数理解析研究所講究録 1456 巻 2005 年 214-219

代数体上のイデアルのグレブナー基底計算について

野呂 正行
MASAYUKI NORO
神戸大学理学部
DEPARTMENT OF MATHEMATICS, KOBE UNIVERSITY

Abstract

本稿では, 代数体, すなわち有理数体 $\mathbf{Q}$ の有限次代数拡大における効率的演算を行うための, 代数的数の簡約および逆元計算について述べる. さらに, それらを用いた, 代数体上での上でのグレブナー基底計算について述べる.

1 代数体上の演算について

1.1 代数体の元の表現方法

よく知られているように, 任意の代数体 F は原始元をもつ. すなわち, ある代数的数 α が存在して $F=\mathbf{Q}(\alpha)$ と書ける. よって, α の $\mathbf{Q}$ 上の最小多項式を $m(t)$ とすれば, $F=\mathbf{Q}[t]/(m(t))$ であり, F の元は一変数多項式で表現できる. しかし, 計算効率を考えた場合, 原始元による表現は, 係数サイズの増大を招きやすいため望ましくない. このため, F を逐次拡大として与えるのが現実的である. すなわち,

$$F_0=\mathbf{Q},\quad F_i=F_{i-1}(\alpha_i)\quad (i=1,\ldots,n),\quad F=F_m$$

で, 各 α_i が F_{i-1} 上代数的で, F_{i-1} 上の最小多項式が $m_i(t,\alpha_1,\ldots,\alpha_{i-1})$ で与えられているとするのである. m_i の既約性のチェックにはしばしば困難な多項式因数分解が必要となるが, 幸い, knapsack factorization アルゴリズムにより, チェック可能な多項式の範囲が大幅に拡大した. 以下では, 既に既約性が保証されている逐次拡大が与えられているとする. イデアルの言葉で述べれば,

$$I=\langle m_1(x_1), m_2(x_1,x_2),\ldots,m_n(x_1,\ldots,x_n)\rangle$$

が $\mathbf{Q}[X]=\mathbf{Q}[x_1,\ldots,x_n]$ の極大イデアルということになる.

1.2 簡約化

各 m_i は,

$$m_i(x_1,\ldots,x_i)=c_{d_i}x_i^{d_i}+c_{d_{i-1}}(x_1,\ldots,x_{i-1})x_i^{d_i-1}+\cdots+c_0(x_1,\ldots,x_{i-1})$$

$(c_{d_i}\in\mathbf{Z},\ c_j(x_1,\ldots,x_{i-1})\in\mathbf{Z}[x_1,\ldots,x_{i-1}])$ という形であるとしてよい. さらに, m_i は $\langle m_1,\ldots,m_{i-1}\rangle$ に関して正規形, すなