[x]$ に対し,

$$s_1 = \mathrm{Tr}(g \bmod f_1) = 0, 1, \quad s_2 = \mathrm{Tr}(g \bmod f_2) = 0, 1.$$

$GF(2^{nd})$ の元のうち Tr の値が $0,\ 1$ になるものは それぞれ 2^{nd-1} 個ずつある.
$\mathrm{GCD}(\mathrm{Tr}(x), f) = f_1$ または f_2 となるのは, $s_1,\ s_2$ の一方のみが 0 になる場合である.
ここで, 中国剰余定理により,

$$GF(q)[x]/(f_1 f_2) \simeq GF(q)[x]/(f_1) \bigoplus GF(q)[x]/(f_2) \simeq GF(2^{nd}) \bigoplus GF(2^{nd})$$

だから, 任意の $(a_1, a_2) \in GF(2^{nd}) \bigoplus GF(2^{nd})$ の元に対し, 多項式の組に対し, $a_1 = g \equiv g \bmod f_1$, $a_2 = g \equiv g \bmod f_2$ なる $2d-1$ 次以下の多項式 g が一意的に対応する.
よって, $2d-1$ 以下の多項式 2^{2nd} 個のうち, $\mathrm{GCD}(\mathrm{Tr}(x), f) = f_1$ または f_2 となるのは, $2 \cdot (2^{nd-1})^2 = 2^{2nd-1}$ 個であり, 確率は $1/2$. □
これらの命題は, ランダムに選んだ g により, f の 2 つの因子が確率 $1/2$ 以上で分離できることを意味している. これにより, 以下のような, Cantor-Zassenhaus アルゴリズムの変形版が適用できる.

アルゴリズム 6.30
Input : $f(x) \in GF(q)[x]$, $q = p^n$, f は無平方で d 次既約因子の積
Output : $f(x) = \prod f_i$, f の 既約因子分解
$r \leftarrow \deg(f)/d, \quad F \leftarrow \{f\}$
while($|F| < r$) do {
  $g \leftarrow 2d-1$ 次のランダムな多項式
  if $p = 2$
    $G \leftarrow \sum_{j=0}^{rd-1} g^{2^i} \bmod f$
  else
    $G \leftarrow g^{(q^d-1)/2} - 1 \bmod f$
  $F_1 \leftarrow \emptyset$
  while ($F \neq \emptyset$) do {
    $h \leftarrow F$ の $\deg(h) > d$ なる元, $\quad F \leftarrow F \backslash \{h\}$
    $z \leftarrow \mathrm{GCD}(h, G)$
    if $z \neq 1, h$

$F_1 \leftarrow F_1 \cup \{z, h/z\}$

else

$F_1 \leftarrow F_1 \cup \{h\}$

}

$F \leftarrow F_1$

}

return F

6.7 整数係数一変数多項式の因数分解

計算機代数システムにおいて用いられている因数分解アルゴリズムは, 何らかの準同型により多項式をより扱いやすい空間に写像し, その空間で, 多項式の像を因数分解し, なんらかの方法によりその分解された像からもとの空間における因子を構成する, という方法によるものが多い. 特に, 二変数以上の多項式の因数分解は一変数多項式に帰着されるなど, 一変数多項式の因数分解は, 因数分解アルゴリズムにおいて重要な位置を占める. ここでは, もっとも普通に用いられている Zassenhaus によるアルゴリズムについて述べる.

無平方な $f \in \mathbb{Z}[x]$ の因数分解アルゴリズムは, 有限体上の因数分解, Hensel 構成, 試し割りの三つの部分からなる.

命題 6.31 (Hensel) $f \in \mathbb{Z}[x], p \in \mathbb{N}$ は素数で,

$$f \equiv \prod f_{1i} \bmod p, \quad f_{1i} \in GF(p)[x], \quad \deg(f) = \deg(\prod_i f_{1i})$$

f_{1i} は $GF(p)$ 上で互いに素とする. この時, 任意の k に対し, $f_{ki} \in \mathbb{Z}/(p^k)[x]$ が存在して,

$$f \equiv \prod_i f_{ki} \bmod p^k, \quad f_{1i} \equiv f_{ki} \bmod p, \quad \deg(f_{1i}) = \deg(f_{ki}).$$

[証明] k に関する帰納法で示す. k が 1 の時は正しい. k まで正しいとする.

$$f_{(k+1)i} = f_{ki} + p^k h_i$$

なる形を仮定すると,

$$\prod_i f_{(k+1)i} \equiv \prod_i f_{ki} + p^k \sum_i h_i \prod_{j \neq i} f_{kj} \bmod p^{k+1}$$

一方, $f \equiv \prod_i f_{ki} \bmod p^k$ より,

$$f = \prod_i f_{ki} + p^k h \bmod p^{k+1}$$

と書ける. よって,

$$h \equiv \sum_i h_i \prod_{j \neq i} f_{kj} \equiv \sum_i h_i \prod_{j \neq i} f_{1j} \bmod p$$

となるように, $h_i \in GF(p)[x], \deg(h_i) \leq \deg(f_{1i})$ を決めることができることを示せばよい. これは, 命題 5.11 により言える. □

命題 6.32 $f \in \mathbb{Z}[x], p \in \mathbb{N}$ は素数,

$$f \equiv g_0 h_0 \bmod p, \quad g_0, h_0 \in GF(p)[x], \quad \deg(f) = \deg(g_0 h_0), \quad \mathrm{GCD}(g_0, h_0) = 1$$

とする. この時, $f \equiv gh \equiv g'h' \bmod p^k$ かつ $g \equiv g' \equiv g_0 \bmod p$, $\deg(g) = \deg(g') = \deg(g_0), \quad \deg(h) = \deg(h') = \deg(h_0), \quad \mathrm{lc}(g) \equiv \mathrm{lc}(g') \bmod p^k$ ならば,

$$g \equiv g' \bmod p^k, \quad h \equiv h' \bmod p^k.$$

[証明] k に関する帰納法で示す. k まで正しいとする. $k+1$ のとき, 帰納法の仮定により,

$$g' - g = p^k r, \quad h' - h = p^k s$$

と書ける. 一方, $gh \equiv g'h' \bmod p^{k+1}$ から $rh_0 + sg_0 \equiv 0 \bmod p$ が成り立つ. もし $s \equiv 0 \bmod p$ ならば $rh_0 \equiv 0 \bmod p$ より $k+1$ で正しい. もし $s \not\equiv 0$ ならば $g_0 | r$. ここで, $\mathrm{lc}(g) \equiv \mathrm{lc}(g') \bmod p^{k+1}$ より, $\deg(r \bmod p) < \deg(g_0)$. よって $r \equiv 0 \bmod p$ となり, この場合も $k+1$ で正しい. □

定義 6.33 $f = \sum_{i=0}^n a_i x^i \in C[x]$ に対し, f のノルム $\| f \|_1, \| f \|_2$ をそれぞれ,

$$\| f \|_1 = \sum_{i=0}^n |a_i|, \quad \| f \|_2 = \sqrt{\sum_{i=0}^n |a_i|^2}$$

($|a|$ は a の絶対値) で定義する.

命題 6.34 (Landau-Mignotte[27]) $f = \sum_{i=0}^n a_i x^i \in \mathbb{Z}[x], g = \sum_{i=0}^m b_i x^i \in \mathbb{Z}[x]$ とする. この時 $g | f$ ならば, $\| g \|_1 \leq |b_m / a_n| 2^m \| f \|_2$

系 6.35 $f, g \in \mathbb{Z}[x], g | f$ ならば, $\| \mathrm{lc}(f)/\mathrm{lc}(g) \cdot g \|_1 \leq 2^{\deg(g)} \| f \|_2$

命題 6.36 $f \in \mathbb{Z}[x]$ が無平方ならば, 有限個の素数 p を除いて $\deg(f \bmod p) = \deg(f)$ かつ $f \bmod p$ は無平方.

証明は, f の無平方性が, f と f' の終結式が 0 でないことと同値であることから明らか.
以上により $\mathbb{Z}[x]$ における因数分解は次のように行なわれる.

アルゴリズム 6.37 (Zassenhaus[49])

Input : $f(x) \in \mathbb{Z}[x]$; f は無平方
Output : $\{f_1, f_2, \cdots\}$ $f = f_1 f_2 \cdots$ は f の既約分解
$p \leftarrow \deg(f) = \deg(f \bmod p)$ かつ $f \bmod p$ が無平方となるような p
$a \leftarrow f$ の主係数
$F_1 \leftarrow f/a \bmod p$ の $GF(p)$ 上のモニックな既約因子全体 (有限体上の因数分解)
if $|F_1| = 1$ then return $\{f\}$
$F_1 = \{f_{11}, \cdots, f_{1m}\}$ とする.
$k \leftarrow p^k > \| f \|_2 \cdot 2^{\deg(f)+1}$ なる整数
$f \equiv \prod_i f_{1i} \bmod p$ から $f \equiv \prod_i f_{ki} \bmod p^k$ なる $F_k = \{f_{k1}, \cdots, f_{km}\}$ を
Hensel 構成で求める
$l \leftarrow 1$
$F \leftarrow \emptyset$
while ($2l \leq m$) {
(a) $S \leftarrow S \subset F_k$, $|S| = l$ なる新しい部分集合
if S が存在しない then $l \leftarrow l + 1$
else {
$g \leftarrow a \cdots \prod_{s \in S} s$
(b) $g \leftarrow g$ の各係数に p^k の整数倍を加えて絶対値が $p^k/2$ 以下としたもの
if $g | af$ {
$g \leftarrow \mathrm{pp}(g)$ (primitive part), $f \leftarrow f/g$, $F_k \leftarrow F_k \backslash S$
}
}
if $(f \neq 1)$ then $F \leftarrow F \cup \{f\}$
return F

命題 6.38 アルゴリズム 6.37 は f の既約因子分解を出力する.

[証明] (a) において, S が真の因子 G の法 p での因子から Hensel 構成により構成された法 p^k での因子とする. すると, 法 p^k での一意性により, (b) において

$$a/\mathrm{lc}(G)\cdot G \equiv g \bmod p^k.$$

ここで, 系 6.35 および k のとり方により,

$$\| a/\mathrm{lc}(G)\cdot G \|_1 \leq 2^{\deg(G)} \| f \|_2 \leq 2^{\deg(f)} \| f \|_2 < p^k/2.$$

また, $\| g \|_1 \leq p^k/2$ より

$$\| g - a/\mathrm{lc}(G)\cdot G \|_1 \leq \| g \|_1 + \| a/\mathrm{lc}(G)\cdot G \|_1 < 2\cdot p^k/2 = p^k.$$

以上により, $a/\mathrm{lc}(G)\cdot G = g$. アルゴリズム 6.37 では, 法 p での因子の個数が小さい順に試しているため, 割り切れた時点で既約性が成り立つ. □

このアルゴリズムでは,

$$g \text{ が } f \text{ の因子ならば, } \mathrm{lc}(f)/\mathrm{lc}(g)\cdot g \text{ は } \mathrm{lc}(f)f \text{ の因子}$$

という簡単な事実が使われている. そして, 予め候補の主係数を $\mathrm{lc}(f)$ にしておけば, もし真の因子ならば, 上記の理由により必ずそれは $\mathrm{lc}(f)f$ を割り切るのである.

さて, 実際にこのアルゴリズムを計算機上で実現する場合, 効率向上のために注意する点は数多くある. それらのいくつかを述べる.

1. p の選択

   $f \bmod p$ が無平方でさえあれば, アルゴリズムは進行するが, 有限体上での因数分解の出力する $\bmod p$ での因子の個数は p により一般に異なる. 特に, ある p に対し $f \bmod p$ が既約ならば f 自身既約と判定できるが, 他の $f \bmod p$ が既約でない p を選ぶことによって無駄な Hensel 構成をする場合もある. このため, 通常はいくつかの p を選んで有限体上での因数分解を複数回起動し, 最も因子の個数が少ない p を選ぶ.

2. Hensel 構成

   ここで述べた Hensel 構成は linear lifting と呼ばれるもので, p の冪が 1 次のオーダでしか増えない. 特に p が小さい場合, 目的の評価値に到達するのに多くの段数を必要とする. このため, p の冪を 2 次のオーダで上げていく quadratic lifting というアルゴリズムが考案されている. ただしこれは, $\sum_i a_i \prod_{k\neq i} f_k = 1$ における a_i も同時に Hensel 構成しなければならないため, p^k がマシン整数程度のうちは quadratic lifting し, マシン整数を越えたあたりで linear lifting に切替えるということが行なわれる.

3. 試し割り

   この部分の計算量は最悪で $2^{\deg(f)}$ のオーダとなる. これは, 因子の候補のあらゆる組合せをとって試し割りを行なっているからである. これを避ける多項式時間計算量アルゴリズムは Lenstra 等により lattice アルゴリズムとして提案されているが, あくまで漸近計算量における効率化であり, ここでは述べない. 実際には, この試し割りが使われるため, 一回の試し割りを少しでも高速にすることは重要である. これは, 試し割りの前処理として,

   $$g|f \text{ ならば, 整数 } a \text{ に対し } g(a)|f(a)$$

   を利用して,

   - 定数項どうしの試し割り ($g|f$ ならば $g(0)|f(0)$)
   - 係数の和どうしの試し割り ($g|f$ ならば $g(1)|f(1)$)

   などによる前処理が考案されており, それぞれ効果を発揮している.

6.8 多変数多項式の因数分解, GCD, 無平方分解

6.8.1 一般化された Hensel 構成

2 変数以上の多項式 (以下, 多変数多項式と呼ぶ) の場合, 因数分解, 無平方分解, GCD は複雑かつ再帰的に結び付いていて, その全体像を把握するのは容易ではない. 例えば, 多変数多項式 f の無平方分解は, 原理的には 1 変数の場合と全く同様に計算できるが, ある主変数を固定した時, 1 変数の時には単なる整数であった $cont(f)$ が, 多変数の場合には 1 変数少ない多項式となり, $cont(f)$ の無平方分解が必要となり, また, $cont(f)$ 自身を求める際に, やはり 1 変数少ない多項式の GCD を実行する必要がある. この事情は因数分解についても同じである. 多項式の GCD については Euclid の互除法およびその改良版である PRS 法を用いれば原理的に可能ではあるが, この方法は, GCD が 1 の場合に最も時間がかかり, かつ実際に現れるほとんどの場合は GCD が 1 であることを考えると, サブアルゴリズムとして PRS を用いることは避けたい. これに代わるものとして, 一般化された Hensel 構成がある. この方法は, 1 変数多項式の因数分解と同様, ある準同型による像から因子の「タネ」を求め, それから Hensel 構成により因子の候補を求めるものである. しかも, この方法は,「タネ」を求めさえすればその後の計算は同一であるため, 因数分解に限らず, GCD, 無平方分解にも応用できる.

以下では,

$$R = \mathbb{Z}[x_1, \cdots, x_n], \quad I = (x_2 - a_2, \cdots, x_n - a_n)$$

($(x_i - a_i$ で生成される R のイデアル) とする. I に対し $\phi_I : R \to \mathbb{Z}[x_1]$ を

$$\phi_I(f) = f(x_1, a_2, \cdots, a_n)$$

で定義する. $f \in R$ に対し, $\mathrm{lc}(f)$ は, x_1 を主変数とみた時の主係数を表すとする.

命題 6.39 (Moses-Yun[28]) $f \in R, p \in \mathbb{Z}$ は素数,

$$f \equiv \prod_i f_{1i} \bmod (p^l, I), \quad f_{1i} \in \mathbb{Z}_{p^l}[x_1], \quad \deg(f) = \sum_i \deg(f_{1i})$$

f_{1i} は GF(p) 上互いに素, $\phi_I(\mathrm{lc}(f)) \neq 0 \bmod p$ とする. この時, 任意の k に対し, $f_{ki} \in \mathbb{Z}_{p^l}[x_1, \cdots, x_n]$ が存在して, $f \equiv \prod f_{ki} \bmod (p^l, I^k)$ かつ $f_{ki} \equiv f_{1i} \bmod (p^l, I), \quad \deg(f_{ki}) = \deg(f_{1i})$.

[証明] k に関する帰納法で示す. k が 1 の時は正しい. k まで正しいとする.

$$f_{(k+1)i} = f_{ki} + h_i \quad (h_i \in (p^l, I^k))$$

なる形を仮定すると,

$$\prod_i f_{(k+1)i} \equiv \prod_i f_{ki} + \sum_i h_i \prod_{j \neq i} f_{kj} \bmod (p^l, I^{k+1})$$

一方, $f \equiv \prod_i f_{ki} \bmod (p^l, I^k)$ より,

$$f = \prod_i f_{ki} + h \quad (h \in (p^l, I^k))$$

と書ける. よって,

$$h \equiv \sum_i h_i \prod_{j \neq i} f_{kj} \equiv \sum_i h_i \prod_{j \neq i} f_{1j} \bmod (p^l, I^{k+1})$$

となるように, $h_i \in (p^l, I^k), \deg(h_i) \leq \deg(f_{1i})$ を決めることができることを示せばよい.

$$h = \sum_\alpha h_\alpha(x_1) \prod_{j \geq 2} (x_j - a_j)^{\alpha_j} \bmod I^{k+1}$$
$$h_i = \sum_\alpha h_{i\alpha}(x_1) \prod_{j \geq 2} (x_j - a_j)^{\alpha_j} \bmod I^{k+1}$$

と書くと, 各係数に対して

$$h_\alpha \equiv \sum_i h_{i\alpha} \prod_{j\neq i} f_{1j} \bmod p^l$$

となるように $h_{i\alpha}$ を選べればよいが, これは, 次の命題により次数の条件を込めて可能である. □

命題 6.40 $f_i \in \mathbb{Z}[x], p$ は素数, $p \not| \mathrm{lc}(f_i)$, $f_i \bmod p$ は互いに素とする. この時, 任意の k, 任意の $c \in \mathbb{Z}[x]$ に対し, $c_i \in \mathbb{Z}[x]$ が存在して,

$$\sum_i c_i \prod_{j\neq i} f_j \equiv c \bmod p^k, \quad \deg(c_i) < \deg(f_i)(i \geq 2).$$

さらに, $\deg(c) < \sum_i \deg(f_i)$ ならば, $\deg(c_1) < \deg(f_1)$ かつ, このような c_i は p^k を法として一意的.

[証明] $\bmod p$ における条件から, $e_{1i} \in GF(p)[x]$ が存在して, $\sum_i e_{1i} f_i \equiv 1 \bmod p$. これを再帰的に用いると, 任意の k に対して $e_{ki} \in \mathbb{Z}_{p^k}[x]$ が存在して $\sum_i e_{ki} f_i \equiv 1 \bmod p^k$. 各 f_i の主係数が p で割り切れないことから, p^k を法とする多項式除算が可能となるから, この式の両辺に c を掛けて, $i \geq 2$ に対し, $c \cdot e_{ki}$ を f_i で割った余りを c_i とし, 残りを $\prod_{j\neq 1} f_j$ の係数にまとめればよい. ここで, $\deg(c) < \sum_i \deg(f_i)$ ならば, $\deg(c_1) + \sum_{j\neq 1} \deg(f_i) < \sum_i \deg(f_i)$ より $\deg(c_1) < \deg(f_1)$ で, 一意性は, $k = 1$ における一意性から帰納法で証明できる. □

命題 6.41 命題 6.39 において, 主係数が (p^l, I^k) を法として等しい f_{ki} は (p^l, I^k) を法として一意的である.

証明は, 前二つの命題を用いれば, 一変数の場合と同様にできる.

命題 6.42 (Gel'fond[19]) $f \in \mathbb{C}[x_1, \cdots, x_n]$ に対し, 各変数に対する次数を d_i とすると $|f$ の因子の係数 $| \leq e^{d_1+\cdots+d_n} \cdot \max(|f$ の係数 $|)$.

命題 6.43 $f \in R$ が無平方ならば, 無限に多くの I に対し $\mathrm{lc}(\phi_I(f)) \neq 0$ かつ $\phi_I(f)$ は無平方.

証明は, f の無平方性が, f と f' の x_1 に関する終結式が 0 でないことと同値であることから明らか.

6.8.2 多変数多項式の因数分解

以上により, R における因数分解は次のように行なわれる.

アルゴリズム 6.44

Input : $f(x) \in R$; f は無平方かつ x_1 について原始的

Output : $\{f_1, f_2, \cdots\}$ $f = f_1 f_2 \cdots$ は f の既約分解

$a \leftarrow \deg(f) = \deg(f_a(x_1))$ かつ $f_a(x_1) = f(x_1, a_2, \cdots, a_n)$ が無平方な

$a = (a_2, \cdots, a_n) \in \mathbb{Z}^{n-1}$

$F_1 \leftarrow \{f_{1i}\} \leftarrow f_a(x_1)$ の $\mathbb{Z}$ 上での既約分解

if $|F_1| = 1$ then return $\{f\}$

$p \leftarrow f_a(x_1)$ の既約分解で用いた p

$F_1 = \{f_{11}, \cdots, f_{1m}\}$ とする

$l \leftarrow 1$

$F \leftarrow \emptyset$

while ($2l \leq m$) {

  $S \leftarrow S \subset F_1$, $|S| = l$ なる新しい部分集合

  if S が存在しない then $l \leftarrow l + 1$

  else {

    $g_1 \leftarrow \prod_{s \in S} s$, $\quad h_1 \leftarrow \prod_{s \in F_1 \backslash S} s$

    $g_1 \leftarrow \mathrm{lc}(f_a)/\mathrm{lc}(g_1) \cdot g_1$, $\quad \mathrm{lc}(g_1) \leftarrow \mathrm{lc}(f)$

    $h_1 \leftarrow \mathrm{lc}(f_a)/\mathrm{lc}(h_1) \cdot h_1$, $\quad \mathrm{lc}(h_1) \leftarrow \mathrm{lc}(f)$

    $B \leftarrow \mathrm{lc}(f)f$ の Gel'fond bound

    $l \leftarrow p^l > 2B$ なる整数 l

    $k \leftarrow f$ の $x_2, \cdots, x_n$ に関する全次数 $+1$

    $\mathrm{lc}(f)f_a = g_1 h_1 \bmod I$ から $\mathrm{lc}(f)f = g_k h_k \bmod p^l, I^k$ なる g_k, h_k を

    Hensel 構成で求める.

    $g \leftarrow g$ の各係数に p^l の整数倍を加えて絶対値が $p^l/2$ 以下としたもの

    if$g|\mathrm{lc}(f)f$ {

      $g \leftarrow \mathrm{pp}(g)$ (primitive part), $\quad f \leftarrow f/g$, $\quad F_1 \leftarrow F_1 \backslash S$

    }

  }

}

if $(f \neq 1)$ then $F \leftarrow F \cup \{f\}$

return F

このアルゴリズムを実現する場合, 一変数の場合とは異なる問題がいくつか生ずる.

1. 非零代入問題

   Hensel 構成の手順を見ると, $x_i - a_i$ に関する同次成分を取り出す操作が必要になることがわかる. a_i が全て 0 ならば, 再帰表現された多項式のいくつかの項の取り出しに過ぎないが, 0 でない a_i がある場合, あらかじめ平行移動により a_i を 0 とするか, 微分, 代入の操作を用いて, 必要な係数を計算する必要がある. 元の多項式が疎でも平行移動を行なうと密な多項式となるため, a_i のうち 0 をなるべく多く選びたいが, a_i に対する条件を満たすためにやむなく 0 でない a_i を用いることがあり得る.

2. 主係数問題

   主係数が 1 でない場合を扱えるよう, 1 変数の場合と同様に, 因子の候補の主係数を与えられた多項式の主係数と等しくしてある. これにより, Hensel 構成の際に, 不必要な項の増加を押えられるが, 主係数が大きな多項式の場合必要な Hensel 構成の段数の増加を招く. このため, あらかじめ因子の候補に対応する主係数を何らかの方法により決めてしまう方法が Wang [44] により提案されている.

3. ニセ因子の問題

   ここで述べたアルゴリズムにおいては, 2 つの因子を Hensel 構成するという手順にしてある. これは,

   - 多くの因子を同時に Hensel 構成する場合, 主係数をモニックのまま行なうことは, 不必要な項の増加を招き, 全ての因子の主係数を与えられた多項式の主係数に合わせることは, Hensel 構成の段数の増加につながる.
   - 候補が真の因子のイメージであることを期待している. 真の因子の場合, 次数の評価で得られる段数より前に候補が安定するので, その時点で試し割りをして Hensel 構成を切り上げることもできる.

   などの理由による. ニセ因子により Hensel 構成を行なった場合, 評価で得られる段数まで Hensel 構成が進行し, 巨大な候補を生成してしまう. これは, 大きな無駄となるため, 各変数の次数を常にチェックして, 候補のどれかの変数に関する次数が与えられた多項式の次数を越えた時点で Hensel 構成を中断するなどの操作が必要となる.

ここで述べた方法は, EZ 法と呼ばれ, n 変数を 一変数に落して候補を計算するものであった. しかし, 一度に一変数に落すことは非零代入問題や, ニセ因子の出現の確率を大きする. このため, 変数の個数を一つずつ落していく EEZ 法 [44] が考案された. この方法によれば, 非零代入問題も, 一変数ずつに対するものになるため項の数が指数的に増加することはなく, 変数を一つ増やして Hensel 構成を行なう際, ニセ因子は次第に排除されていくため, ニセ因子に対して強いアルゴリズムとなる.

6.8.3 多変数多項式の無平方分解及び GCD

一般化された Hensel 構成は, 多変数多項式の無平方分解, GCD にも応用できる. このとき問題となるのは, Hensel 構成の出発点となる因子 (命題 6.39 の f_{1i}) が, 法 p で互いに素であるという条件である. $\mathrm{GCD}(g,h)$ の計算においては, f_{1i} として

$$\{\mathrm{GCD}(\phi_I(g),\phi_I(h)),\phi_I(g)/\mathrm{GCD}(\phi_I(g),\phi_I(h))\}$$

をとれれば自然であるが, 一般にこれらが法 p で互いに素であることは期待できない. しかし, g の既約因子を全て含む無平方因子 $g_s = g/\mathrm{GCD}(g,g')$ を求める際に必要な $g_1 = \mathrm{GCD}(g,g')$ は,

$$\mathrm{GCD}(g_1, g'/g_1) = 1$$

より, g' の因子として, 一般化された Hensel 構成で計算である. また, 一般に, g が無平方ならば可能である. よって $\mathrm{GCD}(g_s,h)$ は 一般化された Hensel 構成で計算でき, $\mathrm{GCD}(g,h)$ の既約因子を全て含む無平方な多項式となり, これを用いて $\mathrm{GCD}(g,h)$ を求めることができる. また, g_s さえ求まれば, g の無平方分解自身も一般化された Hensel 構成で求めることができる.

しかし, 無平方分解において, g が重複度の大きい因子を含んでいるとき, g_s の計算に多くの時間がかかる. これを避けるため, 重複度の最も高い因子から求めていく方法が考案された. これは, 次の命題に基づく.

命題 6.45 $f \in K[x]$ に対し, $f = hg^e \quad (h,g \in K[x]), \quad \mathrm{GCD}(h,g) = 1$ かつ g が無平方ならば $g|(d/dx)^{e-1}f$ で

$$\mathrm{GCD}(g, ((d/dx)^{e-1}f)/g) = 1$$

[証明] $e = 1$ のとき明らか. $e \geq 2$ のとき, $(d/dx)^{e-1}(g^e) \equiv e!gg'^{e-1} \bmod g^2$ がいえる. よって, $(d/dx)^{e-1}f \equiv h(d/dx)^{e-1}(g^e) \equiv e!hgg'^{e-1} \bmod g^2$ となり, $g|(d/dx)^{e-1}f$. さらに, $((d/dx)^{e-1}f)/g \equiv e!hg'^{e-1} \bmod g$ で, $\mathrm{GCD}(g,h) = 1$, $\mathrm{GCD}(g,g') = 1$ より $\mathrm{GCD}(g,((d/dx)^{e-1}f)/g) = 1$. □

[46] のアルゴリズムは, 多変数の場合, まず, 代入により一変数に落した多項式を無平方分解する. 得られた無平方分解の, 重複度の最も高い因子 g_0 と, f の $e-1$ 回微分から, 一般化された Hensel 構成により f の, 重複度の最も高い因子 g を復元するのである. そして, この Hensel 構成が失敗した場合には, 代入した値が妥当なものではなかったとして, 値を取り直す. 妥当な値が存在することは, 無平方な多変数多項式に対する妥当な代入値の存在と同様に言える. この方法のよい点は,

- (重複度 -1) だけ多項式の次数が落せる
- 無平方因子を直接求めることができるため, f の既約因子すべての積を求める場合に比較して計算の手間を減らすことができる
- 全体に対して妥当でない代入でも, 重複度最大の因子には影響がない場合がある.
- ある多項式の冪になっている場合に高速に分解できる.

などがある. さらに, この方法は, 一変数多項式の無平方分解にも応用できる. すなわち, 一変数多項式に対しては, 法 p での無平方分解を用いて整数上の無平方因子を, 重複度最大の因子から復元していく. この際, 復元方法としては, [46] では中国剰余定理による方法を提案しているが, Hensel 構成によるものも可能である.

以上, 一般化された Hensel 構成の応用について述べたが, これらを計算機上にインプリメントすることはかなりの大仕事となる. これは, 最初に述べたように, 因数分解, GCD, 無平方分解が再帰的に結合し, かつそれらは様々に付帯状況の異なる Hensel 構成により計算されなければならない. さらに, これらを効率よく計算するための種々の工夫を入れれば, プログラムは相当に大きなものとなり, 当然 debug も困難なものになる.

6.9 代数体上の因数分解

この節では, K を Q の有限次拡大体とし, $K[x]$ における因数分解を考える. ここで述べるのは, Trager [42] による, ノルムを用いる方法である. $K=\mathbb{Q}(\alpha)$ という単拡大に対しては, この他に, Hensel 構成を用いる方法がいくつか提案されているがここでは述べない. Trager の方法は, $K(\alpha)$ 上での因数分解を, K 上の因数分解に帰着させるもので, $\mathbb{Q}$ 上の因数分解さえ完備していれば, $K=\mathbb{Q}(\alpha_1,\cdots,\alpha_r)$ 上での因数分解が可能になる.

以下, K を, その上で多項式因数分解が可能な体, $g\in K[t]$ を $\deg(g)=m$ なる既約多項式, α を g の一つの根とする. $K(\alpha)=K[\alpha]=K[t]/(g)$ である. $L\supset K$ を g の最

小分解体とする. $\alpha_1 = \alpha, \alpha_2, \cdots, \alpha_m$ を g の相異なる根とすると, $L = K(\alpha_1, \cdots, \alpha_m)$ とかける.

定義 6.46 $f \in K(\alpha)[x]$ に対し, $\mathrm{Norm}(f)$ を

$$\mathrm{Norm}(f) = \prod_i f(x, \alpha_i)$$

で定義する. L/K の Norm の性質により, $\mathrm{Norm}(f) \in K[x]$. また,

$$\mathrm{Norm}(f) = \mathrm{res}_t(f(x,t), g(t)).$$

命題 6.47 $f \in K(\alpha)[x]$ が既約ならば, ある既約多項式 $h \in K[x]$ が存在して,

$$\mathrm{Norm}(f) = h^l.$$

[証明] $\mathrm{Norm}(f) = ab, \quad a, b \in K[x], \quad \mathrm{GCD}(a,b) = 1$ と書けたとする. $f|\mathrm{Norm}(f)$ で, f は $K(\alpha)$ 上既約より, $f|a$ としてよい. この時

$$\mathrm{Norm}(f)|\mathrm{Norm}(a) = a^m.$$

よって $\mathrm{GCD}(\mathrm{Norm}(f), b) = 1$ となり, $b|\mathrm{Norm}(f)$ だから, $b = 1$. よって, $\mathrm{Norm}(f)$ は 2 つ以上の異なる既約因子を含み得ない. □

系 6.48 $f \in K(\alpha)[x]$ が無平方とする時, $\mathrm{Norm}(f)$ が無平方ならば, $\mathrm{Norm}(f) = \prod f_i$ を $K[x]$ における既約分解とするとき, $f = \prod \mathrm{GCD}(f, f_i)$ は f の $K(\alpha)[x]$ における既約分解を与える.

[証明] h を f の $K(\alpha)[x]$ における既約因子とする. h はある f_i を割り切る. $\mathrm{Norm}(h)$ は $K[x]$ の既約多項式の冪で, $\mathrm{Norm}(h)$ は無平方な $\mathrm{Norm}(f)$ を割り切るから, $\mathrm{Norm}(h)$ 自身が $\mathrm{Norm}(f)$ の既約因子. $\mathrm{Norm}(h)$ は $\mathrm{Norm}(f_i)$ の因子でもあるから, $f_i = \mathrm{Norm}(h)$. 今, h と異なる因子 h_1 が f_i を割れば, 同様に $f_i = \mathrm{Norm}(h_1)$. $hh_1|f$ より

$$\mathrm{Norm}(hh_1) = {f_i}^2|\mathrm{Norm}(f).$$

これは $\mathrm{Norm}(f)$ が無平方であることに反する. よって f_i は f のただ一つの既約因子を含む. □

以上により, $\mathrm{Norm}(f)$ が無平方のときには, GCD により f の既約因子が $K[x]$ における既約分解により得られることがわかった. $\mathrm{Norm}(f)$ が無平方でない場合にも, 適当な変数変換により無平方化できることが次の命題よりわかる.

命題 6.49 $f \in K(\alpha)[x]$ が無平方ならば, $\mathrm{Norm}(f(x-s\alpha))$ が無平方とならない $s \in K$ は有限個.

[証明] f の根を $\beta_1, \cdots, \beta_m$ とすると, 仮定よりこれらは相異なる. すると, $f(x-s\alpha_i)$ の根は,

$$\beta_1 + s\alpha_i, \cdots, \beta_m + s\alpha_i$$

となる. これより $\mathrm{Norm}(f(x-s\alpha))$ が無平方でないのは, ある i, j, k, l に対して,

$$\beta_j + s\alpha_i = \beta_k + s\alpha_l$$

の場合に限る. このような条件を満たす s は有限個しかない. □

以上により, 次のアルゴリズムが得られる.

アルゴリズム 6.50

Input : 無平方な $f(x, \alpha) \in K(\alpha)[x]$, $s \in \mathbb{Z}$

Output : f の $K(\alpha)$ 上の既約因子 $\{f_1, \cdots, f_r\}$

again:

$r \leftarrow \mathrm{res}_t(f(x-st, t), g(t))$

if (r が無平方でない) then

  $s \leftarrow s+1$; goto again

$r(x) = \prod r_i(x)$: r の K 上での既約分解

$z \leftarrow f$

for each i do {

  $f_i \leftarrow \mathrm{GCD}(h(x+s\alpha), z(x, \alpha))$

  $t \leftarrow z/f_i$

}

このアルゴリズムにおいて, ボトルネックとなり得る部分が数多くある.

1. 終結式の計算

   終結式の計算は, 部分終結式, 行列式の modular 演算などを組み合わせて行なうが, 実際の計算時間を反映する計算量が与えられていないため, アルゴリズムの選択が困難である. また, いずれにしても, 終結式の計算は一般に時間がかかり, しかもその終結式が無平方でなければ捨てられるため, そのコストの大きさは問題である.

2. K 上での既約分解

   K がまたある拡大体になっている場合にはこのアルゴリズムが再帰的に用いられることになる. この際, ノルムをとることは, 次数が拡大次数倍されるため, 因数分解すべき多項式の次数が急速に増大する. 最終的に $\mathbb{Q}$ 上の因数分解を行なうことになるが, $\mathbb{Q}$ 上の因数分解自体のコストの問題がある.

3. $K(\alpha)$ 上での GCD 計算

   一般に, 拡大体上での GCD の計算は, 整数上でのそれに比較して極めて困難である. 特に, Euclid 互除法を用いる場合, 中間式膨張は, 定義多項式 g が大きい場合, 極めて激しい. これをさけるためいくつかの modular 計算法が提案されている.

この中で, 2 および 3 は止むを得ない問題であるが, 1 に関しては, 無平方でないノルムを利用することにより, ある程度回避できる.

命題 **6.51** $f \in K(\alpha)[x]$ が無平方,

$$\mathrm{Norm}(f) = \prod_i {f_i}^{n_i}$$

($f_i \in K[x]$ は既約, n_i は 1 とは限らない) とすると, $\mathrm{GCD}(f, f_i)$ は f の定数でない因子で,

$$f = \prod \mathrm{GCD}(f, f_i).$$

特に, $\mathrm{Norm}(f)$ の重複度 1 の因子 f_i に対しては, $\mathrm{GCD}(f, f_i)$ は既約.

[証明] h を f の任意の既約因子とすると, ある f_i が存在して $h|f_i$ より $h|\prod \mathrm{GCD}(f, f_i)$. f は無平方より $f|\prod \mathrm{GCD}(f, f_i)$. $\mathrm{GCD}(f, f_i)$ は互いに素より, $f = \prod \mathrm{GCD}(f, f_i)$. さて, f の既約因子のノルムはある既約多項式の冪より, 任意の f_i に対して, その適当な冪は f の既約因子のノルムとなっている. よって, f_i はある f の既約因子を含むので, $\mathrm{GCD}(f, f_i)$ は定数でない. 特に, f_i が $\mathrm{Norm}(f)$ の重複度 1 の因子ならば, それ自身 f のある既約因子 h のノルムとなる. これは, 前の系と同様にして, f_i は h 以外の因子を含まない □

この命題により, ノルム (終結式) が, 二つ以上の既約因子を持てば, その分解は, f の自明でない因数分解を与えることがわかる. もし, ノルムが無平方でなければ, ここで生成した因子に対し, s を取り直してこのアルゴリズムを再帰的に適用することになるが, 問題のサイズは小さくなっている. 特に, ノルム自体が無平方でなくても重

複度が 1 の因子に対しては, GCD が既約因子であることが保証されている. また, ノルムが無平方でない場合にも, K 上での因数分解の前に, 無平方分解によりノルムを分解できるため, K 上の因数分解の計算時間も短縮できる. この改良を採り入れたアルゴリズムを次に示す.

アルゴリズム 6.52

Input : 無平方な $f(x,\alpha) \in K(\alpha)[x]$, $s \in \mathbb{Z}$

Output : f の $K(\alpha)$ 上の既約因子 $\{f_1, \cdots, f_r\}$

$r \leftarrow \mathrm{res}_t(f(x-st,t), g(t))$

$r(x) = \prod r_i(x)^{m_i}$: r の K 上での既約分解

$z \leftarrow f$

for each i do {

  $g_i \leftarrow \mathrm{GCD}(r_i(x+s\alpha), z(x,\alpha))$

  if $m_i = 1$ then g_i は既約

  else $(g_i, s+1)$ にこのアルゴリズムを適用

  $t \leftarrow z/g_i$

}

例 6.53 [1]

$$\begin{aligned} f(x) &= x^{16} - 136x^{14} + 6476x^{12} - 141912x^{10} + 1513334x^8 - 7453176x^6 + 13950764x^4 \\ &\quad -5596840x^2 + 46225 \end{aligned}$$

$f(x)$ は $\alpha = \sqrt{2}+\sqrt{3}+\sqrt{5}+\sqrt{7}$ の $\mathbb{Q}$ 上の最小多項式である. $\mathbb{Q}(\alpha)/\mathbb{Q}$ はガロア拡大であり, $f(x)$ は $\mathbb{Q}(\alpha)$ 上一次式の積に分解する. この因数分解をアルゴリズム 6.50 で求める場合, $F(x) = \mathrm{Norm}(f(x-s\alpha))$ が無平方となる $s \in \mathbb{Z}$ を見つけて $F(x)$ を有理数体上で因数分解する必要があるが, このような $F(x)$ は, 全ての素数 p に対し一次または二次式に分解してしまう. 例えば 二次因子の積 128 個に分解した場合, 有理数体上の一つの既約因子 (16 次) は, 128 個から 8 個を選んで得られることになるが, これは明らかに組合せ爆発を起こしていて計算不可能である. 一方で, 例えば $s = -1$ の場合 $F(x)$ は無平方にならないが, 16 個の異なる既約因子が無平方分解およびそれに続く既約因子分解により容易に得られる.

$$\begin{aligned} F(x) &= x^{16}(x^2-28)^8(x^2-20)^8(x^2-8)^8(x^2-12)^8 \\ &\quad \cdot(x^4-64x^2+64)^4(x^4-40x^2+16)^4 \end{aligned}$$

$$\cdot(x^4-80x^2+256)^4(x^4-56x^2+144)^4$$
$$\cdot(x^4-72x^2+400)^4(x^4-96x^2+64)^4$$
$$\cdot(x^8-240x^6+12512x^4-203520x^2+891136)^2$$
$$\cdot(x^8-192x^6+8576x^4-110592x^2+102400)^2$$
$$\cdot(x^8-224x^6+11264x^4-143360x^2+409600)^2$$
$$\cdot(x^8-160x^6+5632x^4-61440x^2+147456)^2$$
$$\cdot(x^{16}-544x^{14}+103616x^{12}-9082368x^{10}+387413504x^8-7632052224x^6$$
$$+57142329344x^4-91698626560x^2+3029401600)$$

これらの各因子から $f(x)$ の全ての一次因子が得られる. 一般に, 最小分解体を求めるための因数分解など, 多項式を, その根を添加した体上で因数分解する場合にアルゴリズム 6.52 が有効となる場合がある.

6.10 応用 — 有理関数の不定積分

代数体上の GCD を用いて, 有理関数の不定積分を, 積分記号を含まない形で表示する方法について述べる.

一般に, 有理関数 $f(x)=n(x)/d(x)$ $(n,d\in\mathbb{Q}[x])$ の不定積分は, f の部分分数分解により計算できる. しかし, そのために分母 d を 1 次因子の積に分解するには d の最小分解体を求めることが必要となる. また, 不定積分自体に, d の分解により現れた代数的数が現れるとは限らない.

例 **6.54** $f=d'/d$ ならば $\int f dx=\log d$.

このことから, 最小の代数的数の添加で不定積分を表示することを考える.

6.10.1 有理部分の計算

$f=n/d$, $n,d\in\mathbb{Q}[x]$ とする. もし $\deg(n)\geq\deg(d)$ ならば,

$$n=qd+r,\quad q,d\in\mathbb{Q}[x],\deg(r)<\deg(d)$$

により, $f=q+r/d$ と書ける. このとき $\int f dx=\int q dx+\int r/d dx$ で, 多項式の不定積分は自明だから, あらかじめ $\deg(n)<\deg(d)$ としてよい.

命題 6.55 $\deg(n) < \deg(d)$ とする. $d_r = \mathrm{GCD}(d, d')$, $d_l = d/d_r$ とおくと, $\deg(n_r) < \deg(d_r)$, $\deg(n_l) < \deg(d_l)$ なる $n_r, n_l \in \mathbb{Q}[x]$ が一意的に存在して,

$$\int \frac{n}{d} dx = \frac{n_r}{d_r} + \int \frac{n_l}{d_l} dx$$

[証明] $d = \prod_i d_i^i$ なる無平方分解に対応して,

$$\frac{n}{d} = \sum_i \sum_j \frac{n_{ij}}{d_i^j} \quad (\deg(r_{ij}) < \deg(d_i))$$

なる部分分数分解が命題 5.12 により存在する. $d_r = \prod_i d_i^{i-1}$, $d_l = \prod_i d_i$ である. d_i は無平方だから, $\mathrm{GCD}(d_i, d_i') = 1$. よって, 各 n_{ij} に対し, $s_{ij}, t_{ij} \in \mathbb{Q}[x]$ $(\deg(s_{ij}) < \deg(d_i) - 1, \deg(t_{ij}) < \deg(d_i))$ が存在して,

$$s_{ij} d_i + t_{ij} d_i' = n_{ij}.$$

これを用いて, $j > 1$ のとき

$$\int \frac{n_{ij}}{d_i^j} dx = \int \frac{s_{ij}}{d_i^{j-1}} dx + \int \frac{t_{ij} d_i'}{d_i^j} dx$$

ここで,

$$\left(\frac{1}{1-j} \cdot \frac{1}{d^{j-1}}\right)' = \frac{d_i'}{d_i^j}$$

より,

$$\int \frac{t_{ij} d_i'}{d_i^j} dx = \frac{1}{1-j} \cdot \frac{1}{d_i^{j-1}} \cdot t_{ij} - \int \frac{1}{1-j} \cdot \frac{1}{d_i^{j-1}} t_{ij}' dx.$$

よって

$$\int \frac{n_{ij}}{d_i^j} dx = \frac{t_{ij}}{(1-j) d_i^{j-1}} + \int \frac{s_{ij} + \frac{t_{ij}'}{j-1}}{d_i^{j-1}} dx$$

$\deg(s_{ij} + \frac{t_{ij}'}{j-1}) < \deg(d_i) - 1$ より, この積分の第 2 項を $\int \frac{n_{i,j-1}}{d_i^{j-1}} dx$ に加えても, 分子の次数は増大しない. よって, この操作を $j > 1$ なる項に対して繰り返すことにより,

$$\int \frac{n}{d} dx = \sum_i \sum_{j>1} \frac{t_{ij}}{(1-j) d_i^{j-1}} + \sum_i \int \frac{u_i}{d_i} dx \quad (\deg(u_i) < \deg(d_i))$$

なる形に変形できる. 第 1 項, 第 2 項の披積分関数をまとめてそれぞれ $\frac{n_r}{d_r}$, $\frac{n_l}{d_l}$ とおけば次数の条件を満たす. 一意性は明らか. □

この命題で保証された n_r, n_l は, 未定係数法で決定することができる.

アルゴリズム 6.56

Input: $f(x) = n(x)/d(x) \in \mathbb{Q}(x)$, $\deg(n) < \deg(d)$

Output: $\int f(x)dx = \dfrac{n_r(x)}{d_r(x)} + \int \dfrac{n_l(x)}{d_l(x)}dx$, d_l は無平方で $n_r, d_r, n_l, d_l \in \mathbb{Q}[x]$ なる分解

ただし $\deg(n_r) < \deg(d_r)$, $\deg(n_l) < \deg(d_l)$

$d_r \leftarrow \mathrm{GCD}(d, d')$, $\quad d_l \leftarrow d/d_r$

$m_r \leftarrow \deg(d_r)$, $\quad m_l \leftarrow \deg(d_l)$

$n_r \leftarrow \sum_{i=0}^{m_r-1} a_i x^i$, $\quad n_l \leftarrow \sum_{i=0}^{m_l-1} b_i x^i$

$n = n_r' d_l - (\frac{d_l d_r'}{d_r}) n_r + d_r n_l$ から a_i, b_i を求める

return $\int f(x)dx = \dfrac{n_r(x)}{d_r(x)} + \int \dfrac{n_l(x)}{d_l(x)}dx$

6.10.2 対数部分の計算

$f(x) = n(x)/d(x)$, $\mathrm{GCD}(n, d) = 1$, $\deg(n) < \deg(d)$ で d は無平方とする. このとき

$$\int f dx = \sum_i c_i \log r_i$$

と書ける. 一般に $c_i \in \mathbb{Q}, r_i \in \mathbb{Q}[x]$ とは限らず, 何らかの代数的数を含む可能性があるが, この代数拡大を最小限にするような表示を求めたい.

命題 6.57 (Rothstein[12])

K を複素数体 $\mathbb{C}$ の部分体とし, $f(x) = n(x)/d(x)$, $n, d \in K[x]$, $\mathrm{GCD}(n, d) = 1$, $\deg(n) < \deg(d)$ で d は無平方, 無平方とする. このとき,

$$n/d = \sum_{i=1}^{n} c_i v_i'/v_i$$

ただし $c_i \in \mathbb{C}$ は相異なり, $v_i \in \mathbb{C}[x]$, v_i はモニック, 無平方で互いに素, と書けたならば, c_i は

$$R(z) = \mathrm{res}_x(n - zd', d) \in K[z]$$

の根で,

$$v_i = \mathrm{GCD}(n - c_i d', d).$$

[証明] claim 1 $v = d$.

$v = \prod_{i=1}^{n}$ とおくと,

$$nv = d \sum_{i=1}^{n} c_i v_i'(v/v_i).$$

$\mathrm{GCD}(n,d)=1$ より $d|v$. 一方で, $v_i|$ 右辺より, もし $v_i \not| d$ ならば $v_i|c_i v_i'(v/v_i)$ となるが, これは v_i に関する条件より不可能. よって $v_i|d$. 結局 $v|d$ となり $v=d$. □

claim 2 $v_i = \mathrm{GCD}(n-c_i d', d)$.

claim 1 より $n=\sum_{i=1}^n c_i v_i'(v/v_i)$. $d'=\sum_{i=1}^n v_i'(v/v_i)$ より,

$$n - c_i d' = \sum_{j\neq i}(c_j - c_i)v_j'(v/v_j).$$

これから $v_i|n-c_i d'$ がわかる. よって $v_i|\mathrm{GCD}(n-c_i d', d)$. 一方で, $j\neq i$ のとき,

$$\mathrm{GCD}(n-c_i d', v_j) = \mathrm{GCD}((c_j-c_i)v_j'(v/v_j), v_j) = 1$$

より, $v_i = \mathrm{GCD}(n-c_i d', d)$. □

claim 2 より, c_i が $R(z)$ の根でなければならないこともわかる.

claim 3 $\{c_1, \cdots, c_n\} = R(z)$ の根全体

c が $R(z)$ の根ならば, $v_0 = \mathrm{GCD}(n-cd', d)$ は自明でない d の因子. よって v_0 の既約因子 g を一つとれば, ある v_i が存在して $g|v_i$.

$$g|(n-cd') = \sum_{j=1}^n (c_j - c)v_j'(v/v_j)$$

より, $g|(c_i-c)v_i'(v/v_i)$. これは $c_i = c$ のときのみ可能. □

系 6.58 K を複素数体 $\mathbb{C}$ の部分体とし, $f(x)=n(x)/d(x)$, $n,d\in K[x]$, $\mathrm{GCD}(n,d)=1$, $\deg(n)<\deg(d)$ で d は無平方, モニックとし,

$$R(z) = \mathrm{res}_x(n - zd', d) \in K[z]$$

とする. K_R を $R(z)$ の最小分解体とすれば, K_R が $\int n/d dx$ を表示するための最小の K の拡大.

[証明] F を K の拡大体とし, $c_i\in F$, $v_i\in F[x]$ による $n/d=\sum_i c_i v_i'/v_i$ なる表示をとると, 体の拡大なしに, 前命題の c_i v_i に対する条件が満たされるようにできる. このとき, c_i は $R(z)$ の根で $c_i\in F$ だから, $K_R\subset F$. K_R 上でこの表示ができることは前命題で保証されている. □

Chapter 7

グレブナ基底

7.1 代数方程式の解とイデアル

体 K 上の n 変数多項式環 $R = K[x_1, \cdots, x_n]$ を考える. 以下, $(x_1, \cdots, x_n)$ を X と略記する. R の元 $f_1, \cdots, f_m$ に対し,

$$f_1 = 0, \cdots, f_m = 0 \tag{7.1}$$

を代数方程式系, あるいは単に方程式と呼ぶ. (7.1) を満たす K^n の元を (1) の解と呼ぶ. このような方程式を解いて解を求めようとする場合に最も基本的な方法は, 中学以来おなじみの消去法である.

例 **7.1** $f_1(x, y) = x^2 + y^2 - 2 = 0, f_2(x, y) = xy - 1 = 0$ を解け

解 $y^2 f_1 - (xy + 1) f_2 = y^4 - 2y^2 + 1 = 0$ より $y = x = 1$ または $y = x = -1$. これらは実際に解である. □
ここで行った計算は, f_1, f_2 に適当な多項式を掛けたものの和を作ってより変数の少ない多項式を作り出すものである.

定義 **7.2** $f_1, \cdots, f_m \in R$ に対し,

$$Id(f_1, \cdots, f_m) = \{\sum_{i=1}^{n} g_i f_i \mid i \in R\}$$

を $f_1, \cdots, f_m$ で生成されるイデアルと呼ぶ. $f_1, \cdots, f_m$ を I の生成系あるいは基底と呼ぶ.

一般に, 方程式 (7.1) が与えられた場合, $I = Id(f_1, \cdots, f_m)$ を考えれば, 消去法とは I の中から, 含まれる変数の個数が少ないものを選び出す方法と言える. I の元全て

の共通零点は (7.1) の解に一致する. イデアルの基底は一組とは限らないが, 同一のイデアルを生成する基底の共通零点は一致するから, イデアルを考える方が, 方程式の解を考える上でより自然であると言える.

例 **7.3** $Id(x^2+y^2-2, xy-1) = Id(-y^4+2y^2-1, x+y^3-2y)$

例 **7.4** (線形方程式) $Id(2a+3b-4c+d-1, 3a-2c-5d-4, a-b+4d-5, 3a+2b+2c-2d)$
$= Id(-185d+78, -185c-94, -185b-299, -185a+314)$

これらの例では, 右辺の基底は確かに解を容易に求められる形になっている. ここで大事な点は, 両辺が等しいイデアルを与えているかどうか, という点である.

定義 **7.5** イデアル I に対し I の K^n における variety $V_K(I)$ を

$$V_K(I) = \{a \in K^n \mid \text{すべての } f \in I \text{ に対し } f(a) = 0\}$$

で定義する. 混乱のない場合には $V(I)$ と書く.

系 **7.6** $I \subset J \Rightarrow V(J) \subset V(I)$

すなわち, 消去法により得られた多項式は, イデアルの元であることは保証されるが, それらはあくまで解の満たすべき必要条件であり, 実際にもとの方程式の解を表すか否かは別のチェックが必要となる. もし新たに得られた多項式集合がもとのイデアルを生成していれば, 解が等しいことは保証されている.

7.2 項順序, モノイデアル, グレブナ基底

前節で, 代数方程式の解を考える上で, 多項式イデアルを考え, その基底を取り換えて方程式を解きやすい基底に取り換えることが有効であることを示した. しかし, イデアルの概念は, 方程式を解くためだけに用いられるものではない. 特に, 解が無限個になる場合は, その解全体は代数的集合として扱われるべきものであり, 方程式の解として表現することは一般には難しい. むしろ, イデアル I あるいは環 R/I の性質からその代数的集合の成分, 次元などを知ることが重要となる. そのためにイデアルの基底が満たすべき条件として次のようなものを挙げることができる.

- ある多項式が, イデアルに属するか否か (メンバシップ) がアルゴリズムにより判定できる.

  I に属する多項式を具体的に把握するために必要である.

- イデアルを, 連立方程式系とみたとき, 解を求めやすい形をしている.

  消去法と同様の結果を与えることができる.

- イデアルの性質 (次元, 因子など) を表している.

これらの性質のうち, 第一のものに着目する.

例 **7.7** $n = 1$ の場合
$R = K[x]$ は PID (単項イデアル整域) である. すなわち任意のイデアル I はある $f \in R$ により $I = Id(f)$ と書ける. これは, f を基底とすることにより, I に対するメンバシップが,

$$g \in I \Leftrightarrow f \mid g$$

で判定できることを意味する. I の生成元が幾つか与えられている場合, f は, それらの生成元の GCD を求めることで得られる.

一変数の場合, イデアルの生成元は, そのイデアルに属する元のうち, 最も次数の小さいものをとればよかった. これは, 次のようにいいかえられる.

- 多項式の各項を降冪の順に並べたとき, 先頭の項を 頭項 と呼ぶ. この時, 頭項が, イデアルのすべての元の頭項を割り切るような元が生成元となる.

これを多変数に拡張するために, 一変数の場合と同様に, 多変数多項式の項の間に「自然な」全順序を入れる. 以下, 体 K 上の n 変数多項式環 $R = K[x_1, \cdots, x_n]$ を固定して考える. 自然数 $\mathbb{N}$ は, 0 以上の整数を表す.

定義 **7.8** $T = \{x_1^{i_1} \cdots x_n^{i_n} \mid i_1, \cdots, i_n \in \mathbb{N}\}$ とし, T の元を term (項) と呼ぶ. この時, T における全順序 $\leq$ が **term order** であるとは,

1. すべての $t \in T$ に対し $1 \leq t$
2. すべての $t_1, t_2, s \in T$ に対し $(t_1 \leq t_2 \Rightarrow t_1 \cdot s \leq t_2 \cdot s)$

を満たすことを言う.

定義 **7.9** 指数を $\mathbb{N}^n$ の元と考えて, $\mathbb{N}^n$ における term order $<$ を

1. すべての $\alpha \in \mathbb{N}^n$ に対し $0 = (0, \cdots, 0) \leq \alpha$
2. すべての $\alpha_1, \alpha_2, s \in \mathbb{N}$ に対し $(\alpha_1 \leq \alpha_2 \Rightarrow \alpha_1 + s \leq \alpha_2 + s)$

を満たすものとして定義できる.

定義 7.10 $L \subset \mathbb{N}^n$ がモノイデアルとは, 任意の $\alpha \in L, \beta \in \mathbb{N}^n$ に対し $\alpha + \beta \in L$ が成り立つことをいう. また, $S \subset \mathbb{N}^n$ に対し,

$$mono(S) = \{\alpha + \beta \mid \alpha \in S, \beta \in \mathbb{N}^n\}$$

を S で生成されるモノイデアルと呼ぶ.

定義 7.11 term 間の全順序を多項式の半順序に自然に拡張する.
$f, g \in R$ で, $f = \sum_{i \geq 0} c_i f_i$, $g = \sum_{i \geq 0} d_i g_i$ ($c_i, d_i \in K, f_i, g_i \in T, i > j \Rightarrow f_i > f_j, g_i > g_j$) とする時, $f > g$ を

$$f > g \Leftrightarrow \text{ある } i_0 \text{ が存在して } (i < i_0 \Rightarrow f_i = g_i, f_{i_0} > g_{i_0})$$

で定義する.

定義 7.12 $M = \{c \cdot t \mid c \in K, t \in T\}$ とし, M の元を **monomial** と呼ぶ.

定義 7.13 term order を一つ固定した時, 多項式 f に表れる term の中で, その order において最大のものを 頭項 (**head term**) と呼び, $HT(f)$ と書く.
$HT(f)$ の係数を, $HC(f)$ と書く.
$HC(f) \cdot HT(f)$ を $HM(f)$ と書く.
$HT(f)$ の指数を $HE(f)$ と書く. $HE(f) \in \mathbb{N}^n$ である.
さらに, $f - HM(f)$ を $red(f)$ (**reductum** of f) と書く.

補題 7.14 $\mathbb{N}^n$ の任意のモノイデアル L は有限生成.

[証明] n に関する帰納法により示す. $n = 1$ のとき, L の $\mathbb{N}$ 中での最小元 α をとれば L は α で生成される. $n - 1$ まで言えたとする. 各 $j \in \mathbb{N}$ に対し,

$$L_j = \{(\alpha_1, \cdots, \alpha_{n-1}) \in N^{n-1} \mid (\alpha_1, \cdots, \alpha_{n-1}, j) \in L\}$$

とおくと $\{L_j\}$ はモノイデアルの増大列. $L_\infty = \cup L_j$ とおくと L_∞ もモノイデアルで, 帰納法の仮定により L_∞ は有限生成. よってある j_0 が存在して $L_\infty = L_{j_0}$. このとき, L は, $j = 1, \cdots, j_0$ に対する L_j の生成元 (これは帰納法の仮定によりそれぞれ有限集合) の和集合で生成される. □

系 7.15 $\{L_i\}(i = 1, 2, \cdots)$ をモノイデアルの増大列とすれば, ある i_0 が存在して, $i \geq i_0 \Rightarrow L_i = L_{i_0}$.

系 **7.16** $\mathbb{N}^n$ の任意の部分集合は term order $<$ に関して最小元を持つ.

系 **7.17** $\leq$ を T の term order とすれば, T の真の降下列は有限で切れる. $\leq$ の R への自然な拡張についても同様である. 以下, この性質を, term order の Noether 性として引用する.

[証明] T については降下列が最小元を持つことから成り立つ. R については, もし R の真の降下列である無限列があれば, 頭項が T の真の降下列である無限列を作り出せるから矛盾. □

定義 **7.18** $S \subset R$ および term order $<$ に対し,

$$E_<(S) = \{HE(f) \mid f \in S\} \subset \mathbb{N}^n$$

と定義する. 以下混乱のない場合には $<$ を省略して $E(S)$ と書く.

補題 **7.19** イデアル I に対し, $E(I)$ はモノイデアル.

一変数多項式環におけるイデアル I の生成系 $G = \{f\}$ の性質は,

$$E(I) = mono(E(G))$$

と書くことができる. 一般の場合にもこのような性質を満たす G を考えることは有用である.

定義 **7.20** イデアル I に対し, その有限部分集合 G で,

$$E(I) = mono(E(G))$$

を満たすものを I の $<$ に関するグレブナ基底と呼ぶ.

この定義は, イデアルのすべての元の頭項が, G のいずれかの元の頭項で割り切れることを意味する.

命題 **7.21** 任意の term order $<$ に関し, イデアル I のグレブナ基底は存在する.

[証明] モノイデアルの有限生成性より明らか. □

命題 **7.22** イデアル I の グレブナ基底 G は I を生成する.

[証明] $f \in I$ とする. 仮定により, ある $g \in G$ が存在して, $HT(g)|HT(f)$. よって, ある $s \in M$ が存在して, $HT(f - s \cdot g) < f$. $f - s \cdot g \in I$ だから, この操作を繰り返すことができる. term order の Noether 性より, この操作は有限回で終了する. すなわち, 有限回の操作の後, 0 となる. これは, G が I を生成することを意味する. □
この命題において, $f - s \cdot g$ $(f, g \in R; s \in M)$ なる演算が現れた. 命題においては, f の頭項を消去するために行なわれたが, 一般に, f の項を, このように消去する演算が, グレブナ基底計算において基本的な演算となる.

定義 **7.23** $f, g \in R$ とし, f に現れるある $\mathrm{monomial}\, m$ が $HT(g)$ で割り切れるとする. このとき f は g で 簡約可能 (**reducible**) であるという. このとき $h = f - m/HT(g) \cdot g$ に対し, $f \underset{g}{\to} h$ と書く.
$G \subset R$ についても, G のある元により f が h に簡約されるとき $f \underset{G}{\to} h$ と書く. さらに, G による簡約を 0 回以上繰り返して h が得られるとき, $f \overset{*}{\underset{G}{\to}} h$ と書く.
f のどの項も, G で簡約できないとき, f は G に関して 正規形 (**normal form**) であるという.

命題 **7.24** イデアル I の グレブナ基底 G について, 以下のことが成り立つ.

1. $f \in I \Leftrightarrow f \overset{*}{\underset{G}{\to}} 0$
2. $f \overset{*}{\underset{G}{\to}} f_1, f \overset{*}{\underset{G}{\to}} f_2$ かつ f_1, f_2 が正規形 $\Rightarrow f_1 = f_2$
3. $f \in G$ で, ある $h \in G$ が存在して $HT(h)|HT(f) \Rightarrow G \setminus \{f\}$ は I のグレブナ基底

[証明] 1. は, 前命題の証明より明らか. 3. も定義より明らか. 2. を示す. $f - f_1, f - f_2 \in I$ より $f_1 - f_2 \in I$. よって $f_1 - f_2 \overset{*}{\underset{G}{\to}} 0$. ところが, f_1, f_2 とも, G について正規形より $f_1 - f_2$ も正規形. よって, $f_1 - f_2 = 0$. □

系 **7.25** $G = \{g_1, \cdots, g_l\}$ をイデアル I の グレブナ基底とする. このとき, ある $H \subset G$ が存在して H は I の グレブナ基底かつ $HT(g_i)(g_i \in H)$ のどの二つも互いに他を割らない.

この系により, 冗長な元をすべて除去した グレブナ基底に対し, 各元を, 基底の他の元に対して正規形となるよう簡約を行ない, 頭項の係数を 1 となるようにした基底を被約 (**reduced**) グレブナ基底 と呼ぶ. これが実際に元のイデアルの グレブナ基底になっていることは, 頭項が変わっていないことよりわかる. さらに次の命題は, 定義よりただちに得られる.

命題 7.26 被約グレブナ基底は集合として一意的に定まる.

以上が グレブナ基底の定義からただちに導かれる基本的な性質であるが, 実際に グレブナ基底を構成するアルゴリズムを得るためには, グレブナ基底の定義の言いかえをいくつか行なう必要がある.

定義 7.27 $G = \{g_1, \cdots, g_l\}$ を (グレブナ基底とは限らない) R の有限部分集合とする. これに対し, 写像 d_1 を次で定義する.

$$\begin{array}{rll} d_1: & R^l & \longrightarrow R \\ & (f_1, \cdots, f_l) & \longmapsto \sum f_i \cdot HT(g_i) \end{array}$$

定義 7.28 $f = (f_1, \cdots, f_l) \in R^l$ が T-斉次 とは, ある term t が存在して, すべての i に対し, $f_i = 0$ または $t = f_i \cdot HT(g_i)$ と書けるときを言う.

定義 7.29 $e_i \in R^l$ を $e_i = (0, \cdots, 1, \cdots, 0)$ (第 i 成分のみ 1) と定義する.
$i_1, \cdots, i_k$ に対し, $T_{i_1,\cdots,i_k}$ を

$$T_{i_1,\cdots,i_k} = \mathrm{LCM}(HT(g_{i_1}), \cdots, HT(g_{i_k}))$$

と定義する. 特に, $T_i = HT(g_i)$ である.

命題 7.30 イデアル I について, 次は同値.

1. $G = \{g_1, \cdots, g_l\}$ は I のグレブナ基底
2. $f \in I \Leftrightarrow f \xrightarrow[G]{*} 0$
3. $f \in I \Leftrightarrow$ ある $f_i (i = 1, \cdots, l)$ が存在して $f = \sum_i f_i g_i$ かつ $HT(f_i g_i) \leq HT(f)$
4. L を T-斉次 な $\mathrm{Ker}(d_1)$ の基底とするとき, 任意の $h = (h_1, \cdots, h_l) \in L$ に対し, $\sum h_i \cdot g_i \xrightarrow[G]{*} 0$

[証明]
1. $\Leftrightarrow$ 2.) 明らか.
2. $\Rightarrow$ 3.) G の元による簡約操作を一つにまとめると得られる.
3. $\Rightarrow$ 2.) ある i が存在して $HT(f_i g_i) = HT(f)$ となり, $HT(f)$ が $HT(f_i)$ で割り切れることからわかる.
4. $\Rightarrow$ 1.) $f = \sum f_i \cdot g_i \in I$ とし, $HT(f)$ が $HT(g_i)$ のいずれかで割り切れることを言

えばよい. 簡単のため, すべての i に対し, $HC(g_i)=1$ とする. $L=\{b_1,\cdots,b_l\}$ とする. $m=\max_i(HT(f_ig_i))$ とおく.

$\underline{HT(f)=m \text{ の場合}}$ $HT(f)$ は, $HT(g_i)$ のいずれかで割り切れる.

$\underline{HT(f)<m \text{ の場合}}$ $A=\{i \mid HT(f_ig_i)=m\}$ とおくと, 仮定より,

$$\sum_{i\in A} HM(f_i)\cdot HT(g_i)=0.$$

よって, $h=(h_1,\cdots,h_l)\in R^l$ を, $h_i=HM(f_i)(i\in A), h_i=0(i\notin A)$ と定義すれば, $h\in \mathrm{Ker}(d_1)$. よって仮定より,

$$\text{ある } c_i\in M \text{ が存在して } h=\textstyle\sum_i c_ib_i.$$

これより

$$\sum_k h_kg_k=\sum_i c_i\sum_k g_kb_{ik}.$$

$G_i=\sum_k g_kb_{ik}$ とおくと, $G_i\xrightarrow[G]{*}0$ より, 2. $\Rightarrow$ 3. と同様に, ある b'_{ik} が存在して $G_i=\sum_k g_kb'_{ik}$ かつ $HT(g_kb'_{ik})\le HT(G_i)$. 一方, $b_i\in\mathrm{Ker}(d_1)$ より, $HT(G_i)<\max_k(HT(g_kb_{ik}))$. さて, $f'_k=\sum_i c_ib'_{ik}$ とおくと, $\sum_k h_kg_k=\sum_k f'_kg_k$ で,

$$f=\sum_{i\notin A} f_kg_k+\sum_{i\in A}(red(f)+f'_k)g_k$$

と書け,

$$HT(f'_kg_k)\le \max_i(HT(c_ib'_{ik}g_k))\le \max_i(HT(c_iG_i))$$
$$<\max_i(\max_k(HT(c_ig_kb_{ik})))=\max_k(HT(h_kg_k))=m$$

より, m がより低い順序の場合に帰着できる. よって, order の Noether 性により有限回の操作ののち, $m=HT(f)$ の場合に帰着できる. □

命題 7.31 $G=\{g_1,\cdots,g_l\}$ を $HC(g_i)=1$ なる R の有限部分集合とする. $i,j\in\{1,\cdots,l\}$ に対し, $S_{ij}\in R^l$ を $S_{ij}=T_{ij}/T_ie_i-T_{ij}/T_je_j$ で定義する. この時, $L=\{S_{ij}\mid i<j\}$ は $\mathrm{Ker}(d_1)$ の T-斉次 な基底となる. この基底を **Taylor** 基底 と呼ぶ.

[証明] $f\in\mathrm{Ker}(d_1)$ とする. f を T-斉次成分に分解することにより, f 自身 T-斉次としてよい. $f=\sum_i f_ie_i$ とすると, $f\in\mathrm{Ker}(d_1)$ より, $f\neq 0$ ならば, 少なくとも 2 つの成分が 0 でない. それらを $f_k,f_l(k<l)$ とすると, $HT(f_kg_k)=HT(f_lg_l)$ より, $T_{kl}|HT(f_kg_k)$. これより $f'=f-(HM(f_kg_k)/T_{kl})S_{kl}$ とおくと, f' は第 k 成分が 0 となり, 0 でない成分が一つ減る. この操作を繰り返して, f を $\{S_{ij}\}$ で生成できる. □

定義 7.32 $f, g \in R$ に対し, **S** 多項式 $Sp(f,g)$ を,

$$Sp(f,g) = \frac{HC(g)T_{fg}}{HT(f)} \cdot f - \frac{HC(f)T_{fg}}{HT(g)} \cdot g$$

$(T_{fg} = \mathrm{LCM}(HT(f), HT(g)))$ と定義する.

以上により, 新たな グレブナ基底の判定条件が得られる.

命題 7.33 イデアル I について, 次は同値.

1. $G = \{g_1, \cdots, g_l\}$ は I のグレブナ基底
2. 任意の対 $\{f,g\}(f,g \in G; f \neq g)$ に対し, $Sp(f,g) \xrightarrow[G]{*} 0$

7.3 Buchberger アルゴリズム

前節の最後の命題により, 次のアルゴリズムが導かれる.

アルゴリズム 7.34 (Buchberger[7])

Input : R の有限部分集合 $F = f_1, \cdots, f_l$
Output : F で生成されるイデアルの グレブナ基底 G
$D \leftarrow \{\{f,g\} \mid f, g \in F; f \neq g\}$
$G \leftarrow F$
while ($D \neq \emptyset$) do {
　　$\{f,g\} \leftarrow D$ の元
　　$D \leftarrow D \setminus \{C\}$
　　$h \leftarrow Sp(f,g)$ の正規形の一つ
　　if $h \neq 0$ then {
　　　$D \leftarrow D \cup \{\{f,h\} \mid f \in G\}$
　　　$G \leftarrow G \cup \{h\}$
　　}
}
return G

以下で, D の元を対 (pair) と呼ぶことにする.

定理 **7.35** アルゴリズム 7.34 は停止し, グレブナ基底を出力する.

[証明]
停止性　生成される正規形の頭項が, それまでに生成された正規形の頭項で割り切れないことより, 系 7.15 から言える.
出力がグレブナ基底となること　前命題により言える. □
このアルゴリズムが Buchberger アルゴリズムの最も原始的な形であるが,

- 正規形が 0 でない場合, D の要素が G の要素の個数だけ増加する.
- D から一つ元を選ぶ方法が明示されていない.

などの点で実用的でない. 実際に計算機上にインプリメントする場合, 上記二点に関して工夫をする必要がある. これらに関しては, 後に詳しく述べる.

7.4 Term order の例

様々な term order が定義できる. これらの order はそれぞれ異った性質をもち, その性質に応じてさまざまな用途に用いられる.

定義 **7.36** 辞書式順序 (**lexicographical order; LEX**)
$x_1^{i_1} \cdots x_n^{i_n} > x_1^{j_1} \cdots x_n^{j_n} \Leftrightarrow$
ある m が存在して $i_1 = j_1, \cdots, i_{m-1} = j_{m-1}, i_m > j_m$

この順序は消去法による方程式求解に最も適した形のグレブナ基底を与える. しかし, その直接計算は, 時間, 空間計算量がしばしば極めて大きくなるということから不利である.

定義 **7.37** 全次数辞書式順序 (**total degree lexicographical order; DLEX**)
$x_1^{i_1} \cdots x_n^{i_n} > x_1^{j_1} \cdots x_n^{j_n} \Leftrightarrow$
$\sum_k i_k > \sum_k j_k$ または
$(\sum_k i_k = \sum_k j_k$ かつある m が存在して $i_1 = j_1, \cdots, i_{m-1} = j_{m-1}, i_m > j_m)$

入力が斉次の場合, この順序のもとでのグレブナ基底と辞書式順序によるグレブナ基底は一致する. degree compatible order による計算は, そうでない order による計算に比べて効率がよいことが経験的に知られており, 斉次な場合にこの順序で辞書式順序グレブナ基底を計算する, あるいは非斉次な入力を斉次化して, この順序でグレブナ基底を計算し, 非斉次化することでもとの入力の辞書式順序グレブナ基底を求めるということが行われる.

定義 7.38 全次数逆辞書式順序 (total degree reverse lexicographical order; DRL)
$x_1^{i_1} \cdots x_n^{i_n} > x_1^{j_1} \cdots x_n^{j_n} \Leftrightarrow$
$\sum_k i_k > \sum_k j_k$ または
$(\sum_k i_k = \sum_k j_k$ かつある m が存在して $i_n = j_n, \cdots, i_{m+1} = j_{m+1}, i_m < j_m)$

一般に, 最も高速にグレブナ基底を計算できるが, グレブナ基底の個々の元の持つ性質は掴みにくく, 連立方程式を直接解くことは困難である. しかし, 次元, Hilbert function その他の不変量を計算する場合など, グレブナ基底という性質のみが必要とされる場合に, 高速に計算できるという特性を生かして用いられる場合が多い. また, 後に述べる基底変換の入力として用いられることも多い.

定義 7.39 block order
$\{x_1, \cdots, x_n\} = S_1 \cup \cdots \cup S_l$ (disjoint sum) とし, $<_i$ を $T_i = K[y_1, \cdots]$ $(y_k \in S_l)$ 上の term order とする. このとき, T 上の order を, $<_i$ の order を順に適用して決める.

辞書式順序は $\{x_1, \cdots, x_n\} = \{x_1\} \cup \cdots \cup \{x_n\}$ なる分割による block order であるが, 単に, 幾つかの変数を消去した結果を求めたい場合には, 効率を考えれば問題がある. このような場合に, S_1 に消去したい変数, S_2 に残りの変数, と分割し, それぞれに対し, 例えば DRL order を設定することで S_1 に属する変数を消去できる. これについては後で述べる.

定義 7.40 matrix order
M を 次を満たす実 $m \times n$ 行列とする.

1. 長さ n の整数ベクトル v に対し, $Mv = 0 \Leftrightarrow v = 0$
2. 非負成分を持つ長さ n の整数ベクトル v に対し, Mv の 0 でない最初の成分は正.

この時, $\mathbb{N}^n$ のベクトル u, v に対し,

$$u > v \Leftrightarrow M(u - v) \text{ の } 0 \text{ でない最初の成分が正}$$

で定義すれば, この order は term order となる. これを, M により定義される matrix order と呼ぶ.

命題 7.41 (Robbiano [[36]]) 任意の term order は, matrix order により定義できる.

例 7.42 よく知られた order を定義する matrix の例

$$M_{DLEX} = \begin{pmatrix} 1 & \cdots & 1 \\ 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix} M_{DRL} = \begin{pmatrix} 1 & \cdots & 1 \\ 0 & & -1 \\ & \iddots & \\ -1 & & 0 \end{pmatrix} M_{LEX} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

M_{DLEX}, M_{DRL}, M_{LEX} はそれぞれ全次数辞書式, 全次数逆辞書式, 辞書式順序を定義する.

例 7.43 weighted order

$$M_{wDRL} = \begin{pmatrix} w_1 & \cdots & w_n \\ 0 & & -1 \\ & \iddots & \\ -1 & & 0 \end{pmatrix}$$

第一行は, 指数ベクトル $(d_1, \cdots, d_n)$ に対して, $\sum_{i=1}^{n} w_i d_i$ すなわち weight 付きの全次数で最初に比較を行うことを意味する.

例 7.44 block order

$$M_{block} = \begin{pmatrix} M_1 & & 0 \\ & \ddots & \\ 0 & & M_l \end{pmatrix}$$

各 M_k は, 各ブロックに対する term order を定義する matrix である.

Chapter 8

グレブナ基底の応用

グレブナ基底は消去法以外にもさまざまな応用を持つ. 本節では, それらのいくつかについて解説する.

Notation 8.1 以下で, 次のような記法を用いる.
K : 体.
$X = \{x_1, \cdots, x_n\}$: 不定元
R : $K[X]$
T : R の項全体
$HT_<(f)$: f の $<$ に関する頭項.
$HC_<(f)$: f の $<$ に関する頭係数.
$GB_<(S)$: S の $<$ に関する被約グレブナ基底.
$NF_<(f, G)$: f の G に関する正規形の一つ. G がグレブナ基底ならば一意的に定まる.

8.1 イデアルに関する演算

命題 8.2 (イデアルの相等)
イデアル $I, J \subset R$ に関し $I = J \Leftrightarrow GB(I) = GB(J)$.

命題 8.3 (イデアルを法とする合同, メンバシップ)
イデアル I, $f, g \in R$ に関し $f \equiv g \bmod I \Leftrightarrow NF(f, GB(I)) = NF(g, GB(I))$. 特に $f \in I \Leftrightarrow NF(f, GB(I)) = 0$.

命題 8.4 (自明なイデアル)
イデアル $I \subset R$ に関し $I = R \Leftrightarrow GB(I) = \{1\}$.

命題 8.5 (elimination イデアル)
I をイデアルとする. $X = (X \backslash U) \cup U$ とし, この分割によりすべての $u \in T(U)$, すべての $x \in T(X \backslash U)$ に対し $u < x$ なる order $<$ を用いると, $GB(I \cap K[U]) = GB(I) \cap K[U]$

[証明] $f \in J = I \cap K[U]$ とする. $f \in I$ よりある $g \in GB(I)$ が存在して $HT(g) \mid HT(f)$. $HT(f) \in T(U)$ より $HT(g) \in T(U)$. ここで, $<$ の性質より, $HT(g) \in T(U)$ ならば $g \in K[U]$. よって $g \in GB(I) \cap K[U]$ で, $GB(I) \cap K[U]$ は J のグレブナ基底. □

命題 8.6 (イデアルの交わり) $I = Id(f_1, \cdots, f_l)$, $J = Id(g_1, \cdots, g_m)$ とすると, $I \cap J = (yIR[y] + (1-y)JR[y]) \cap R$

[証明] $f \in I \cap J$ とすると, $f = yf + (1-y)f \in (yI + (1-y)J) \cap R$. 逆に $f = yg + (1-y)h$ $(g \in IR[y], h \in JR[y])$ とし, $f \in R$ とする. この時, $y = 0$ を代入して, $f = h|_{y=0} \in J$. $y = 1$ を代入して, $f = g|_{y=1} \in I$. □

系 8.7 $I = Id(f_1, \cdots, f_m)$, $J = Id(g_1, \cdots, g_l)$ に対し $GB(I \cap J) = GB(\{yf_1, \cdots, yf_m, (1-y)g_1, \cdots, (1-y)g_l\}) \cap R$ により $I \cap J$ が計算できる. (左辺は $X < y$ なる elimination order で計算する.)

定義 8.8 (イデアル商) イデアル I, R の部分集合 S に対し, イデアル商 $I : S$ を

$$I : S = \{f \in R \mid fS \subset I\}$$

で定義する. $J = Id(S)$ とすれば, $I : S = I : J$ で, $J = Id(f_1, \cdots, f_m)$ ならば,

$$I : S = \bigcap_{i=1}^{m} I : Id(f_i)$$

命題 8.9 $I : Id(f) = \frac{1}{f}(I \cap Id(f))$

[証明] $g \in I : Id(f)$ ならば $gf \in I$ より $gf \in I \cap Id(f)$. よって, $g \in \frac{1}{f}(I \cap Id(f))$. 逆に, $g \in \frac{1}{f}(I \cap Id(f))$ ならば $gf \in I \cap Id(f)$ より $g \in I : Id(f)$. □

系 8.10 $I \cap Id(f)$ の生成元が求まれば, それらは f を因子に持つので, それぞれ f で割ることにより $I : Id(f)$ が求まる. 一般の場合 $I : S$ はそれらの交わりとなるが, $I \cap Id(f)$ を含めてイデアルの交わりの計算は系 8.7 により計算できるので, イデアル商も計算できることになる.

定義 8.11 (saturation)
I をイデアル, $f \in R$ とすれば, $I : f^i$ はイデアルの増大列だがら, ある $s \in \mathbb{N}$ が存在して

$$i \geq s \Rightarrow I : f^i = I : f^s$$

が成り立つ. このとき

$$I : f^\infty = I : f^s$$

と定義し, I の f に関する saturation と呼ぶ.

命題 8.12 I をイデアル, $f \in R$ に対し,

$$I : f^\infty = (IR[y] + (1 - yf)R[y]) \cap R$$

すなわち $I : f^\infty$ は elimination イデアルにより計算できる.

[証明]
右辺 $\subset$ 左辺 $g \in$ 右辺とすると, $g = ah + (1 - yf)b$ $(a, b \in R[y])$ と書ける. この式で, $y = 1/f$ とおいて, 両辺に f^d (d:十分大) を掛ければ $f^d g \in I$ すなわち $g \in I : f^d$. よって $g \in$ 左辺.
左辺 $\subset$ 右辺 $g \in$ 左辺, すなわちある d に対し $f^d g \in I$ とする. このとき

$$g \equiv (yf)^d g \equiv 0 \bmod IR[y] + (1 - yf)R[y]$$

また, $g \in R$ より $g \in$ 右辺. □

8.2 剰余環, 次元

命題 8.13 (剰余環の表現)
イデアル I に対し, 剰余環 R/I は, 正規形を元として定義される代数構造に同形である . すなわち, R/I は, 元の集合として

$$\{NF(f, GB(I)) \mid f \in R\}$$

と同一視でき, その加法 $(\oplus)$ · 乗法 $(\odot)$ として次式で定義されるものに同形となる.

$$f \oplus g = NF(f + g, GB(I))$$

$$f \odot g = NF(fg, GB(I))$$

命題 8.14 (剰余環の線形空間としての基底)
イデアル I に対し, 剰余環 R/I は K-線形空間とみなせるが, その線形空間の基底としてイデアルの グレブナ基底に対して正規形である項全体がとれる. すなわち, R/I の基底として

$$\{u \in T \mid \text{すべての } f \in GB(I) \text{ に対し } HT(f) \not| u\}$$

がとれる.

定義 8.15 I をイデアルとする. $U \subset X$ に対し, $I \cap K[U] = 0$ が成り立つとき U は independent modulo I という.

定義 8.16 (イデアルの次元)
イデアル I に対し, イデアルの次元 $\dim(I)$ を

$$\dim(I) = \max(|U| \mid U \subset X \text{ independent modulo } I)$$

で定義する.

注意 8.17 (幾何学的意味)

1. イデアル I の次元は, K の代数閉包上で考えた代数的集合 $V(I)$ の成分の最大次数に等しい.

2. より一般に, 素イデアルの減少列の長さを用いて環の次元 (Krull 次元) が定義され, 上の定義と一致することが示される.

定義 8.18 (Hilbert function)
$R = K[X]$ の s-次斉次元全体を R_s と書くことにする. イデアル I に対し, $I_s = I \cap K[X]_s$ と書く. 斉次イデアル I に対し, I の Hilbert function $H_{R/I}(s)$ を

$$H_{R/I}(s) = \dim_K R_s/I_s$$

で定義する.

命題 8.19 $<$ を任意の order とし, J を I の元の頭項で生成されるイデアルとすると, $H_{R/I}(s) = H_{R/J}(s)$

8.3 消去法

定義 8.20 イデアル I に対し, I の radical (根基) $\sqrt{I}$ を

$$\sqrt{I} = \{f \in R \mid \text{ある } e \in \mathbb{N} \text{ が存在して } f^e \in I\}$$

で定義する. $\sqrt{I}$ もイデアルとなる.

定義 8.21 L を K の拡大体とし, $V \subset K^n$ とする. このときイデアル $I(V) \subset R$ を

$$I(V) = \{f \in R \mid f|_V = 0\}$$

で定義する.

次の定理は消去法の基本となる.

定理 8.22 (Nullstellensatz; Hilbert の零点定理)
K を体, $\bar{K}$ を K の代数閉包とする. イデアル $I \subset K[X]$ に対し, $I(V_{\bar{K}}(I)) = \sqrt{I}$

系 8.23 イデアル I, J に対し, $V_{\bar{K}}(I) = V_{\bar{K}}(J) \Leftrightarrow \sqrt{I} = \sqrt{J}$

命題 8.24 (0 次元イデアルの性質)
代数閉体 K 上の多項式環のイデアル I の零点の個数が有限個 $\Leftrightarrow$ R/I が K 上有限次元の線形空間

[証明]
$\Rightarrow$) 有限個の解を $r_k = (r_{k1}, \cdots, r_{kn}) \quad (k = 1, \cdots, m)$ とする. $f_i(x_i) = \prod_k (x_i - r_{ki})$ とおくと, $f_i(x_i)$ は I の零点上で 0 となるから, Hilbert の零点定理によりある t が存在して $f_i(x_i)^t \in I$. よって, $GB(I)$ にも, 各 i に対し, $HT(g)$ が x_i の冪となるものが存在する. すると, 命題 8.14 より R/I は K 上有限次元となる.
$\Leftarrow$) 各 i に対し, 変数中で x_i が最低の順序になるような辞書式順序をとれば, $GB(I)$ 中に, $f_i(x_i)$ なる一変数多項式が存在することがわかる. よって, 解は有限個. □

変数順序として $x_1 < x_2 < \cdots < x_n$ なる辞書式順序を考えれば, イデアル I の零点が有限個の場合は, すべての i に対し, ある $f_i \in GB(I) \cap (K[x_1, \cdots, x_i] \setminus K[x_1, \cdots, x_{i-1}])$ が存在する. よって $f_1(x_1)$ から根 α_1 を求め, $f_2(\alpha_1, x_2)$ から根 α_2 を求め, という操作を繰り返せば, F の共通零点をすべて求めることができる.

8.4 加群のグレブナ基底

自由加群 $K[X]^l$ および, その部分加群 $M \subset F$ に対してもグレブナ基底が定義される. この場合, 項としては, te_i $(t \in T(X);\ e_i = (0, \cdots, 1, \cdots, 0)$: 第 i 成分のみ $1)$ をとり,

1. すべての $t \in T$, すべての F の項 m に対し $m \leq tm$
2. すべての $t \in T$, すべての F の項の組 m_1, m_2 に対し $m_1 \leq m_2 \Rightarrow tm_1 \leq tm_2$

を満たす全順序を入れる. モノイデアル $E(S)$ も同様に定義され, グレブナ基底も, $E(G)$ が $E(M)$ を生成するものとして定義される. Buchberger アルゴリズムは, S-多項式を, 頭項の F における位置が等しい (すなわち, $HT(a) = t_a e_a$, $HT(b) = t_b e_b$ のとき $a = b$) に対して通常の多項式と同様に定義し, それ以外は 0 と定義すれば全く同様にできる. 加群のグレブナ基底は, syzygy の計算を通して, 加群の自由分解 (free resolution) を与える. これにより, 加群のホモロジーの計算が可能になるが, ここでは述べない.

8.5 例 : 双対曲線の計算

elimination イデアルの応用として, 双対曲線の計算の例を示す. $f(x_1, x_2) \in \mathbb{Q}[x_1, x_2]$ とし, F の total degree を d とすれば, $F(x_0, x_1, x_2) = x_0^d f(x_1/x_0, x_2/x_0)$ は d 次同次多項式で, F の定義する代数曲線の双対曲線は,

$$\begin{cases} u_i = \frac{\partial F}{\partial x_i}(x_0, x_1, x_2)\ (i = 0, 1, 2) \\ F(x_0, x_1, x_2) = 0 \end{cases}$$

から x_0, x_1, x_2 を消去して得られる. 消去法の一つとしてグレブナ基底による消去が可能である.

$$I = Id(u_0 - \frac{\partial F}{\partial x_0}, u_1 - \frac{\partial F}{\partial x_1}, u_2 - \frac{\partial F}{\partial x_2}, F)$$

とする時, $\{x_0, x_1, x_2\} \succ \{u_0, u_1, u_2\}$ なる任意の消去順序により I のグレブナ基底 $GB(I)$ を計算すれば,

$$I \cap \mathbb{Q}[u_0, u_1, u_2] = Id(GB(I) \cap Q[u_0, u_1, u_2]).$$

以下の例で, $V(g_i)$ は $V(f_i)$ の双対曲線である.

$$\begin{cases} f_1 = x^5 - x^3 + y^2 \\ g_1 = 108x^7 - 108x^5 + 1017y^2x^4 - 16y^4x^3 - 4250y^2x^2 + 1800y^4x - 108y^6 + 3125y^2 \end{cases}$$

$$\begin{cases} f_2 = x^6 + 3y^2x^4 + (3y^4 - 4y^2)x^2 + y^6 \\ g_2 = -256x^6 + (64y^4 - 192y^2 + 864)x^4 + (-192y^4 + 1620y^2 - 729)x^2 - 256y^6 + 864y^4 - 729y^2 \end{cases}$$

$$\begin{cases} f_3 = 2x^4 - 3yx^2 + y^4 - 2y^3 + y^2 \\ g_3 = -12x^6 + (-y^2 + 178y - 37)x^4 + (12y^3 - 768y^2 + 2208y + 4608)x^2 - 32y^4 + 1024y^3 - \\ 7680y^2 - 8192y - 2048 \end{cases}$$

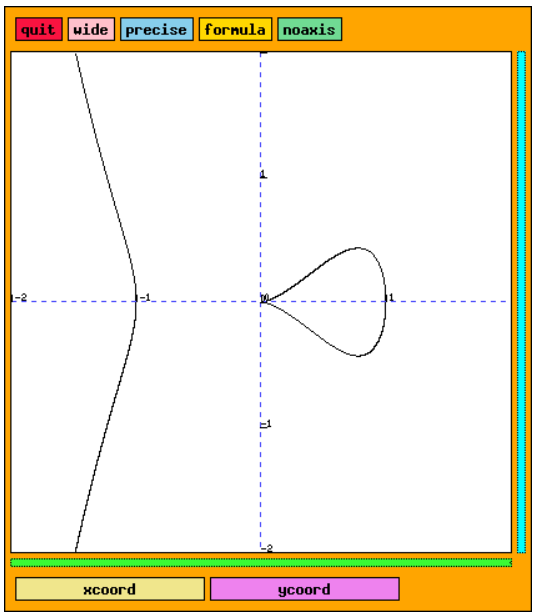

Figure 8.1: $f_1 = 0$

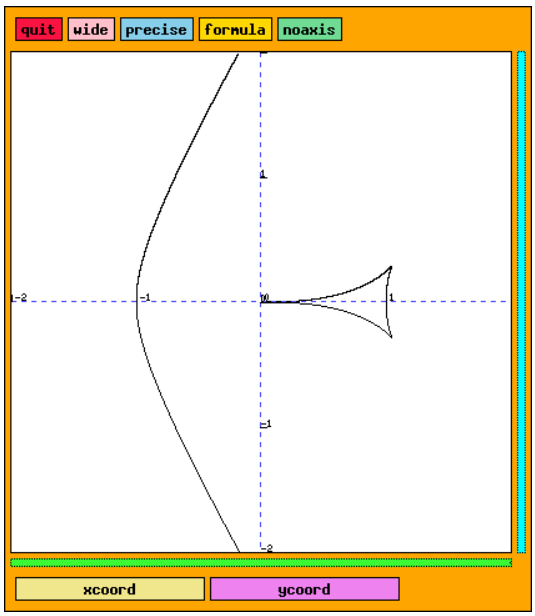

Figure 8.2: $g_1 = 0$

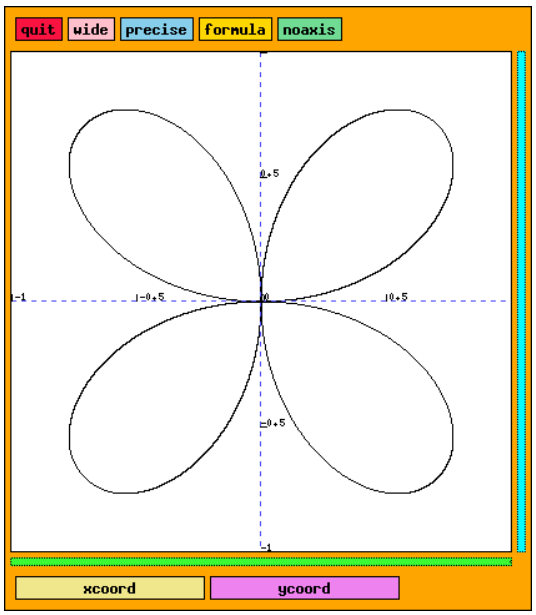

Figure 8.3: $f_2 = 0$

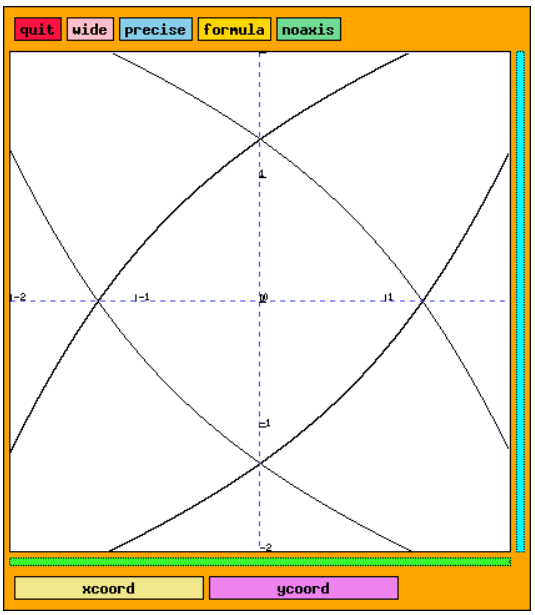

Figure 8.4: $g_2 = 0$

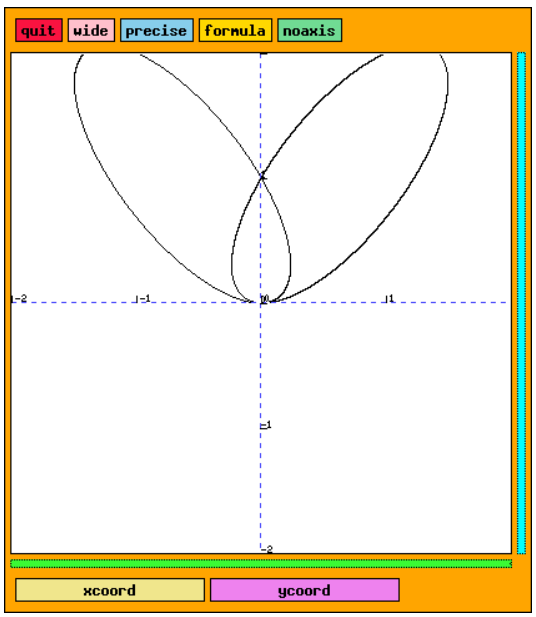

Figure 8.5: $f_3 = 0$

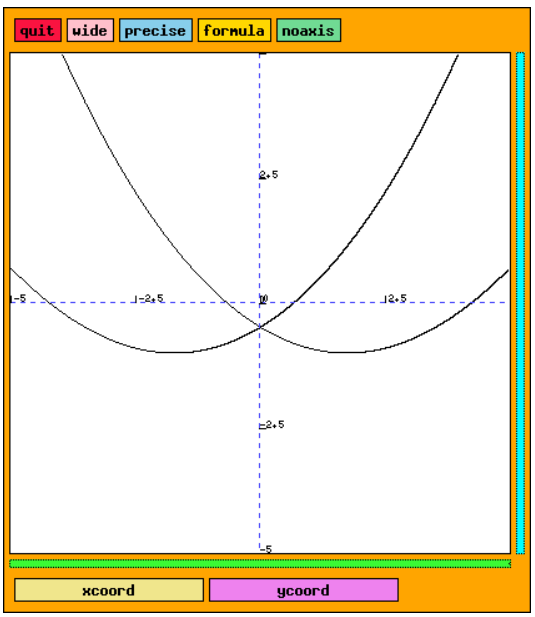

Figure 8.6: $g_3 = 0$

Chapter 9

イデアルの分解

イデアル $I \subset R = K[X]$ に対し, $I = I_1 \cap I_2$ と書ける時, $V(I) = V(I_1) \cup V(I_2)$ が成り立つ. すなわち, イデアルの分解は零点の分解を与える. より詳しく言えば, 零点の分解は radical の分解に対応する. 零点を可能な限り分解することは, 代数的集合を既約分解することに対応する. 方程式について言えば, 一般に解の分解により, より扱いやすい解の表現を与えることができる.

9.1 素イデアル, 準素イデアル, 準素イデアル分解

以下, $R = K[X]$ を固定して考える.

定義 9.1 イデアル $I \subset R$ が素イデアル (prime ideal) とは,

$$ab \in I \text{ かつ } a \notin I \text{ ならば } b \in I$$

なること. これは R/I が整域となることと同値.

定義 9.2 イデアル $I \subset R$ が準素イデアル (primary ideal) とは,

$$ab \in I \text{ かつ } a \notin I \text{ ならば } b \in \sqrt{I}$$

なること.

定義 9.3 イデアル $I \subset R$ が $I = \sqrt{I}$ を満たすとき I は radical イデアルであるという. $\sqrt{I}$ は radical イデアルである.

補題 9.4 $\sqrt{I} = \cap_{I \subset P: prime} P$

[証明] $I \subset P$ ならば $\sqrt{I} \subset \sqrt{P} = P$ より左辺 $\subset$ 右辺. 右辺が左辺を真に含むとすれば, ある $f \in \cap_{I \subset P: prime} P \setminus \sqrt{I}$ が存在する. このとき, $S = \{f, f^2, \cdots, \}$ とおけば, $S \cap \sqrt{I} = \emptyset$. $F = \{J : \text{イデアル} \mid \sqrt{I} \subset J \text{ かつ } S \cap J = \emptyset\}$ とおくと, $F \neq \emptyset$ で, 包含関係に関して帰納的. よって極大元 J_0 が存在する.
<u>主張</u> J_0 は素.
[証明] $ab \in J_0$ かつ $a, b \notin J_0$ とする. J_0 の極大性より, $S \cap (J_0 + Id(a)) \neq \emptyset$ かつ $S \cap (J_0 + Id(b)) \neq \emptyset$. すなわち, ある自然数 $s, t > 0$, $c, d \in J_0$, $e, f \in R$ が存在して, $f^s = c + ae$, $f^t = d + bf$ と書ける. すると $f^{s+t} = abef + cd + aed + cbf \in J_0$ となり矛盾. よって J_0 は素. □
この主張により, $f \notin J_0$ かつ $I \subset J_0$ なる素イデアル J_0 が存在する. これは矛盾. □

補題 9.5 イデアル $I \subset R$ が準素ならば, $\sqrt{I}$ は素イデアル.

定義 9.6 イデアル $I \subset R$ が準素で $\sqrt{I} = P$ のとき, I は P-準素という. また P を付属素イデアル (associated prime ideal) と呼ぶ.

定義 9.7 イデアル $I \subset R$ が

$$I = I_1 \cap I_2 \Rightarrow I = I_1 \text{ または } I = I_2$$

を満たすとき, I は既約という.

補題 9.8 既約イデアルは準素.

[証明] I が既約とし, $fg \in I$ かつ $f \notin I$ とする. $I : g^\infty = I : g^s$ なる s をとると,

$$I = (I + Id(g^s)) \cap (I + Id(f)).$$

これは, $h = a + bg^s = c + df$ ($a, c \in I$) なる h をとると,

$$bg^{s+1} = (c - a)g + dfg \in I \Rightarrow b \in I : g^{s+1} = I : g^s \Rightarrow bg^s \in I \Rightarrow h \in I$$

からわかる. $I \neq I + Id(f)$ だから $I = I + Id(g^s)$. すなわち $g \in \sqrt{I}$. □

定理 9.9 任意のイデアル $I \subset R$ は有限個の準素イデアルの交わりとして書ける.

[証明] I が有限個の既約イデアルの有限個の交わりで書けることをいえばよい. I が既約でなければ, $I = I_1 \cap I_2$ と, I を真に含むイデアルの交わりで書ける. I_1 が既約でなければ, I_1 は同様の交わりで書ける. もし, この操作が有限回で終らなければ, イデアルの真の無限増大列が存在することになり, R の Noether 性に反する. □

定義 9.10 イデアル I の準素分解とは, $I = \cap_{i=1}^{r} Q_i$ (Q_i:準素) なる表示のこと. 各 Q_i を準素成分と呼ぶ. 準素分解が minimal とは, 全ての $\sqrt{Q_i}$ が相異なり, $\cap_{j \neq i} Q_j \not\subset Q_i$ なること.

補題 9.11 準素イデアル I, J について, $\sqrt{I} = \sqrt{J}$ ならば $I \cap J$ も準素.

[証明] $ab \in I \cap J$ かつ $a \notin I \cap J$ とする. $a \notin I$ のとき, $b \in \sqrt{I}$, $a \notin J$ のとき $b \in \sqrt{J}$ が成り立つ. $\sqrt{I \cap J} = \sqrt{I} \cap \sqrt{J}$ より, $b \in \sqrt{I \cap J}$. □

定理 9.12 任意のイデアル $I \subset R$ は minimal な準素分解を持つ.

[証明] 補題より, 付属素イデアルが一致する準素成分は交わりをとることにより一つにまとめることができる. その後, $\cap_{j \neq i} Q_j \subset Q_i$ なる Q_i を取り除いても残りは I の準素分解となるから, これを繰り返して minimal な準素分解を得る. □
さらに, 次が成り立つ.

定理 9.13 minimal な準素分解の長さは一意的で, 付属素イデアルは集合として一致する.

定義 9.14 イデアル I の準素成分 Q の付属素イデアルが極小の時孤立 (isolated) という. そうでないとき埋没 (embedded) という.

注意 9.15 孤立, 埋没という名称は, その variety の様子による. 準素成分 Q の付属素イデアルを P とする. Q が孤立のとき, P は他の付属素イデアルを含まないから, variety でみれば, $V(P)$ は他の成分が定義する variety に含まれない. 一方, Q が埋没ならば, P はある付属素イデアル P' を含むから, variety でみれば $V(P) \subset V(P')$ すなわち埋没している.

9.2 準素分解の概略

準素分解のための主な手段は, イデアル商, extension, contraction および多項式の因数分解である.

補題 9.16 イデアル I, $f \in R \setminus I$ に対し, $I : f^m = I : f^\infty$ なる m をとれば,

$$I = I : f^\infty \cap (I + Id(f^m))$$

[証明] $h \in$ 右辺とすると, $hf^m \in I$ かつ $h = a + bf^m$ $(a \in I)$ とかける. よって $bf^{2m} = hf^m - af^m \in I$ すなわち $b \in I : f^{2m} = I : f^m$ これから $h \in I$. □

定義 9.17 (extension)
$Y \subset X$ に対し, $I^e = K(Y)[X \setminus Y]I$ と定義する.

定義 9.18 イデアル $\dim(I) = d$ とすると, 次元の定義により, $|Y| = d$ なる independent set $Y \subset X$ がとれる. これを maximally independent set とよぶ.

補題 9.19 Y が maximally independent set のとき I^e は $K(Y)$ 上 0 次元イデアル.

[証明] 定義により, 全ての $x \in X \setminus Y$ に対し, $I \cap K[\{x\} \cup Y] \neq 0$ だから, $K(Y)$ 上で考えれば x の一変数多項式が I^e 中に存在することになる. よって I^e は 0 次元.□

補題 9.20 $<$ を $Y < (X \setminus Y)$ なる block order とする. G が I の $<$ に関するグレブナ基底ならば, G は, I^e の, 同じ order に関するグレブナ基底.

定義 9.21 (contraction)
イデアル $J \subset K(Y)[X \setminus Y]$ に対し, $J \cap K[X]$ を J の contraction とよび, J^c と書く.

命題 9.22 I をイデアル, $<$ を $Y < (X \setminus Y)$ なる block order とし, G を $<$ に関する I のグレブナ基底とする. このとき,

$$f = \mathrm{LCM}\{HC(g) \mid g \in G\}$$

(ただし, $HC(g)$ は $K(Y)[X \setminus Y]$ の元としてとる) とすれば, $I^{ec} = I : f^\infty$

これらと, 後で述べる 0 次元イデアルの準素分解を用いて, 次のアルゴリズムが得られる.

アルゴリズム 9.23 [20]

Input : イデアル $I \in R = K[X]$
Output : $I = \cap Q_i$ (I の minimal な準素分解)
$P_i = \sqrt{Q_i}$ (Q_i の付属素イデアル)
$Y \leftarrow$ maximally independent set modulo I
$\cap \bar{Q}_i \leftarrow I^e$ (0 次元) の $K(Y)$ 上の準素分解
$(f, s) \leftarrow I^{ec} = I : f^\infty = I : f^s$ なる f および s
$\cap R_j \leftarrow I + Id(f^s)$ の準素分解
return ${\bar{Q}_i}^c \cap (\cap R_j)$

$I + Id(f^s)$ は I を真に含むから, 停止性は保証される. このアルゴリズムを実現するためには,

- 0 次元イデアルの準素分解
- maximally independent set の選び方

のアルゴリズムを与える必要がある. この内, maximally independent set に関しては, 次の命題がある.

命題 9.24 I をイデアルとし, $MB_<$ を, R/I の, 全次数つき order $<$ に関する monomial による K-基底とする. このとき, $\dim(I) == max(|U| \mid T(U) \subset MB_<)$.

この命題に基づいて, 一つのグレブナ基底から, $\dim(I)$ を求めるアルゴリズムが構成できる. 以下で, 0 次元イデアルの準素分解について概略を述べる.

9.3 0 次元イデアルの準素分解

$I \subset R = K[X]$ を 0 次元イデアルとする.

定義 9.25 $f \in K[X]$ が I の separating element とは, I の $\overline{K}$ 上の相異る零点 a, b に対し, $f(a) \neq f(b)$ なること.

定義 9.26 $f \in K[X]$ とする. $t \notin X$ なる不定元をとり, $R[t]$ のイデアル $J = IR[t] + Id(t - f)$ を考えれば, J も 0 次元イデアルより, $(I + Id(t - f)) \cap K[t] = Id(g(t))$ なるモニックな $g \in K[t]$ が存在する. g を f の I に関する最小多項式と呼ぶ.

命題 9.27 f を I の separating element とし, g を f の最小多項式とする. このとき,

$$g = g_1^{e_1} \cdots g_r^{e_r}$$

(g_i は K 上既約) と因数分解すれば,

$$I = (I + g_1^{e_1}) \cap \cdots \cap (I + g_r^{e_r})$$

は I の準素分解で,

$$\sqrt{I} = \sqrt{I + g_1} \cap \cdots \cap \sqrt{I + g_r}$$

は $\sqrt{I}$ の素イデアル分解となる. すなわち, $\sqrt{I + g_i}$ は $I + g_i^{e_i}$ の付属素イデアル.

イデアルの seratating element は, 直接求めるのは困難である. 定義により, $\sqrt{I}$ の separating element が I の seratating element となることを用いて, $\sqrt{I}$ を計算し, separating element を求めることを考える.

定義 9.28 体 K が完全体とは, 既約多項式が全て分離的であること.

注意 9.29 以下の命題, アルゴリズムでは, 基礎体が完全体であることを要求するものがいくつかある. 標数 0 の体は全て完全体である. また, 有限体も完全体であるが, 有限体上の有理関数体は完全体でない. 準素分解においては基礎体上の有理関数体を係数とする多項式環での計算を行うため, 基礎体の標数は 0 に限られる.

命題 9.30 K が完全体なら,

0 次元イデアル I が radical $\Leftrightarrow$ I が, 各変数について一変数無平方多項式を含む.

命題 9.31 K が完全体とすると, 0 次元 radical イデアル I の零点の個数は $\dim_K K[X]/I$ に等しい.

定義 9.32 多項式 f が $f = f_1^{e_1} \cdots f_m^{e_m}$ (f_i は無平方) と書けたとき, $f_1 \cdots f_m$ を f の無平方部分と呼ぶ.

系 9.33 $I \cap K[x_i] = Id(f_i(x_i))$ とする.

$$\sqrt{I} = I + Id(h_1, \cdots, h_n)$$

(h_i は, f_i の無平方部分)

radical イデアルに対しては, separating element の判定は次のように述べられる.

命題 9.34 K を完全体とし, イデアル I が 0 次元 radical とする. f の最小多項式を g とすると,

$$f \text{ が separating element} \Leftrightarrow \deg(f) = \dim_K R/I$$

このような f は存在する. K が無限体ならば, f として X の元の線形和から選べる.

以上が, 完全体上の 0 次元イデアルの準素分解の概略である. 直前の命題に関連して, 次のことが成り立つ.

命題 9.35 (shape lemma)
I を完全体 K 上の 0 次元 radical イデアル とし, f を separating element とする. $z << X$ なる任意の順序のもとで, $R[z]$ のイデアル $IR[z] + Id(z - f)$ は

$$\{x_1 - f_1(z), \cdots, x_n - f_n(z), z - f_z(z), m(z)\}$$

という形のグレブナ基底をもつ. この形の基底を shape basis と呼ぶ.

[証明] z の最小多項式 m は f の最小多項式に一致し, その次数は $\dim_K K[X]/I$ と等しくなる. $z << X$ なる順序のもとでは, グレブナ基底は m を含み, モノイデアルを考えれば, m 以外の元の頭項は各変数の 1 次式以外ではありえない. □
shape basis は, 0 次元イデアルの零点を数値で求めようとする場合に, 見掛け上有効な形をしている. 実際, I の零点は, $f_n(x_n)$ の零点により,

$$\{(f_1(\alpha), \cdots, f_n(\alpha)) \mid m(\alpha) = 0\}$$

と書ける. しかし, 有理数体上で実際に shape basis を求めて見ると, m の係数に比べて f_i の係数が極めて大きくなることが多い. この困難を克服するため, 次の方法が考案された.

命題 9.36 (rational univariate representation; RUR)
前命題と同じ仮定のもとで, $IR[z] + Id(z - f)$ の基底として,

$$\{m'x_1 - g_1(z), \cdots, m'x_z - g_n(z), m(z)\}$$

という形のものがとれる.

m は shape basis の場合と一致する. この基底によるの零点の表現は,

$$\{(\frac{g_1(\alpha)}{m'(\alpha)}, \cdots, \frac{g_n(\alpha)}{m'(\alpha)}) \mid m(\alpha) = 0\}$$

と書ける. 多くの実例において, g_i の各係数が, m の係数と同程度の大きさに押えられることが分かっており, 0 次元 radical の零点の表現としては RUR によるものが優れているといってよい. RUR の計算法としては, 対称式による方法が最初に提案されていが, modular change of ordering と同様の手法を適用することもでき, RUR が結果の大きさ程度で計算できる [34].

9.4 準素分解の例

次の例は, symplectic integrator と呼ばれる安定な積分スキームの数値計算法に関して現れた方程式系である [48].

$$
\begin{cases}
d_1+d_2+d_3+d_4=1, c_1+c_2+c_3+c_4=1,\\
(6d_1c_2+(6d_1+6d_2)c_3+(6d_1+6d_2+6d_3)c_4)c_1+(6d_2c_3+(6d_2+6d_3)c_4)c_2+6d_3c_4c_3=1,\\
(3d_1^2+(6d_2+6d_3+6d_4)d_1+3d_2^2+(6d_3+6d_4)d_2+3d_3^2+6d_4d_3+3d_4^2)c_1\\
+(3d_2^2+(6d_3+6d_4)d_2+3d_3^2+6d_4d_3+3d_4^2)c_2+(3d_3^2+6d_4d_3+3d_4^2)c_3+3d_4^2c_4=1,\\
(3d_1+3d_2+3d_3+3d_4)c_1^2+((6d_2+6d_3+6d_4)c_2+(6d_3+6d_4)c_3+6d_4c_4)c_1\\
+(3d_2+3d_3+3d_4)c_2^2+((6d_3+6d_4)c_3+6d_4c_4)c_2+(3d_3+3d_4)c_3^2+6d_4c_4c_3+3d_4c_4^2=1,\\
(24d_2d_1c_3+(24d_2+24d_3)d_1c_4)c_2+(24d_3d_1+24d_3d_2)c_4c_3=1,\\
(12d_2^2+(24d_3+24d_4)d_2+12d_3^2+24d_4d_3+12d_4^2)d_1c_2+((12d_3^2+24d_4d_3+12d_4^2)d_1\\
+(12d_3^2+24d_4d_3+12d_4^2)d_2)c_3+(12d_4^2d_1+12d_4^2d_2+12d_4^2d_3)c_4=1,\\
4d_1c_2^3+(12d_1c_3+12d_1c_4)c_2^2+(12d_1c_3^2+24d_1c_4c_3+12d_1c_4^2)c_2+(4d_1+4d_2)c_3^3\\
+(12d_1+12d_2)c_4c_3^2+(12d_1+12d_2)c_4^2c_3+(4d_1+4d_2+4d_3)c_4^3=1
\end{cases}
$$

準素分解により, この方程式は以下のように分解されることが分かる.

$$
\begin{cases}
24c_4^2-6c_4+1=0\\
c_1=-c_4+\frac{1}{4},\ c_2=-c_4+\frac{1}{2},\ c_3=c_4+\frac{1}{4}\ d_1=-2c_4+\frac{1}{2},\ d_2=\frac{1}{2},\ d_3=2c_4,\ d_4=0
\end{cases}
$$

$$
\begin{cases}
6c_4^3-12c_4^2+6c_4-1=0\\
c_1=0,\ c_2=c_4,\ c_3=-2c_4+1\ d_1=\frac{1}{2}c_4,\ d_2=-\frac{1}{2}c_4+\frac{1}{2},\ d_3=-\frac{1}{2}c_4+\frac{1}{2},\ d_4=\frac{1}{2}c_4
\end{cases}
$$

$$
\begin{cases}
48c_4^3-48c_4^2+12c_4-1=0\\
c_1=c_4,\ c_2=-c_4+\frac{1}{2},\ c_3=-c_4+\frac{1}{2}\ d_1=2c_4,\ d_2=-4c_4+1,\ d_3=2c_4,\ d_4=0
\end{cases}
$$

$$
\begin{cases}
6c_4^2-3c_4+1=0\\
c_1=0,\ c_2=-c_4+\frac{1}{2},\ c_3=\frac{1}{2}\ d_1=-\frac{1}{2}c_4+\frac{1}{4},\ d_2=-\frac{1}{2}c_4+\frac{1}{2},\ d_3=\frac{1}{2}c_4+\frac{1}{4},\ d_4=\frac{1}{2}c_4
\end{cases}
$$

Chapter 10

グレブナ基底計算の効率化

第 7 章でグレブナ基底を計算するアルゴリズムである Buchberger アルゴリズムの最も原始的な形を示した. しかし, そこで述べたように, アルゴリズム 7.34 をそのまま実行することは, 効率の点からみて問題がある. これは, アルゴリズムの進行につれて, 正規化して 0 になる対が増加していくためである. また, 対の選び方にも任意性がある. どのような選び方を用いても, 最終結果であるグレブナ基底の一意性は保証されているが, 選び方により効率に大きな差がでることが多くの実験により確かめられている. 本章では, グレブナ基底の計算を効率化するさまざまな方法を紹介する.

Notation 10.1 以下で, 次のような記法を用いる.

p : 素数.

$GF(p)$: p 元体.

$\mathbb{Z}_{(p)} = \{a/b \mid a \in \mathbb{Z}; b \in \mathbb{Z} \setminus p\mathbb{Z}\} \subset \mathbb{Q}$: $\mathbb{Z}$ の $p\mathbb{Z}$ における局所化.

$X = \{x_1, \cdots, x_n\}$: 不定元.

ϕ_p : $\mathbb{Z}_{(p)}[X]$ から $GF(p)[X]$ への標準的射影. $\phi_p(a/b) = \phi_p(a)/\phi_p(b)$. $(a \in \mathbb{Z}, b \in \mathbb{Z} \setminus p\mathbb{Z}.)$

$<, <_0, <_1, <_i$: term order.

$HT_<(f)$: f の $<$ に関する頭項.

$HC_<(f)$: f の $<$ に関する頭係数.

$T(f)$: f に現れる項全体の集合.

$GB_<(S)$: S の $<$ に関する被約グレブナ基底.

$f_1, \cdots, f_m$: $\mathbb{Z}[X]$ の元.

$NF_<(f, G)$: f の G に関する正規形の一つ.

$Init_<(I)$: $\{HT_<(f) \mid f \in I\}$ で生成されるイデアル.

$Useless_Pairs(D)$: 不必要対の検出.

$Select_Pair(D)$: 対の選択
$Sp(C)$: 多項式対の S-多項式の計算

10.1 不必要対の検出

Buchberger アルゴリズムにおいて, 正規化により 0 になる対を不必要対と呼ぶ. これらは, ある多項式集合が グレブナ基底であることの確認のためにのみ意味があり, もし計算せずに 0 に正規化されることがわかれば計算効率の点からみて極めて好都合である. さらに, この 「0 への正規化」は, アルゴリズムのある時点においてではなく, グレブナ基底のすべての元が生成されたのち, 0 に正規化されるのであってもよい. 実際, そのような状況は, 命題 7.30 において現れている. この命題を, Taylor 基底に適用すれば, Taylor 基底のうち, 冗長な (すなわち他の基底により生成される) 基底をとり除いた残りについて 0 に正規化されればよいことになる. ここで取り除かれる基底に対応する多項式対のほか, 他の方法により 0 に正規化されることがわかり, とり除かれる対もあり得る. これらについて述べる.

10.1.1 冗長な基底の除去

Taylor 基底から冗長な基底を取り除くための基本的な道具は, 基底間の次の自明な恒等式である.

$$\frac{T_{ijk}}{T_{ij}}S_{ij} + \frac{T_{ijk}}{T_{jk}}S_{jk} + \frac{T_{ijk}}{T_{ki}}S_{ki} = 0$$

この恒等式において, いずれかの係数が 1 であれば, その項に対応する基底は, 他の基底により生成される. これを繰り返して冗長な基底を取り除いて行くが, この際, 相互に依存し合う基底を誤って取り除くことを避けるため, 基底間に全順序を入れ, ある基底がそれより順序の低い基底により生成されている場合にのみ, その基底を取り除くこととする.

定義 10.2 $\{S_{ij} \mid i < j\}$ における全順序を次のように定義する.
$S_{ij} < S_{kl} \Leftrightarrow$
$T_{ij} < T_{kl}$ または
$(T_{ij} = T_{kl}$ かつ $(j < l$ または $(j = l$ かつ $i < k)))$

定義 10.3 対 (i, j) に関する三つの性質を次のように定義する.
$M(i,j) \Leftrightarrow$ ある $k < j$ が存在して $T_{kj} \mid T_{ij}$ かつ $T_{kj} \neq T_{ij}$
$F(i,j) \Leftrightarrow$ ある $k < i$ が存在して $T_{kj} = T_{ij}$

$B(i,j) \Leftrightarrow$ ある $k > j$ が存在して $T_k \mid T_{ij}$ かつ $T_{jk} \neq T_{ij}$ かつ $T_{ik} \neq T_{ij}$

これらは, 前記恒等式において, S_{ij} の係数が 1 かつ S_{ij} の順序が最大になる条件を場合分けにより表したものである. いずれの性質においても$T_k|T_{ij}$ より S_{ij} の係数は 1. それぞれについて確かめれば, 実際に S_{ij} が最大順序となっていることがわかる. よって, 次の命題がなり立つ.

命題 **10.4** (Gebauer and Möller[17]) Taylor 基底は, $\{S_{ij} \mid \neg M(i,j), \neg F(i,j), \neg B(i,j)\}$ で生成される.

注意 **10.5** 実際のアルゴリズムにおいては, 0 でない正規形が生成されて, D に対を追加する際に, 上記の性質を満たす対を削除していくが, 性質 M, F, B のうち, M, F は, 多項式集合 G に追加する元を含む対が対象となるのに対し, B は既に存在している対が対象となる. この事実は, 正規化対の選択戦略との関連で, アルゴリズムの進行に重大な影響を及ぼす場合がある.

10.1.2 0 に正規化される対の除去

前節で述べたもの以外に, 頭項のみにより, 0 に正規化できると判断できる対があり得る.

命題 **10.6** (Buchberger)
$\mathrm{GCD}(HT(f), HT(g)) = 1$ ならば $Sp(f,g) \xrightarrow[\{f,g\}]{*} 0$

[証明] $f = \sum_i f_i,\ g = \sum_i g_i\ (f_i, g_i \in M; f_1 > f_2 > \cdots, g_1 > g_2 > \cdots;$ ある添字以降は 0 としておく) とする. 仮定により $\mathrm{GCD}(f_1, g_1) = 1$. このとき, $Sp(f,g) = g_1 \sum_{i\geq 2} f_i - f_1 \sum_{j\geq 2} g_j$ これから, 任意の $m \geq 2$ に対してある k が存在して
$Sp(f,g) \xrightarrow[\{f,g\}]{*} R_m = (g_1 + \cdots + g_k)(f_{m-k+1} + \cdots) - (f_1 + \cdots + f_{m-k})(g_{k+1} + \cdots)$
となる. 実際, $m = 2$ については正しい. m まで正しいとする. $\mathrm{GCD}(f_1, g_1) = 1$ より, $g_1 f_{m-k+1} \neq f_1 g_{k+1}$ が言える. よって, もし $g_1 f_{m-k+1} > f_1 g_{k+1}$ ならば, $HM(R_m) = g_1 f_{m-k+1}$. すると,
$R_m \xrightarrow[f]{} R_m - g f_{m-k+1} = (g_1 + \cdots + g_k)(f_{m-k+2} + \cdots) - (f_1 + \cdots + f_{m-k+1})(g_{k+1} + \cdots)$
となり, $m+1$ の時も正しい. $g_1 f_{m-k+1} < f_1 g_{k+1}$ でも同様である. よって, この操作を繰り返して, 結局 $Sp(f,g) \xrightarrow[\{f,g\}]{*} 0$ を得る. □
この命題により, 前節の操作で残った基底の中からさらに 0 に正規化される対を除去することができる.

10.2 正規化対の選択戦略

アルゴリズム 7.34 のループにおける, 正規化の対象となる対の選択は, アルゴリズムの正当性にはなんら影響しないが, 実際の計算効率に大きく影響を与える. Buchberger により提案された戦略は次のものである.

定義 10.7 normal strategy
T_{ij} が最小な対から選択する戦略をいう.

この戦略は, 順序の低い頭項をもつ基底を早めに生成するという実用的な意味をもつ他に, この戦略によれば S 多項式は, 簡約の仕方によらず一意的な正規形となることが示されている. この戦略は, term order によっては極めて悪い結果をもたらす場合が多く, 特に, 辞書式順序のように, 全次数による比較を行わない順序で甚だしい.

定義 10.8 斉次化
t を斉次化変数とし, $R = k[x_1, \cdots, x_n]$ and $R_h = k[x_1, \cdots, x_n, t]$ とする. $f \in R$ の斉次化を f^*, $g \in R_h$ の非斉次化を g_* と書く. 以下, f の全次数を $\mathrm{tdeg}(f)$ と書く.

$$f^* = t^{\mathrm{tdeg}(f)} f(x_1/t, \cdots, x_n/t)$$

$$g_* = g|_{t=1}$$

である. 多項式集合 $F \subset R$, $G \subset R_h$ に対しても, 各元を斉次化, 非斉次化したものをそれぞれ F^*, F_* と書く.

このとき, R の order $<$ に対して, R_h の order $<_h$ が

$$\text{(H) 任意の斉次多項式 } g \in R_h \text{ に対し, } HT(g)_* = HT(g_*)$$

を満たすとき, $<_h$ を $<$ の斉次化と呼ぶことにする.

例 10.9 $X \cup \{t\}$ (X に関して $<$) による block order は $<$ の斉次化の一つであるが, 定義により,

$$ut^m <_h vt^n \Leftrightarrow \mathrm{tdeg}(ut^m) < \mathrm{tdeg}(vt^n) \text{ または } \mathrm{tdeg}(ut^m) = \mathrm{tdeg}(vt^n) \text{ かつ } u < v$$

なる $<_h$ も $<$ の斉次化である.

命題 10.10 $F \subset R$ とし, $<_h$ を $<$ の斉次化 order とする. このとき, もし G が $Id(F^*)$ の斉次多項式からなるグレブナ基底なら, G_* は $Id(F)$ のグレブナ基底となる.

斉次化は選択戦略そのものではないが, 入力を斉次化した後, 上の例で述べた全次数比較入りの $<_h$ のもとで normal strategy で計算した後非斉次化するという手順でグレブナ基底を求めると, 不必要対は増加するものの, 計算がスムーズに進行することが経験的に知られていた. 係数体が有限体の場合には, 後で述べる, 不必要対に対する正規形計算で生じる係数膨張の問題が生じないため, 有限体上でのグレブナ基底計算には斉次化は有用である.

定義 10.11 sugar strategy
グレブナ基底計算の途中に現れる多項式 f に, 次の規則によりある自然数 s_f (sugar) を対応させる.

1. 入力多項式 f に対しては, $s_f = \mathrm{tdeg}(f)$
2. m が monomial の時 $s_{mf} = \mathrm{tdeg}(m) + s_f$
3. $s_{f+g} = \max(s_f, s_g)$

sugar strategy とは, S 多項式の sugar の最も小さいものの集合から normal strategy で対を選択する戦略をいう.

Giovini ら [21] により提案されたこの戦略は, 入力多項式集合を斉次化してのグレブナ基底計算を仮想的に行なうもので, sugar は斉次化後の次数に対応している.

10.3 Trace lifting

sugar strategy により, 少なくとも係数体が有限体の場合には, 十分な効率が達成できるようになった. しかし, 係数体が有理数の場合, 前に述べた検出基準にかからない不必要対の S 多項式の正規化計算にかかるコストが大きい場合がしばしばある. そのため, 次のようなアルゴリズム (**trace lifting**) が提案された [43].

アルゴリズム 10.12
trace_lifting$(F, <)$
Input : $F \subset \mathbb{Z}[X]$; term order $<$
Output : F の $<$ に関するグレブナ基底
do {
  $p \leftarrow$ 未使用の素数
  $G \leftarrow tl_guess(F, <, p)$

if $G \neq$ **nil かつ** $\mathrm{tl_check}(F, G, <) = 1$
then return G
}

アルゴリズム 10.13

$\mathrm{tl_guess}(F, <, p)$
Input : $F \subset \mathbb{Z}[X]$; term order $<$; **素数** p
Output : F **のグレブナ基底候補または nil**
if **ある** $f \in F$ **が存在して** $\phi_p(HC(f)) = 0$ then return **nil**
$G \leftarrow F$
$G_p \leftarrow \{\phi_p(f) \mid f \in G\}$
$D \leftarrow \{\{f, g\} \mid f, g \in G; f \neq g\}$
do {
$D \leftarrow D \setminus Useless_Pairs(D)$
if $D = \emptyset$ then return G
else { $C \leftarrow Select_Pair(D)$
$D \leftarrow D \setminus \{C\}$
}
$s \leftarrow Sp(C)$
$t_p = NF(\phi_p(s), G_p)$
if $t_p \neq 0$ then {
$t \leftarrow NF(s, G)$
if $\phi_p(HC(f)) = 0$ then return **nil**
$D \leftarrow D \cup \{\{f, t\} \mid f \in G\}$
$G \leftarrow G \cup \{t\}$
$G_p \leftarrow G_p \cup \{t_p\}$
}
}

アルゴリズム 10.13 は, 有理数体上での正規形計算を行なう前に有限体上で正規形を計算し, それが 0 となる場合には, 有理数体上の正規形も 0 であると仮定してその計算を省略する. このような操作を行なっても, 生成される 0 でない多項式について, その頭項は, 通常の Buchberger アルゴリズムと同様の性質を持つためアルゴリズムは停止する. しかし, p の選び方によってはグレブナ基底を出力しない場合があるため, 次のようなテストが必要となる.

アルゴリズム 10.14

tl_check$(F, G, <)$

Input : 多項式集合 $F, G \subset \mathbb{Z}[X]\, s.t.\, G \subset Id(F)$; order $<$

Output :もし G が $<$ に関する F のグレブナ基底なら 1

そうでなければ 0

if G が $<$ に関してグレブナ基底でない then return 0

if すべての $f \in F$ に対し $NF_<(f, G) = 0$ then return 1 else return 0

実際, アルゴリズム 10.13 の出力 G に対し, $G \subset Id(F)$ は構成法より常に成り立つから, 10.14 が 1 を返せば $Id(G) = Id(F)$ で, G は $Id(F)$ のグレブナ基底となる. このテストを組み合わせたものが, アルゴリズム 10.12 である. このテストを通過できるような p としては, 通常の Buchberger アルゴリズムを実行した場合に現れるすべての多項式の頭係数を割らないような素数をとればよい. もちろんこのような素数はあらかじめ決定することはできないが, 不都合な素数は有限個であり, アルゴリズム全体としての停止性も保証される. このアルゴリズムにおいては,

不必要対に対する S 多項式の計算コスト

$\Downarrow$

有限体上でのグレブナ基底計算 + グレブナ基底テスト + 入力多項式に対するメンバシップテスト

という置き換えが行なわれている. よって, この置き換えが効率を向上させるためには, 後者が前者より高速に実行できなければならない. この比較は入力多項式集合 F の性質, あるいは用いる order の性質によって異なるが, 一般的には次のようなことが言える.

- $V(Id(F))$ の余次元が大きい場合

  グレブナ基底計算に現れる係数が長大になる傾向が大きく, 特に, $V(Id(F))$ が有限集合になる場合に, F のグレブナ基底を辞書式順序で計算する場合に甚だしい. そのような場合でも, 最終的なグレブナ基底の係数, 元の個数は小さい場合が多く, 2 つのテストも比較的容易である.

- $V(Id(F))$ の余次元が小さい場合

  多項式の項数が増大する場合が多く, そのような場合には, 後者がそのまま余分なコストにつながる.

このように, 効果が期待できない場合もあり得るが, そのような場合でも高々 2 倍程度の時間で計算は終了するため, 常に trace lifting を用いるのが安全であろう.

以上の議論においては, p としては計算機が提供する 1 ワード以内の整数を用いることを仮定している. これは, 1 ワード以内の整数に対する加減乗除が通常は機械命令として提供されていて, 高速に実行されるからである. また, 係数が法 p で一様に分布していると仮定すれば, p が大きい程, p が頭係数のどれかを割り切る確率は小さくなると期待できる. よって, 実際の計算では p として 1 ワードの範囲内でなるべく大きい整数を選ぶのがよい.

10.4 斉次化 と trace lifting の組合せ

trace lifting により, 不必要対に対する S 多項式の計算コストを, 有限体上のグレブナ基底計算および, グレブナ基底候補生成後のグレブナ基底テストと入力多項式のメンバシップテストに置き換えることができた. しかし, 実際に斉次化した場合には生じないような中間係数膨張が, sugar strategy の適用により生じる場合がしばしば見られる. 本節では, この困難を克服するための方法について述べる.
このような係数膨張を簡単な例で示す.

例 10.15 $I = Id(-3x_1 - 3x_2 - 5, -3x_1x_0^2 - 8x_0 - 6, 4x_0^2 + (-2x_1^3 - 7)x_0 - 3x_3x_1^2 - 2, -x_0^4 - x_0^2 + 3x_1x_0 - 4x_1)$

I の $x_0 > x_1 > x_2 > x_3$ なる辞書式順序のもとでのグレブナ基底 $GB(I)$ は,

$$GB(I) = \{f_3(x_3), f_2(x_2, x_3), f_1(x_1, x_3), f_0(x_0, x_3)\}$$

$$f_3 = 286978140000x_3^6 - 30735638450064x_3^5 + 139368108773718x_3^4 - 401122901874078x_3^3 + 3813285736741284x_3^2 - 1771266879937657x_3 + 25309718023001309$$

$$f_2 = -24359129516408255978429894458178435051554483918248x_2 - 91173387252004471831217734836181460114173800000x_3^5 + 94543724248166839034890914739396232954660361748 8x_3^4 - 12151997137046789022734072995370860483918763507 46x_3^3 + 92038914277032211724991748423074087630602785195 44x_3^2 - 87128026603776953223695902278763008965980027109460x_3 + 19807212871855034606976039002295220479435694630719 5$$

$$f_1 = 24359129516408255978429894458178435051554483918248x_1 - 91173387252004471831217734836181460114173800000x_3^5 + 94543724248166839034890914739396232954660361748 8x_3^4$$

$$
\begin{aligned}
&- 12151997137046789022734072995370860483918763507 46x_3^3 \\
&+ 92038914277032211724991748423074087630602785195 44x_3^2 \\
&- 87128026603776953223695902278763008965980027109460x_3 \\
&+ 23867067791256410603381021411991626321361441950427 5 \\
f_0 = &-91346735686530959919112104218169131443329314693 43x_0 \\
&- 33444249414397078848103849201574810848200000 0x_3^5 \\
&+ 37671432710327352302696037580172208903376423200x_3^4 \\
&- 35235604916430753666624787497338663840087253804 0x_3^3 \\
&+ 49996927085598535116090951898965322091760877126 2x_3^2 \\
&- 68416189150670571465095231265107250942388567714 32x_3 \\
&+ 31588951614319486540726259755101613855437086625238
\end{aligned}
$$

この計算を, 斉次化を経由して行った場合, 現れる中間基底の係数のビット長の和の最大値は 1489 ビットだが, sugar strategy のみを用いて行った場合その最大値は 10258121 ビットであった. このような小さい問題に対しても, sugar strategy がうまく働かない場合には不必要な係数膨張が生じてしまう. 斉次化は生成される基底の係数膨張が起こらないような strategy を与える場合が多いが, 非斉次の場合に比べて前に述べた不必要対の検出にかからない不必要対を多く生成し, 問題が大きくなると, それらの 0 への正規化のコストが非常に大きくなる. これらを克服する有力な方法が, 次のアルゴリズムである.

アルゴリズム 10.16

homogenized_trace_lifting$(F, <)$

Input : $F \subset \mathbb{Z}[X]$; term order $<$

Output : F の $<$ に関するグレブナ基底

do {

  $p \leftarrow$ 未使用の素数

  $G \leftarrow tl_guess(F^*, <_h, p)$

  if $G \neq$ nil

    かつ G_* が $<$ に関するグレブナ基底

    かつすべての $f \in F$ に対し $NF(f, G_*) = 0$

  then return G_*

}

$<_h$ は 例 10.9 で述べた order の全次数付斉次化を用いている. この方法と, 単に入力を斉次化したものに対して trace lifting によりグレブナ基底を求めてから非斉次化す

る方法を比較すると次のようになる. いずれも, 斉次化により選択戦略を安定にし, かつ斉次化により増加した不必要対に関するコストは trace lifting により緩和される.

- 斉次化 $\Rightarrow$ 候補生成 $\Rightarrow$ 非斉次化 $\Rightarrow$ テスト

  非斉次化を行なった時点で冗長な基底を取り除くため, テストにかかるコストは斉次化を行なわない場合と変わらない.

- 斉次化 $\Rightarrow$ 候補生成 $\Rightarrow$ テスト $\Rightarrow$ 非斉次化

  斉次なグレブナ基底候補の場合, 冗長な基底がほとんどなく, trace lifting によりカットされた大部分の不必要対を有理数体上で改めて正規化する必要がある.

すなわち, 入力が既に斉次の場合を除いて, 非斉次化後にテストを行なう方が効率がよい. この方法により, trace lifting のみでは計算不可能だった多くの問題が計算可能になった.

10.5 F_4 アルゴリズム

代数方程式求解に限らず, 代数幾何における不変量の計算などにおいても任意入力多項式集合からのグレブナ基底の計算は, 計算量的にみて dominant step となることが多い. このような場合の計算法としては Buchberger アルゴリズムがほぼ唯一の方法であったが, 最近 Faugère により F_4 (あるいは F_5) アルゴリズムが提案され, その高速性が注目されている. 本節では, このアルゴリズムについて概説する.

10.5.1 Symbolic preprocessing

アルゴリズム 7.34 を実行する場合, D からどの元を選んで来るかで, その後の計算の進行の様子が全くことなる, ということが起こる. また, 最善の元を選んだつもりでも, 特に有理数体上などにおいて, $NF(Sp(f,g),G)$ の係数が極端に大きくなり, 計算不能になることも起こる. そのため, trace lifting, homogenization などの種々のテクニックのもとで計算が試みられて来た. ここで, Buchberger アルゴリズムの核心である $NF(Sp(f,g),G)$ について考察する. $NF(Sp(f,g),G)$ は次のような手順で計算される.

$r \leftarrow af - bg$ (a, b は単項式)
while (r の項 t を割り切る $h \in G$ がある)
    $r \leftarrow r - ch$
    (r に t の項はない)

ここで, r を簡約するのに必要な $h \in G$ は一見実際に簡約操作を実行しなければ分からないように見えるが, 冗長な元 (すなわち, 実際には簡約に用いられない元) も許せば, 次のような方法で事前に得られる.

アルゴリズム 10.17 (Symbolic preprocessing)

Input : 多項式 f, 多項式集合 G
Output : $Red = \{ah \mid a : \text{単項式}, h \in G\}$
$T \leftarrow T(f)$
$Red \leftarrow \emptyset$
while $HT(g)|t$ なる $t \in T, g \in G$ が存在する do {
　　$Red \leftarrow Red \cup \{t/HT(g) \cdot g\}$
　　$T \leftarrow (T \setminus \{t\}) \cup T(reductum(t/HT(g) \cdot g)$
}
return Red

このアルゴリズムで得られた Red (Reducer) は次の性質をもつ:

- $\{f\} \cup Red$ の元のある項 t がある $g \in G$ に対し $HT(g)$ で割り切れるならば, $HT(x) = t$ なる $x \in Red$ がある.

これより, 次が言える.

- $\{f\}$ の, Red の元による K 係数の簡約操作での正規形は, G に関する正規形となる.

これを, 行列の言葉で言い換えるために次の準備をする. まず, $\{f\} \cup Red$ に現れる全ての項を T とし, T の元を順序の高い順に並べたものを $t_1 > t_2 > \cdots$ とする. T で K 上張られる線形空間の元 h の, $(t_1, t_2, \cdots)$ を基底とする行ベクトル表現を $[h]$, 行ベクトル v に対する多項式を $poly(v)$ と書くことにする.

$$[f] = [f_1 f_2 \cdots]$$

$$r_i = [r_{i1} r_{i2} \cdots] \quad (r_i \in Red)$$

とするとき,

$$A = \begin{pmatrix} f_1 & f_2 & \cdots \\ r_{11} & r_{12} & \cdots \\ r_{21} & r_{22} & \cdots \\ & \cdots & \end{pmatrix}$$

とならべ, これを, 行に関する基本変形により次の条件を満たす行列 B に変形する.

- $poly(B_i)$ が 0 でないとき, $poly(B_k)(k \neq i)$ は $HT(poly(B_i))$ を含まない.

$$B = \begin{pmatrix} 1 & 0 & ? & ? & ? & 0 & \cdots \\ 0 & 1 & ? & ? & ? & 0 & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & \cdots \\ & & & \cdots & & & \\ 0 & 0 & 0 & \cdots & 0 & 0 & \cdots \end{pmatrix}$$

このとき, $poly(B_i)$ のうち, $HT(poly(B_i))$ が $\{HT(r) \mid r \in Red\}$ に含まれないものが, $NF(f, G)$ となる.

10.5.2 F_4 アルゴリズム

前節での 正規形計算の言い替えは, 多項式剰余演算計算においてもしばしば用いられ, 特に目新しいものでもない. F_4 アルゴリズムは, 複数の S-多項式をまとめて行列形式で簡約する. そのための symbolic preprocessing も, 出発点が, S-多項式に現れる項の和集合となるだけで, 既に述べたアルゴリズムがそのまま適用される.

アルゴリズム 10.18 (Symbolic preprocessing)

Input : 多項式集合 F, 多項式集合 G
Output : $Red = \{ah \mid a : \text{単項式}, h \in G\}$
$T \leftarrow \cup_{f \in F} T(f)$
$Red \leftarrow \emptyset$
while $HT(g)|t$ なる $t \in T, g \in G$ が存在する do {
　　$Red \leftarrow Red \cup \{t/HT(g) \cdot g\}$
　　$T \leftarrow (T \setminus \{t\}) \cup T(reductum(t/HT(g) \cdot g)$
}
return Red

行列 A も, 全ての S-多項式に対応するベクトルおよび symbolic preprocessing で得られた Red に属する多項式に対応するベクトルを行とする行列として構成される. この行列を, 行基本変形により, 次の条件を満たす行列 B に変形する.

- $poly(B_i)$ が 0 でないとき, $poly(B_k)(k \neq i)$ は $HT(poly(B_i))$ を含まない.

これは, $t_1 > t_2 > \cdots$ を新たな変数とみて, この順序で reduced なグレブナ基底を計算した結果に対応する.

$$F' = \{h = poly(B_i) \mid h \neq 0, HT(h) \notin \{HT(r) \mid r \in Red\}\}$$

$$Red' = \{h = poly(B_i) \mid h \neq 0, HT(h) \in \{HT(r) \mid r \in Red\}\}$$

とおく.

命題 10.19 1. $h \in F'$ は $G \cup (F' \setminus \{h\})$ に関して正規形

2. $f \in F$ ならば $f \xrightarrow[G \cup F']{*} 0$

Proof
1: $h \in F'$ ならば $T(h) \cap \{HT(r) \mid r \in Red\} = \emptyset$. これは, Red の作り方より h が G に関して正規形であることを意味する. もちろん h は $F' \setminus \{h\}$ に関しても正規形である.
2: $f \in F$ とする.

$$NF(f, G) = f - \sum_{r_i \in Red} c_i r_i \quad (c_i \in K)$$

と書ける. よって, $NF(f, G)$ は $F \cup Red$ で K 上生成される線形空間に属する. ここで, $F \cup Red$ と $F' \cup Red'$ が K 上同一の線形空間を生成し, かつ $NF(f, G)$ には Red' の元の頭項は現れないから,

$$NF(f, G) = \sum_{f_i \in F'} d_i f_i \quad (d_i \in K)$$

と書ける. この線形和は直ちに F' に関する正規形計算となる. □
2. により F として, いくつかの S-多項式の集合をとり, F' をまとめて G に付け加えることにより, F に属する S-多項式が 0 に簡約されることが分かる. また, 1. により, アルゴリズムの停止性も言える. F として一つの S-多項式をとった場合が通常の Buchberger アルゴリズムであり, その場合には, 選び方 (selection strategy) によって効率に大きく差がでることは既に述べた. F_4 においても F の選び方が問題となるが, 複数元を選択できることで, selection strategy の効率に対する影響が少なくなることが実験により知られている.

例えば, 次のような strategy により, 多くの例が効率よく計算できる.

定義 10.20 S-多項式の sugar の最小のものを全て選ぶ.

ここで, F_4 の場合, 正規形計算において通常の sugar は意味を持たないので, ある sugar の S-多項式を集めて計算した場合, 生成された基底の sugar もその値であるとして,

再帰的に sugar の値を定めることとする. 特に, 入力多項式集合が homogeneous の場合, この strategy により, 各ステップで計算される基底は, 次数が d の場合, **reduced** なグレブナ基底のうちの, d 次の元全てとなる. これは, homogeneous ideal の場合次が成り立つことから分かる.

1. d 次の基底を生成するための S-多項式は全て $d-1$ 次以下の基底から作られる.
2. d 次の斉次多項式は, $d+1$ 次以上の基底では簡約されない.

10.5.3 Modular 計算による線形方程式の求解

F_4 においては, selection strategy に従って構成された行列を, 左基本変形するという操作の繰り返しで基底を生成していく. ここで, 左基本変形は, 線形方程式求解とみなすことができる. すなわち,

1. 行列 A から線形独立な行を全て抜きだして A' を作る.
2. A' の列を左から順に見て, それまで取り出した列と線形独立な列を抜き出す, という操作を繰り返して正方行列 A'' を作る.
3. A' で残った列を集めて B'' とする.
4. $A''X = B''$ なる線形方程式を解く.
5. X から, A' の行基本変形の結果を構成する.

元の体上で考える限り, これは単なる言い替えに過ぎないが, 例えば有理数体上の場合, $\det(A'') \neq 0$ なる A'' を modular 計算により推測して抜きだし, $A''X = B''$ の解候補を modular 計算などにより構成し,

- A の各行に, 第 5 項の結果を代入して 0 になることを確かめる.

というチェックにより modular 計算の結果が, 有理数体上での行基本変形の結果と等しいこと, 特に A を構成する多項式で張られていること, 線形方程式の解の一意性により言える. このような解候補を与えるものとして, 中国剰余定理および Hensel 構成がある. 中国剰余定理を用いる方法は, 法 p が有理数上の掃き出しと異なるものを与えない限り, 自明な方法で精度をあげていき, 整数-有理数変換の結果が stable になったら A に代入してチェックという方法である. また, Hensel 構成は アルゴリズム 11.30 そのものである. Faugère によれば, F_4 においては B が大きくなるため, B の

列数が A のサイズを越える場合には中国剰余定理を用いたほうがよいとのことである. いずれにせよ, modular 計算による方法が効率を上げる場合というのは, $det(A'')$ に比べて解の係数が小さい場合である. これは一般に期待できることではないが, 前節で述べたように, homogeneous の場合に F_4 が reduced なグレブナ基底の一部を生成する, ということから, reduced にした場合に係数が小さくなるような問題では modular 計算が効率を向上させることが期待できる.

10.5.4 例

[30] では, 4 変数の代数方程式系 (odd order 方程式) のグレブナ基底計算を Buchberger アルゴリズムにより行った. この計算に現れる中間基底において, 16 次の基底を線形形式とみて, inter reduction してみた結果, 係数が極端に小さくなることがわかる. もともと, Buchberger アルゴリズムで odd order 方程式を計算する場合, 最後に係数が大変小さい基底が何本か出て終了する. 特に, 最初に現れる, 係数の小さい基底が, 16 次の基底のうち最後のものであったため, その基底を用いて一つ前の基底を簡約したところ, 係数が小さくなった. その操作を基底全てに適用したところ, 16 次の基底全ての係数が, inter reduction により小さくできることが分かったのである. 既に述べたように, このような場合には, modular 計算による解候補の計算が有効となる. しかし, 15 次の基底を inter reduction したところ, 係数の大きさはほとんど変わらず, 大きいままであった. 以上のような背景のもとに, 現在の実装での odd order 方程式に対する Buchberger アルゴリズム計算 (homogenization+trace lifting) および F_4 の比較を示す. 表で, GaussElim, ChRem, IntRat, Check はそれぞれ Gauss 消去, 中国剰余定理, 整数-有理数変換, 結果のチェックにかかった時間を示す.

Table 10.1: 計算時間の比較

	total	GaussElim	ChRem	IntRat	Check
Buchberger	264710	—	—	—	—
F_4 homo	32880	6437	6300	5763	13340
F_4 non homo	39970	4832	6143	18240	9950

まず, total 時間が 1/8 程度に減っていることに注目したい. これは, 基底の計算時間の内訳をみればわかるように, Buchberger アルゴリズムによる計算で全体の計算時間の大部分を占めていた 16 次の計算が 1/100 程度の時間で終っているためである. これは, 最大係数のビット長が reduced な基底で非常に小さくなることに対応してい

Table 10.2: 各次数の行列のサイズ

	10	11	12	13	14	15
F_4 homo	(56,334)	(119,441)	(182,545)	(362,774)	(671,1009)	(1195,1359)
F_4 non homo	(62,339)	(128,448)	(137,461)	(227,571)	(307,621)	(955,1149)

	16	17	18	19	20	21
F_4 homo	(836,688)	(2323,1761)	(895,964)	(204,271)	(446,515)	(308,374)
F_4 non homo	(555,508)	(2022,1763)	—	—	—	—

	22	23	24	25
F_4 homo	—	—	—	—
F_4 non homo	(799,868)	(300,367)	(398,467)	(361,427)

Table 10.3: 各次数の基底の計算時間

	10	11	12	13	14	15
Buchberger	2.3	11	43	164	1214	32512
F_4 homo	1.8	10	45	357	2574	26519
F_4 non homo	2.2	12	28	166	1064	35906

	16	17	18	19	20	21	22	23	24	25
Buchberger	205234	10300	4530	—	10700	—	—	—	—	—
F_4 homo	1340	1097	359	45	208	150	—	—	—	—
F_4 non homo	937	902	—	—	—	—	341	132	186	164

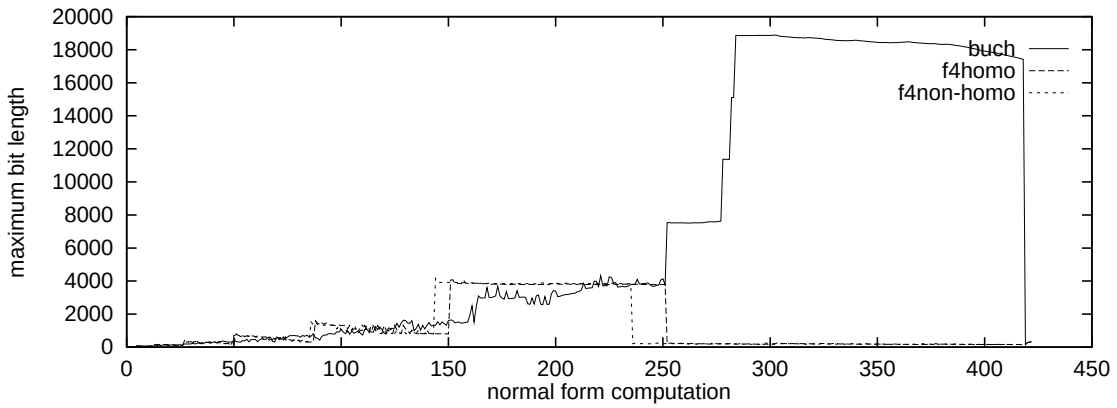

Figure 10.1: 中間基底の最大係数のビット長

る. それ以上の次数については, 小さい係数を持つ多項式による簡約化で済むことが高速化の理由である.

10.5.5 まとめ

F_4 は本質的には Buchberger アルゴリズムの改良と考えられるが, 次のような長所を持つ.

- 行列の掃き出し中は, 項の順序比較が単なる index の比較になる.
- 中間基底を reduced あるいはそれに近い形に保つため, その後の掃き出し計算が効率化する. (「対角化」された行列による簡約化 vs. 「三角化」された行列による簡約化)
- reduced な基底の係数が小さくなる場合には, modular 計算による効率化が期待できる.

一方で,

- 多数の S-多項式および reducer を一度に行列として扱うために, 巨大なメモリを必要とする場合がしばしば生ずる.
- 一般に行列は疎となるため, 大規模疎行列を効率よく保持し, 掃き出す必要がある.
- modular 計算を行う場合, 行列の各行の 0 簡約チェックによるオーバーヘッドが問題となる.

などの困難により, 実用的な実装を行うには Buchberger アルゴリズムと同様さまざまな工夫が必要となる. とはいえ, 前節の例で見たように, 実験的な実装でも従来の Buchberger アルゴリズムより効率よく計算できる場合があることは確かで, Faugère によるもの以外にもさまざまな実装実験が行われることが期待される.

Chapter 11

Change of ordering

前節では, 主として Buchberger アルゴリズムの効率化について述べた. しかし, グレブナ基底の計算法は Buchberger アルゴリズムだけとは限らない. 本節では, 既に何らかの order に関してグレブナ基底になっている多項式集合を入力として, 他の order のグレブナ基底を求める方法について述べる.

11.1 FGLM アルゴリズム

$I \subset K[X]$ を 0 次元イデアルとし, I のある order $<_1$ に関する被約グレブナ基底 G_1 が既に得られているとする. このとき, 他の order $<$ に関する I のグレブナ基底 G を, 主として線形代数により求めるのが FGLM アルゴリズムである.

補題 11.1 $<$ を admissible order, $F = GB_<(I)$ とする. $T = \{t_1, \cdots, t_l\} \subset T(X)$ を項の集合とする. a_i を未定係数とし,

$$E = \sum_{i=1}^{l} a_i NF_<(t_i, F)$$

とおく. E の X に関する係数の集合を C とすれば, $Eq = \{f = 0 \mid f \in C\}$ は a_i に関する線形方程式となる. この時

$$Eq \text{ が自明でない解を持つ} \Leftrightarrow T \text{ が } K[X]/I \text{ において } K\text{-線形従属}$$

アルゴリズム 11.2 (FGLM アルゴリズム [15])

FGLM$(F, <_1, <)$
Input : order $<_1$, $<$; $F \subset K[X]$ s.t. $F = GB_{<_1}(I)$ かつ $\dim(I) = 0$
Output : F の $<$ に関するグレブナ基底

$G \leftarrow \emptyset$

$h \leftarrow 1$

$B \leftarrow \{h\}$

$H \leftarrow \emptyset$

do {

$N \leftarrow \{u \mid u > h$ かつすべての $m \in H$ に対し $m \not| u\}$

(0) if $N = \emptyset$ then return G

(1) $h_1 \leftarrow \min(N)$

$a_t : t \in B$ に対応する未定係数

$a_{h_1} \leftarrow 1$

(2) $E \leftarrow NF_{<_1}(h_1, F) + \sum_{t \in B} a_t NF_{<_1}(t, F)$

$C \leftarrow E$ の X に関する係数の集合

if 線形方程式 $\{f = 0 \mid f \in C\}$ が解 $\{a_t = c_t \mid c_t \in K\}$ を持つ

then

$G \leftarrow G \cup \{h_1 + \sum_{t \in B} c_t t\}$

$H \leftarrow H \cup \{h_1\}$

else $B \leftarrow \{h_1\} \cup B$

$h \leftarrow h_1$

}

補題 11.3 (0) において, $B = \{u \mid u \leq h$ かつすべての $m \in H$ に対し $m \not| u\}$.

[証明] ループに関する帰納法で示す. 最初にループに入った時点では成立している. ある時点で成立しているとする. その時点からループが一回回った時点の (0) において成立することを示す. 示そうとする時点の一つ前の時点での N, B, h を N_0, B_0, h_0 と書く. $h = \min(N_0)$ で, $B = B_0 \cup \{h\}$ または $H = H_0 \cup \{h\}$ である.

$B = B_0 \cup \{h\}$ の時 $H = H_0$ より, 右辺 $= B_0 \cup \{u \mid h_0 < u \leq h$ かつすべての $m \in H_0$ に対し $m \not| u\} = B_0 \cup \{h\}$.

$H = H_0 \cup \{h\}$ の時 右辺 $= B_0 \cup (\{u \mid h_0 < u \leq h$ かつすべての $m \in H_0$ に対し $m \not| u\} \cap \{u \mid h \not| u\}) = B_0 \cup (\{h\} \cap \{u \mid h \not| u\}) = B_0$.

よって, いずれの場合にも右辺 $= B$ となる. □

補題 11.4 (1) において, $h_1 \in x_1 B \cup \cdots \cup x_n B$.

[証明] $h_1 > 1$ よりある k が存在して $h_1 = x_k h'$ と書ける. もし $h' \in N$ ならば $h_1 = \min(N)$ に反するから $h' \in B$. □

命題 11.5 アルゴリズム 11.2 は $GB_<(F)$ を出力する.

[証明] 補題 11.4 より, (1) における h の候補は 有限集合 $x_1B \cup \cdots \cup x_nB$ の元だから $\min(N)$ を与える選択アルゴリズムが存在する.
停止性 まず, (2) が解を持たないことと, $\{h_1\} \cup B$ が $K[X]/I$ で K 上一次独立であることが同値であることに注意する. よって, $|B|$ は $\dim_K K[X]/I$ を越えられない. また, H の元は, どの元も他の元を割らないから, 系 7.15 よりやはり有限集合. よってアルゴリズムは停止する.
$G = GB_<(F)$ $f \in I$ とし, $t = HT_<(F)$ とする. f は G に関して被約としてよい. $H = \{h_1, \cdots, h_m\}$ $(h_1 < \cdots < h_m)$ とする. また, (1) により選択される元を順に並べたものを $t_1 < t_2 < \cdots$ とする. もし, すべての $h \in H$ に対し $h \not| t$ ならば, 停止の条件よりある k があって, $t_k < t \leq t_{k+1}$. 仮定により, $k+1$ 番目に選ばれる元は t でなければならないから $t = t_{k+1}$. $t' \in T(f) \setminus \{t\}$ とすれば, $t' < t$ かつ全ての $h \in H$ に対し $h \not| t'$. これより $t' \leq t_k$. よって補題 11.3 より $t = t_{k+1}$ が選択されている時点で $t' \in B$. よってこの時点で f は線形方程式の解となり, $t \in H$ となるがこれは矛盾.□

FGLM アルゴリズムを計算機上で実装する場合, 特に (2) の部分の実装に工夫が必要となる. 要点をまとめると,

1. 正規形の計算は, 各項につきただ一度だけ行い, 結果は表にして保持する.
2. 毎回独立な線形方程式として解くのではなく, 結果が後で使えるような工夫をする.

1. に関連して, 次のような線形写像を考えることで, 正規形計算の効率を上げることができる.

定義 11.6 各 $i(1 \leq i \leq n)$ に対し, $\phi_i \in End(K[X]/I)$ を

$$\phi_i : f \bmod I \mapsto x_i f \bmod I$$

で定義する. $H_1 = \{HT_{<_1}(g) \mid g \in G_1\}$, $MB_1 = \{u \in T \mid$ すべての $m \in H_1$ に対し $m \not| u\}$ とおけば MB_1 は $K[X]/I$ の K-基底より, $\{NF_{<_1}(x_i u, G_1) \mid u \in MB_1\}$ を全て計算することで, ϕ_i が表現できる.

あらかじめ, ϕ_i を計算しておけば, $NF(x_i t, G_1) = \phi_i(NF(t, G_1))$ より, 既に得られているはずの $NF(t, G_1)$ の像としてあらたな項の正規形が計算できる.

注意 **11.7** 一般には, FGLM アルゴリズムは 0 次元イデアルの場合にのみ適用可能だが, 目的の order が全次数比較を含む場合など, 任意の $s \in T$ に対し $\{t \in T \mid t < s\}$ が有限集合の場合には, 任意のイデアルに適用できる. しかし, 効率は一般に 0 次元の場合に比べて期待できない.

11.2 Modular change of ordering

FGLM は, 目的の項に到達するまで行列の Gauss 消去を繰り返す方法といえる. この Gauss 消去は有理数体上で行なわれるため, 結果のグレブナ基底の元の係数に比べて途中の係数膨張が激しくなる場合がしばしば生ずる. これは, 次の例で示される.

例 **11.8** $A \in GL(n, \mathbb{Q})$ とする. $V \in \mathbb{Q}^n$ に対し $B = AV$ とすると, 線形方程式 $AX = B$ は $X = V$ を唯一の解とする. この方程式を Gauss 消去で解く場合, それは A にのみ注目して行なわれ, B の値に左右されない. すなわち, 解 V の成分が小さい整数の場合でも大きい場合でも解く手間は変わらないことになる.

11.2.1 Modular 計算と線形代数によるグレブナ基底候補生成

ここで紹介するアルゴリズムは, modular 計算を応用して, 結果の係数の大きさの程度のコストでグレブナ基底を計算するものである [33][34]. アルゴリズム 11.9 では, 有限体上のグレブナ基底計算により, 有理数体上のグレブナ基底の各元に現れる項を推測し, 未定係数法で, それらの項を実際にもつ $I = Id(F)$ の元を求める.

アルゴリズム 11.9

candidate_by_linear_algebra$(F, p, <_1, <)$

Input : order $<_1$, $<$

$F \subset \mathbb{Z}[X]$ s.t. $F = GB_{<_1}(Id(F))$

F の各元の $<_1$ に関する主係数を割らない p

Output : F の $<$ に関するグレブナ基底候補または nil

$\overline{G} \leftarrow GB_{<}(Id(\phi_p(F)))$ (被約グレブナ基底)

$G \leftarrow \emptyset$

for each $h \in \overline{G}$ do {

$a_t : t \in T(h)$ に対応する未定係数

$a_t \leftarrow 1$ ($t = ht_<(h)$ に対して)
$H \leftarrow \sum_{t \in T(h)} a_t NF_{<_1}(t, F)$
$C \leftarrow H$ の X に関する係数の集合
if 線形方程式 $E_h = \{f = 0 \mid f \in C\}$ が解 $S_h = \{a_t = c_t \mid c_t \in \mathbb{Q}\}$ を持つ
then $G \leftarrow G \cup \{d \sum_{t \in T(h)} c_t t\}$
($d : c_t$ の分母の LCM)
else return **nil**
}
return G

命題 11.10 アルゴリズム 11.9 は, 有限個の p を除いて $GB_<(F)$ を与える.

[証明] trace lifting の場合と同様に, 有限体上でのグレブナ基底の各元に現れる項が, 有理数体上でのそれと一致していれば, 全ての線形方程式は解を持ち, $G = GB_<(F)$ となる. そうでない p は有限個しかないため, それらを除いては, このアルゴリズムは $GB_<(F)$ を与える. □

注意 11.11 線形方程式が全部解けたとしても, 結果が $Id(F)$ のグレブナ基底になっている保証は現時点ではないので, この命題は不十分である. 実は, 次で述べる結果により, アルゴリズム 11.9 が **nil** でない多項式集合を返せば, それはただちに $Id(F)$ のグレブナ基底となっていることが分かる. これについては, 線形方程式の, 結果の大きさに応じた計算量を必要とする求解法とともに, 後で述べる.

11.2.2 グレブナ基底候補がグレブナ基底となる条件

ここでは, change of ordering の場合には, trace lifting の場合に必要だったグレブナ基底テストとメンバシップテストが不必要になることを示す. $F \subset \mathbb{Z}[X]$ とする.

仮定 11.12 有理数体上と有限体上の計算が異らないよう次の仮定をおく.

1. 不必要対の検出基準は頭項のみで行う.
2. 正規形計算において, 正規化に用いる元 (reducer) の選択は, 正規化される多項式の項および reducer の頭項の集合にのみ依存する.

定義 11.13 (compatibile な素数) 素数 p が F に関し compatible とは,

$$\phi_p(Id(F) \cap \mathbb{Z}[X])(= \phi_p(Id(F) \cap \mathbb{Z}_{(p)}[X])) = Id(\phi_p(F))$$

なること.

定義 11.14 素数 p が $(F,<)$ に関して strongly compatible とは p が F に関して compatible で

$$E_<(Id(F)) = E_<(Id(\phi_p(F))$$

なること.

定義 11.15 (permissible な素数) 素数 p が $(F,<)$ について permissible とは各 $f \in F$ に対し $p \not| hc_<(f)$ なること.

定義 11.16 $f \in \mathbb{Q}[X]$ が p に関し stable とは $f \in \mathbb{Z}_{(p)}[X]$ なること.

定義 11.17 (modular グレブナ基底の逆像) $G \subset Id(F) \cap \mathbb{Z}[X]$ が F の $<$ に関する p-compatible なグレブナ基底候補とは p が $(G,<)$ について permissible で $\phi_p(G)$ が $Id(\phi_p(F))$ の $<$ に関するグレブナ基底なるときをいう.

注意 11.18 compatibility は order に独立な概念である.

補題 11.19 $G \subset \mathbb{Z}[X]$, p を $(G,<)$ について permissible な素数, $f \in \mathbb{Z}[X]$ とする. このとき仮定 11.12 のもとで

$$NF(\phi_p(f), \phi_p(G)) = \phi_p(NF(f,G)).$$

[証明] $NF(f,G)$ は次の recurrence で計算される.

$$f_0 \leftarrow f, f_i \leftarrow f_{i-1} - \alpha_i t_i g_{k_i}$$

ここで, $\alpha_i \in \mathbf{Q}$, t_i : a term, $g_{k_i} \in G$. すると, p が $(G,<)$ に対して permissible より $\alpha_i \in \mathbf{Z}_{(p)}$. よって 全ての i に対し $f_i \in \mathbf{Z}_{(p)}[X]$ で, この recurrence に ϕ_p を適用して次を得る.

$$\phi_p(f_i) = \phi_p(f_{i-1}) - \phi_p(\alpha_i) t_i \phi_p(g_{k_i}).$$

もし $\phi_p(\alpha_i) \neq 0$ ならば $\phi_p(f_{i-1}) \neq 0$ で,

$$\phi_p(f_i) \leftarrow \phi_p(f_{i-1}) - \phi_p(\alpha_i) t_i \phi_p(g_{k_i})$$

は $\phi_p(G)$ による頭項消去である. もし $\phi_p(\alpha_i) = 0$ ならば $\phi_p(f_i) = \phi_p(f_{i-1})$ で

$$\{\phi_p(f_i) \mid i = 0 \text{ or } \phi_p(\alpha_i) \neq 0\}$$

なる列は $NF(\phi_p(f), \phi_p(G))$ の計算に対応する. もし $\phi_p(f_i) = 0$ なる i が存在すれば像は途中で切れるが

$$NF(\phi_p(f), \phi_p(G)) = \phi_p(NF(f, G)) = 0$$

だから主張が成立する. □

定理 11.20 $G \subset Id(F) \cap \mathbf{Z}[X]$ を $Id(F)$ の $<$ に関するグレブナ基底とする. もし p が $(G, <)$ に関して permissible かつ $\phi_p(G) \subset Id(\phi_p(F))$ ならば p は F に関して compatible である. 更に, $\phi_p(G)$ は $Id(\phi_p(F))$ のグレブナ基底で p は $(F, <)$ について strongly compatible である.

[証明] $h \in Id(F) \cap \mathbf{Z}[X]$ とする. G が $Id(F)$ のグレブナ基底だから, $NF(h, G) = 0$ より $h = \sum_{g \in G} a_g g$ と書ける. ここで $a_g \in \mathbf{Q}[X]$. 更に p が $(G, <)$ について permissible より, a_g は p について stable で, 仮定 $\phi_p(g) \in Id(\phi_d(F))$ より

$$\phi_p(h) = \sum_{g \in G} \phi_p(a_g)\phi_p(g).$$

よって $\phi_p(h) \in Id(\phi_p(F))$. 故に $\phi_p(Id(F) \cap \mathbf{Z}[X]) \subset Id(\phi_p(F))$.
逆に $\overline{h} \in Id(\phi_p(F))$ は

$$\overline{h} = \sum_{f \in F} \overline{a}_f \phi_p(f)$$

と書ける. ここで $\overline{a}_f \in GF(p)[X]$. すると, $\phi_p(a_f) = \overline{a}_f$ なる a_f を選んで $h = \sum_{f \in F} a_f f$ とおけば $\phi_p(h) = \overline{h}$. これから $\phi_p(Id(F) \cap \mathbf{Z}[X]) = Id(\phi_p(F))$.
最後に $\phi(G)$ が $Id(\phi_p(F))$ のグレブナ基底となることを示す. 上で述べたことから, $\overline{h} \in Id(\phi_p(F))$ に対し, $h \in Id(F) \cap \mathbf{Z}[X]$ が存在して $\overline{h} = \phi_p(h)$. すると, 補題 11.19 より

$$NF(\overline{h}, \phi_p(G)) = \phi_p(NF(h, G)) = 0.$$

従って, $\phi_p(G)$ は $Id(\phi_p(F))$ のグレブナ基底. strong compatibility は $E_<(Id(F))$ が $E_<(G)$ で生成されることから分かる. □
この定理で, $\phi_p(G) \subset Id(\phi_p(F))$ は $Id(\phi_p(F))$ のグレブナ基底によりチェックできる. よって, p の compatibility のチェックは有理数体上, 有限体上の任意の order でのグレブナ基底を計算することで行うことができる. もし, 入力が既にある order でのグレブナ基底なら compatibility のチェックは極めて簡単である.

系 **11.21** $G \subset \mathbf{Z}[X]$ が $<$ に関する $Id(G)$ のグレブナ基底とする. もし p が $(G,<)$ に対し permissible ならば $\phi_p(G)$ は $Id(\phi_p(G))$ のグレブナ基底で p は $(G,<)$ に対し strongly compatible.

次の定理は, グレブナ基底候補が実際にグレブナ基底になるための十分条件を与える. すなわち, 我々が求めるものである.

定理 **11.22** p が F について compatible で G が $<$ に関して p-compatible なグレブナ基底候補ならば, G は $<$ に関する $Id(F)$ のグレブナ基底である.

[証明] 全ての $f \in Id(F)$ が $<$ に関して G により 0 に正規化されることを示せばよい. f は G について被約としてよい. もし $f \neq 0$ ならば, 適当な有理数をかけて, $f \in Id(F)\backslash\{0\}$ が G について被約で f の係数の整数 GCD ($cont(f)$ と書く) が 1 に等しい, としてよい. すると $\phi_p(f) \neq 0$. さもなくば $cont(f)$ は因子 p を持つことになる. p は F につき compatible だから $\phi_p(f) \in Id(\phi_p(F))$. すると $\phi_p(f)$ は $<$ に関し, $\phi_p(G)$ により 0 に正規化されなければならない. しかし f は G について被約だから $\phi_p(G)$ の頭項の集合は G のそれと等しい. よって $\phi_p(f)$ は $\phi_p(G)$ について被約となり, $\phi_p(f) = 0$. これは矛盾. □
次の定理は前定理の精密化である. すなわち, 昇順に計算された部分的な p-compatible なグレブナ基底候補が実際にグレブナ基底の一部となっていることを保証する. これは, 途中までの結果を再利用できるという点で有用である. また, 後で述べるように, グレブナ基底のある特定の元, 例えば, 順序最小の元のみを求めたい, あるいは elimination 後の結果のみを求めたい場合にも有用である.

定理 **11.23** p が F について compatible とする. $\overline{G} \subset GF(p)[X], \overline{G} = GB_<(Id(\phi_p(F))$ とし $\overline{g}_1 < \cdots < \overline{g}_s$ なる $\overline{g}_i$ により $\overline{G} = \{\overline{g}_1, \cdots, \overline{g}_s\}$ と書く. 更に, ある正数 $t \leq s$ に対し, $g_i \in Id(F) \cap \mathbf{Z}_{(p)}[X]$ $(1 \leq i \leq t)$ が存在して $\phi_p(g_i) = \overline{g}_i$ かつ g_i は $\{g_1, \cdots, g_{i-1}\}$ について被約とする. このとき, $g_1, \cdots, g_t$ は $GB_<(Id(F))$ の最初の t 個の元に一致する.

証明略 (帰納法による.)
以上述べたことにより, 次のような一般的な change of ordering アルゴリズムが得られる.

Procedure 11.24
candidate$(F, p, <)$

Input : $F \subset Z[X]$
　　素数 p
　　order $<$
Output : F の p-compatible なグレブナ基底候補または nil
(各 F に対し, nil を返す p の個数は有限個でなければならない.)

アルゴリズム 11.25 (compatibility check によるテストの省略)

gröbner_by_change-of-ordering$(F, <)$
Input : $F \subset \mathbf{Z}[X]$, order $<$
Output : $Id(F)$ の $<$ に関するグレブナ基底 G
$G_0 \leftarrow F$ の, ある order $<_0$ に関するグレブナ基底; $G_0 \subset \mathbf{Z}[X]$
again:
　$p \leftarrow (G_0, <_0)$ に関して permissible な未使用の素数
　$G \leftarrow$ candidate(G_0,p,$<$)
　If $G =$ **nil** goto **again:**
　else return G

candidate() においては, p-compatible なグレブナ基底候補を返す任意のアルゴリズムが使用可能である. これまで述べたものでは,

- tl_guess()
- 斉次化 + tl_guess() + 非斉次化
- candidate_by_linear_algebra()

が適合する. これらのうち, 前者 2 つについては明らかだが, 最後のものについては検証を要する. これについて次節で述べる.

11.2.3 candidate_by_linear_algebra()

補題 11.26 アルゴリズム 11.9 において C に属する多項式は p について stable で, $E_{h,p} = \{\phi_p(c) = 0 \mid c \in C\}$ は一意解を持つ.

[証明] p が $(F, <_1)$ に関し permissible より $NF_{<_1}(t, F)$ は p について stable. よって $c \in C$ も p について stable.

$$S_{h,p} = \{a_t = \overline{c}_t \mid \overline{c}_t \in GF(p)\}$$

が $E_{h,p}$ の解とする.

$$\overline{h} = \sum_{t \in T(h)} \overline{c}_t t$$

とおく. すると,

$$0 = \sum_{t \in T(h)} \overline{c}_t \phi_p(NF_{<_1}(t, F)) = NF_{<_1}(\overline{h}, \phi_p(F)).$$

よって $\overline{h} \in Id(\phi_p(F))$ より $NF_<(\overline{h}, \overline{G}) = 0$. すると $\overline{G}$ が被約グレブナ基底で $T(\overline{h}) \subset T(h)$ より $\overline{h} = h$ が成り立つ. これは, 解が一意的で h に一致することを意味する. □

系 **11.27** n を不定元 a_t の個数とすると, E_h から次の性質をもつ subsystem E'_h を選ぶことができる.

- E'_h は n 個の方程式からなる.
- $\phi_p(E'_h)$ は $GF(p)$ 上で一意解をもつ.

これから次のことが分かる.

- E'_h は $\mathbf{Q}$ 上一意解を持ち, 解は p について stable.
- E_h が解をもてば, それは E'_h の一意解に一致する.

定理 **11.28** アルゴリズム 11.9 が多項式集合 G を返せば, G は F の $<$ に関して p-compatible なグレブナ基底候補である.

[証明] 各 $g \in G$ に対し

$$g = \sum_{t \in T(h)} c_t t$$

と書け, $\{c_t/c\}$ $(c = c_{hc<(g)})$ が E_h の解となるような $h \in \overline{G}$ が存在する. すると,

$$0 = c \sum_{t \in T(h)} c_t/c NF_{<_1}(t, F) = NF_{<_1}(g, F).$$

故に $g \in Id(F)$. 系 11.27 により p は $(G, <)$ について permissible で $\phi_p(g) = \phi_p(c)h$ より $\phi_p(G)$ は $\overline{G}$ のグレブナ基底である. □
上の補題を用いて E_h を次の手順で解く.

1. E'_h を選ぶ.

2. $S \leftarrow E'_h$ の一意解.

3. もし S が E_h を満たせば S は E_h の一意解, さもなくば E_h は解を持たない.

E_h は E'_h を $GF(p)$ 上で解く仮定で得られる. 以下では, E'_h を解く方法について述べる. これは次のように定式化できる.

問題 11.29 M, B をそれぞれ $n \times n$, $n \times 1$ 整数行列とし, X を, 未定係数を成分とする $n \times 1$ 行列とする. $\det(\phi_p(M)) \neq 0$ のもとで, $MX = B$ を解け.

M, B は, 一般に, 長大な整数を成分に持つ密行列となる. しかし, もともとの入力多項式の係数が小さい場合, グレブナ基底の元の係数すなわち $MX = B$ の解 X は比較的小さい場合が多い. このような場合にこの問題を Gauss 消去で解くことはこの節の最初の例で述べたように非効率的である. このような場合に有効なのが, Hensel 構成, 中国剰余定理などによる modular 計算である. ここでは Hensel 構成による方法を紹介する.

アルゴリズム 11.30

solve_linear_equation_by_hensel(M, B, p)

Input : $n \times n$ 行列 M, $n \times 1$ 行列 B

　　$\phi_p(\det(M)) \neq 0$ なる素数

Output : $MX = B$ なる $n \times 1$ matrix X

$R \leftarrow \phi_p(M)^{-1}$

$c \leftarrow B$

$x \leftarrow 0$

$q \leftarrow 1$

$count \leftarrow 0$

do {

(1)　　$t \leftarrow \phi_p^{-1}(R\phi_p(c))$

(2)　　$x \leftarrow x + qt$

　　$c \leftarrow (c - Mt)/p$

　　$q \leftarrow qp$

　　$count \leftarrow count + 1$

　　(ϕ_p^{-1} は $[-p/2, p/2]$ に正規化された逆像, $(c - Mt)/p$ は整除.)

　　if $count =$ **Predetermined_Constant** then {

　　$count \leftarrow 0$

$X \leftarrow \mathrm{inttorat}(x, q)$
if $X \neq$ **nil かつ** $MX = B$ then return X
}
}

アルゴリズム 11.31
$\mathrm{inttoat}(x, q)$
Input : $q > x$, $\mathrm{GCD}(x, q) = 1$ **なる正整数** x, q
Output : $bx \equiv a \bmod q$ **かつ** $|a|, |b| \leq \sqrt{\frac{q}{2}}$ **なる有理数** a/b **または nil**
if $x \leq \sqrt{\frac{q}{2}}$ then return x
$f_1 \leftarrow q$, $f_2 = x$, $a_1 \leftarrow 0$, $a_2 \leftarrow 1$
$i \leftarrow 1$
do {
if $|f_i| \leq \sqrt{\frac{q}{2}}$ then
if $|a_i| \leq \sqrt{\frac{q}{2}}$ then return f_i/a_i
else return **nil**
$f_i - q_i f_{i+1} < f_{i+1}$ **なる** q_i **を求める**
$f_{i+2} \leftarrow f_i - q_i f_{i+1}$
$a_{i+2} \leftarrow a_i - q_i a_{i+1}$
$i \leftarrow i + 1$
}

補題 11.32 ([45][13]) x, q **を** $q > x > 1$, $\mathrm{GCD}(x, q) = 1$ **なる正整数とする. このとき,** $|a|, |b| \leq \sqrt{q/2}$, $\mathrm{GCD}(a, b) = 1$ **なる整数** a, b $(b > 0)$ **および整数** c **があって** $ax + cq = b$ **となるならば,** (a, b, c) **は** (q, x) **に対する拡張** Euclid **互除法により得られる.**

[**略証**] $x > 1$ **より** $a \neq b$ **としてよい.** $ax + cq = b$ **より** $\frac{x}{q} + \frac{c}{a} = \frac{b}{aq}$**. ここで**

$$|\frac{b}{aq}| = |\frac{ab}{a^2 q}| < \frac{1}{2a^2}$$

より

$$|\frac{x}{q} + \frac{c}{a}| < \frac{1}{2a^2}.$$

よって, $-c/a$ **は** x/q **の主近似分数の一つ.** (**連分数の性質による.** [39, Section 24] **参照.**) $\mathrm{GCD}(a, b) = 1$ **より** $\mathrm{GCD}(a, c) = 1$ **であり,** $b > 0$ **に対して** (a, c) **は一意的に決まる. よって,** (a, b, c) **は** (q, x) **に対する拡張** Euclid **互除法の係数として得られる.** □

補題 11.33 (1) において $Mx \equiv B \bmod q$ かつ $c = (B - Mx)/q$.

[証明] 帰納法により示す. $count = 0$ のとき明らか. $count = m$ まで言えたとする. このとき, (2) で $Mt \equiv c \bmod p$ より $M(x+qt) \equiv Mx + q(B - Mx)/q \bmod qp$. すなわち $M(x+qt) \equiv B \bmod qp$. また, $(c - Mt)/p = ((B - Mx) - qMt)/qp = (B - M(x+qt))/qp$ より $count = m + 1$ でも言える. □

命題 11.34 アルゴリズム 11.30 は $MX = B$ の解 X を出力する.

[証明] 補題 11.33 より, $count = m$ のとき (1) において $Mx \equiv B \bmod p^m$. $\det(M) \neq 0$ より x は法 p^m で一意的である. 一方 $MX = B$ は有理数体上一意的に解を持つ. この解を $X = N/D$ (D は整数, N は整数ベクトル) と書くとき, $Dr \equiv 1 \bmod p^m$ なる整数 r をとれば $M(rN) \equiv B \bmod p^n$. よって $rN \equiv x \bmod p^m$. D, N の各成分が A を越えないとき, $p^m > 2A^2$ なる m をとれば, 補題 11.32 より rN すなわち x からアルゴリズム 11.31 により N/D が復元される. □

Predetermined_Constant はある正整数で, 有理数に引き戻してチェックを行う頻度を制御する. アルゴリズム 11.30 において c は $nMAX(\|M\|_\infty, \|B\|_\infty)$ で押えられることがわかる. これは, 各ステップがある constant 時間内で計算できることを示す. また, 解の分母分子が A で押えられていれば, $q > 2A^2$ になった段階で解を復元できる. これは, 解の大きさに応じた手間で計算できることを意味する. これに対して, 係数膨張を押えた Gauss 消去法である fraction-free 法によっても, 解の大きさにかかわらず, 係数行列の行列式を計算してしまうという点で, Gauss 消去法は, この問題を解くためには不適切である.

11.3 タイミングデータ

本節では, 以下のようなベンチマーク問題に関し, 本章で述べた様々な change of ordering アルゴリズムの効率の比較を示す. また, 命題 9.36 で触れた RUR のモジュラ計算を同じ問題に適用し, その優位性を実験結果により示す. 計測は, PC (FreeBSD, 300MHz Pentium II, 512MB of memory) で行った. 単位は秒. garbage collection 時間は除いてある.

$C(n)$ The cyclic n-roots system of n variables. (Faugere *et al.*,1993).
$\{f_1, \cdots, f_n\}$ where $f_k = \sum_{i=1}^{n} \prod_{j=i}^{k+j-1} c_{j \bmod n} - \delta_{k,n}$. ($\delta$ is the Kronecker symbol.)
The variables and ordering : $c_n \succ c_{n-1} \succ \cdots \succ c_1$

$K(n)$ The Katsura system of n+1 variables.

$\{u_l - \sum_{i=-n}^{n} u_i u_{l-i} (l = 0, \cdots, n-1), \sum_{l=-n}^{n} u_l - 1\}$

The variables and ordering : $u_0 \succ u_1 \succ \cdots \succ u_n$.

Conditions : $u_{-l} = u_l$ and $u_l = 0 (|l| > n)$.

$R(n)$ `e7` in Rouillier (1996).

$\{-1/2 + \sum_{i=1}^{n} (-1)^{i+1} x_i^k (k = 2, \cdots, n+1)\}$

The variables and ordering : $x_n \succ x_{n-1} \succ \cdots \succ x_1$.

$D(3)$ `e8` in Rouillier (1996).

$\{f_0, f_1, f_2, \cdots, f_7\}$

$f_0 = -420y^2 - 280zy - 168uy - 140vy - 120sy - 210ty - 105ay + 12600y - 13440$

$f_1 = -840zy - 630z^2 - 420uz - 360vz - 315sz - 504tz - 280az + 18900z - 20160$

$f_2 = -630ty - 504tz - 360tu - 315tv - 280ts - 420t^2 - 252at + 12600t - 13440$

$f_3 = -5544uy - 4620uz - 3465u^2 - 3080vu - 2772su - 3960tu - 2520au + 103950u - 110880$

$f_4 = -4620vy - 3960vz - 3080vu - 2772v^2 - 2520sv - 3465tv - 2310av + 83160v - 88704$

$f_5 = -51480sy - 45045sz - 36036su - 32760sv - 30030s^2 - 40040ts - 27720as + 900900s - 960960$

$f_6 = -45045ay - 40040az - 32760au - 30030av - 27720as - 36036at - 25740a^2 + 772200a - 823680$

$f_7 = -40040by - 36036bz - 30030bu - 27720bv - 25740bs - 32760bt - 24024ba + 675675b - 720720$

The variables and ordering : $b \succ a \succ s \succ v \succ u \succ t \succ z \succ y$.

$Rose$ The Rose system.

$O_1 : u_3 \succ u_4 \succ a_{46}$, $O_2 : u_3 \succ a_{46} \succ u_4$.

Liu The Liu system.

$\{y(z-t) - x + a, z(t-x) - y + a, t(x-y) - z + a, x(y-z) - t + a\}$

The variables and ordering : $x \succ y \succ z \succ t \succ a$.

$Fate$ The Fateman system, appeared on NetNews.

$\{s^3 + 2r^3 + 2q^3 + 2p^3,\ s^5 + 2r^5 + 2q^5 + 2p^5,$

$-s^5 + (r+q+p)s^4 + (r^2 + (2q+2p)r + q^2 + 2pq + p^2)s^3 + (r^3 + q^3 + p^3)s^2$

$+(3r^4 + (2q+2p)r^3 + (4q^3 + 4p^3)r + 3q^4 + 2pq^3 + 4p^3q + 3p^4)s + (4q+4p)r^4$

$+(2q^2 + 4pq + 2p^2)r^3 + (4q^3 + 4p^3)r^2 + (6q^4 + 4pq^3 + 8p^3q + 6p^4)r$

$+4pq^4 + 2p^2q^3 + 4p^3q^2 + 6p^4q\}$

The variables and ordering : $p \succ q \succ r \succ s$.

$hC(6)$ A homogenization of C(6).

$(C_6 \backslash \{c_1c_2c_3c_4c_5c_6 - 1\}) \cup \{c_1c_2c_3c_4c_5c_6 - t^6\}$

The variables and ordering : $c_1 \succ c_2 \succ c_3 \succ c_4 \succ c_5 \succ c_6 \succ t$.

11.3.1 Change of ordering

予め計算してある DRL (全次数逆辞書式順序) グレブナ基底から出発して, LEX (辞書式順序) グレブナ基底を計算する. 用いるアルゴリズムは, TL (tl_guess(); アルゴリズム 10.13), HTL (斉次化+tl_guess()+非斉化), LA (candidate_by_linear_algebra(); アルゴリズム 11.9 (0 次元システムのみ)) である. 表 11.1 は DRL から LEX への変換にかかる時間をしめす. *DRL* は, DRL の計算時間を示す. グレブナ基底チェックを省く効果を示すために, tl_ckeck() (アルゴリズム 10.14) の時間も示す.

Table 11.1: Modular change of ordering

	$K(5)$	$K(6)$	$K(7)$	$C(6)$	$C(7)$	$R(5)$	$R(6)$
DRL	0.84	8.4	74	3.1	1616	11	1775
TL	∞	∞	∞	∞	∞	∞	∞
HTL	16	1402	1.6×10^5	5.6	2×10^4	383	2.1×10^5
LA	4.7	158	6813	4	435	9.5	258
tl_check	2.3	177	1.3×10^4	1.1	2172	3	40

	$D(3)$	$RoseO_1$	$RoseO_2$	Liu	$Fate$	$hC(6)$
DRL	30	0.19	0.15	0.06	0.5	7.2
TL	∞	1.7	354	∞	4	25
HTL	4.1×10^4	1.7	36	18	4	25
LA	585	3.3	12	—	—	—
tl_check	575	0.6	13	17	26	24

整数係数多項式に対し, その maginitude を, 係数のビット長の和で定義する. *TL* と *HTL* の差を見るために, 表 11.2 で, 計算途中における最大 magnitude を示す.

Table 11.2: Maximal magnitude

	$C(6)$	$K(5)$	$K(6)$	$RoseO_1$	$RoseO_2$	Liu
TL	> 735380	> 2407737	> 57368231	69764	947321	> 327330
HTL	1992	44187	422732	37220	70018	21095

表より明らかに, *TL* は非斉次多項式に対するグレブナ基底計算に不向きであることがわかる. さらに, 表 11.1 は *HTL* に対する *LA* の優位性を示している. これは, Buchberger アルゴリズムが Euclid の互除法に対応していて, 中間係数膨張で効率が

左右されるのに対し, modular アルゴリズムの効率は結果の大きさのみに依存することによる.

11.3.2 RUR

RUR の modular 計算のタイミングデータを示す. 計算環境は前節と同様である. ここでは, 予め modular 計算により separating element を求めてある. これらを用いて, それぞれ次のような多項式を添加したイデアルに対し, w に関する RUR 計算を行う.

$C(6)$ $\quad w-(c_1+3c_2+9c_3+27c_4+81c_5+243c_6)$

$C(7)$ $\quad w-(c_1+3c_2+9c_3+27c_4+81c_5+243c_6+729c_7)$

$K(n)$ $\quad w-u_n$

$R(5)$ $\quad w-(x_1-3x_2-2x_3+3x_4+2x_5)$

$R(6)$ $\quad w-(x_1-3x_2-2x_3+3x_4+2x_5-4x_6)$

$D(3)$ $\quad w-y$

Table 11.3: 計算時間 (秒)

	$K(6)$	$K(7)$	$K(8)$	$C(6)$	$C(7)$	$R(5)$	$R(6)$	$D(3)$
Total	7.4	69	1209	4.6	1643	52	8768	67
Quick test	0.4	3.2	26	0.5	57	6.5	384	3.1
Normal form	1.1	12	308	1.4	762	15	2861	7.3
Linear equation	4.1	43	775	1.4	641	22	3841	45
Garbage collection	1.7	10	100	1.2	181	7.8	1681	11

表で, Quick Test は modular 計算で w が separating element となることをチェックする時間, Normal Form は, 線形方程式を生成するための, monomial の正規形の計算, Linear Equation は, 線形方程式求解の時間である. 表 11.1 と比較すれば, ある変数が separating element となっているような問題では, RUR 計算の効率が非常によいことが分かる. この理由は, RUR に現れる係数の bit 長が, LEX 基底のそれに比べて非常に小さい (例えば $K(7)$ では 60 分の 1 程度) ことと, 線形方程式の求解が, 結果の大きさに応じた手間でできることから分かる.

Bibliography

[1] Abbott, J.A. et al, Factorisation of Polynomials: Old Ideas and Recent Results. Trends in Computer Algebra (LNCS 296), 81-91.

[2] Adams, W., Loustaunau, P., An Introduction to Gröbner Bases. Graduate Strudies in Mathematics, Vol. 3., AMS (1994).

[3] Alonso, M. E., Becker, E., Roy, M. F., Wörmann, T., Zeros, Multiplicities and Idempotents for Zerodimensinal Systems. Proc. MEGA94, Birkhäuser (1996).

[4] Becker, T., Weispfenning, V., Gröbner Bases. GTM 141, Springer-Verlag(1993).

[5] Berlekamp, E.R., Factoring Polynomials over Large Finite Fields. Math. Comp. 24 (1970), 713-735.

[6] Boehm, H.,Weiser, M., Garbage Collection in an Uncooperative Environment. Software Practice & Experience(September 1988), 807-820.

`http://reality.sgi.com/boehm_mti/gc.html`

[7] Buchberger, B., Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Geichungssystems. Aequ. Math. 4/3 (1970), 374-383.

[8] Cantor, D.G., Zassenhaus, H., On Algorithms for Factoring Polynomials over Finite Fields. Math. Comp. 36 (1981), 587-592.

[9] Collart, M. et al, Converting Bases with the Gröbner Walk. J. Symb. Comp. 24/3/4(1997), 465-469.

[10] Cox, D., Little, J., O'Shea, D., Ideals, Varieties, and Algorithms. UTM, Springer-Verlag (1992).

[11] Cox, D., Little, J., O'Shea, D., Using Algebraic Geometry. GTM 185, Springer-Verlag (1998).

[12] Davenport, J.H. et al, Computer Algebra. Academic Press (1988).

[13] Dixon, J. D., Exact Solution of Linear Equations Using P-Adic Expansions. Numerische Mathematik **40** (1982), 137-141.

[14] Eisenbud, D., Commutative Algebra with a View Toward Algebraic Geometry. GTM 150, Springer-Verlag (1995).

[15] Faugère, J.C., Gianni, P., Lazard, D., Mora, T., Efficient Computation of Zero-dimensional Gröbnerr Bases by Change of Ordering. J. Symb. Comp. 16/4(1993), 329-344.

[16] Faugère, J.C., A new efficient algorithm for computing Groebner bases (F_4), Journal of Pure and Applied Algebra (139) 1-3 (1999), 61-88.

[17] Gebauer, R., Möller, H.M., On an installation of Buchberger's algorithm. J. Symb. Comp. 6/2/3(1989), 275-286.

[18] Geddes, K.O. et al., Algorithms for Computer Algebra. Kluwer Academic Publishers, Boston (1992).

[19] Gel'fond, A.O., Transcendental and Algebraic Numbers. New York, Dover (1960).

[20] Gianni, P., Trager, B., Zacharias, G., Gröbner bases and primary decomposition of polynomial ideals. J. Symb. Comp. 6/2,3(1988), 149-167.

[21] Giovini, A., Mora, T., Nielsi, G., Robbiano, L., Traverso, C., "One sugar cube, please" OR Selection strategies in the Buchberger algorithm. Proc. ISSAC '91, 49-54.

[22] van Hoeij, M., Factoring polynomials and the knapsack problem. To appear in Journal of Number Theory. The preprint is available from `http://euclid.math.fsu.edu/~hoeij/papers.html`.

[23] カーニハン, B.W., リッチー, D.M., プログラミング言語 C 第 2 版. 共立出版 (1989).

[24] Knuth, D.E., The Art of Computer Programming, Vol. 2. Seminumerical Algorithms, 2nd ed. Addison-Wesley (1981).

[25] Lenstra, A.K., Lenstra, H.W., Lobász, Factoring polynomials with rational coefficients, Math, Ann. 261 (1982), 515-534.

[26] Loos, R., Generalized Polynomial Remainder Sequences. Computing, Suppl. 4 (1982), 115-137.

[27] Mignotte, M., Mathematics for Computer Algebra. Springer-Verlag (1982).

[28] Moses, J., Yun, D.Y.Y., The EZ GCD Algorithm, Proc. ACM Annual Conf. (1973), 159-166.

[29] 永尾汎, 代数学. 新数学講座 4, 朝倉書店 (1983).

[30] Noro, M., J. McKay, Computation of replicable functions on Risa/Asir. Proc. PASCO'97, ACM Press (1997), 130-138.

[31] Noro, M. et al, Asir.

`ftp://archives.cs.ehime-u.ac.jp/pub/asir2000`

[32] Noro, M., Takeshima, T., Risa/Asir — A Computer Algebra system. Proc. ISSAC'92, ACM Press(1992), 387-396.

[33] Noro, M., Yokoyama, K., New methods for the change-of-ordering in Gröbner basis computation. Research Report ISIS-RR-95-8E, FUJITSU LABS, ISIS (1995).

[34] Noro, M., Yokoyama, K., A Modular Method to Compute the Rational Univariate Representation of Zero-Dimensional Ideals. J. Symb. Comp. **28**/1 (1999), 243-263.

[35] 大阿久俊則, グレブナ基底と線型偏微分方程式系 (計算代数解析入門). 上智大学数学講究録 No. 38 (1994).

[36] Robbiano, L., Term orderings on the polynomial ring. Proc. EUROCAL'85 (LNCS 204), 513-517.

[37] 佐々木建昭, 数式処理. 情報処理叢書 7, 情報処理学会 (1981).

[38] Shimoyama, T., Yokoyaka, K., Localization and Primary Decomposition of Polynomial ideals. J. Symb. Comp. 22(1996), 247-277.

[39] 高木貞治, 初等整数論講義 第 2 版. 共立出版 (1971).

[40] 高木貞治, 代数学講義 改訂新版. 共立出版 (1965).

[41] Torbjörn et al, GNU MP library. Free Software Foundation (1996).

`http://www.fsf.org/software/gmp/gmp.html`

[42] Trager, B.M., Algebraic Factoring and Rational Function Integration. Proc. SYMSAC 76, 219-226.

[43] Traverso, C., Gröbner trace algorithms. Proc. ISSAC '88 (LNCS 358), 125-138.

[44] Wang, P.S., An Improved Multivariate Polynomial Factoring Algorithm. Math. Comp. 32(1978), 1215-1231.

[45] Wang, P.S. et al, p-adic Reconstruction of Rational Numbers. SIGSAM Bulletin 16(1982), 2-3.

[46] Wang, P.S., Trager, B.M., New Algorithms for Polynomial Square-Free Decomposition over the Integers. SIAM J. Comp. 8(1979), 300-305.

[47] Weber, K., The Accelerated Integer GCD Algorithm. ACM Transactions on Mathematic al Software, Vol. 21, No. 1 (1995), 111-122.

[48] 吉田春夫, シンプレクティック数値解法. 数理科学, 33, 6 (1995), 27-46.

[49] Zassenhaus, H., On Hensel Factorization I. J. Number Theory 1 (1969), 291-311.

本講では, 予備知識として, 線形代数および代数学の基礎的な内容を仮定している. 後者に関しては, 具体的には群, 環, 体などの代数系に関する基礎的事項, 特に体の拡大およびイデアルに関する知識を持っていることが望ましい. 例えば [29] などを薦める. [24] は, 数, 多項式に関するさまざまなアルゴリズムが詳細かつ正確に解説されていて, 辞書的に使える. [12][18][37] は計算機代数の教科書. [2][4][10][11][14][35] はグレブナ基底あるいは可換代数に関する教科書. [23] は C 言語の標準的な解説書である.

以下, 各章に関し, 参考書, 論文を挙げながら簡単に振り返る.

第 2 章:

ここで述べたのは, CPU および C 言語に関する最低限のことがらである. CPU に関しては, 文字通り CPU 毎にマニュアルが存在するが, 興味のある方は, 例えば Pentium などのマニュアルを眺めてみてはどうだろうか. C 言語に関しては数え切れない程の解説書があるが, 筆者は [23] の他には, 何らかのフリーソフトウェアのソースコードを読むことをお勧めする. 特に, gmp [41] は第 3 章で述べたアルゴリズムおよびより進んだアルゴリズムが実装されており, 参考になると思う.

第 3 章:

ここで最も理解しにくいのは除算アルゴリズムであろう. 各補題の証明は省略してあるので, 練習問題として解いてみれば除算アルゴリズムの仕組みがよく分かると思う. これらは全て [24] に証明がある. 整数 GCD についてなにも述べなかったが, Euclid 互除法の他に binary GCD アルゴリズムとよばれるタイプのアルゴリズムがあり, 倍精度整数除算を用いないため高速に実行できる. これに関してはやはり [24] に解説されている. また, 最近このタイプのアルゴリズムをさらに進化させたものが提案されている. これについては [47] 参照.

第 4 章:

多項式を C などの高級言語で表現し, その演算を記述してみることは, プログラミングのよい練習になる. その際に問題となるのが, 不要になったメモリ領域の開放に関することである. C 言語ではこれはプログラマの責任であるが, これを手作業で行うことは, 開放し忘れによるメモリリーク, あるいは不適切な開放など, 重大かつ見つけにくいバグの原因となり易い. これを防ぐために, [6] で, 自動ガーベッジコレクタ付きのメモリ管理が提案された. これはフリーなサブルーチンライブラリとして利用できる. この章では, 多項式乗算の高速アルゴリズムとして Karatsuba 法のみを取り上げたが, 高次 (数十次以上) の多項式の乗算が多く必要な場合には FFT アルゴリズムが有効な場合もある. これについては [24] を参照.

第 5 章:

ここで述べられていることのうち, 互除法に関することは Euclid 環において成り立つ. また, 終結式については, その重要性にもかかわらず, 一般的な代数学の講義などで詳しく扱われることは少ないと思われる. [40] などで勉強してみてほしい.

第 6 章:

ここで扱われている内容は, 計算機代数研究における一つのハイライトとも言えるものである. 特に, 有限体上の多項式因数分解は, 楕円曲線暗号などへの直接の応用もあり, 現在も活発に研究されている. また, 有理数体上の因数分解も, 数学における実験ツールとして計算機代数システムを用いる場合に大変有効な機能である. この章

の内容のうち, 有限体, 有理数体上の一変数多項式の因数分解に関しては [24], 多変数多項式の因数分解に関しては [37] あるいは [18] を参照. なお, アルゴリズム 6.37 は分解される多項式の次数 n に関して最悪計算量が $O(2^n)$ となるが, LLL アルゴリズム を用いる多項式時間アルゴリズムが [25] により提案されている. ただしこのアルゴリズムは実用的とはいえず, 一般にはアルゴリズム 6.37 が用いられてきた. ごく最近, 異なる観点から LLL アルゴリズムを用いる方法が [22] により提案された. この方法は, これまでの方法では事実上分解が不可能だった多項式を効率よく分解するなど, 実用的にも優れていることが報告されている.

第 7 章, 第 8 章:

ここでは, グレブナ基底, Buchberger アルゴリズムおよびそれらの応用についてごく基本的なことがらのみに限定して述べている. 詳しくは, [10] [11] を参照. 特に [11] は代数幾何, 可換代数の研究のためのツールとなるアルゴリズムについて, グレブナ基底に限らない最近の成果も含めて幅広く記述されている.

第 9 章:

ここでは, 理論上も応用上も重要なイデアルの準素分解について概説している. アルゴリズムはさまざまな部品から成っており, それらの説明については, [4] を参照. なお, ここで述べたのとは異なる準素分解アルゴリズムが [38] で提案されており, Risa/Asir に実装されている.

第 10 章, 第 11 章:

これらの章はやや特殊な内容を扱っており, グレブナ基底計算をツールとして用いる場合には特に意識する必要はない. しかし, それを単なる便利なブラックボックスと考えると, ちょっとした問題でもすぐに計算が破綻してしまうことは知っておく必要がある. 任意入力からのグレブナ基底計算については, 最近提案された F_4 アルゴリズム [16] が有力であるが, まだ汎用計算機代数システムなどには実装されていないようである. また, change of ordering については, ここで述べた方法の他に Gröbner walk と呼ばれる方法が [9] で提案されている. modular change of ordering および modular RUR は Risa/Asir に実装されている.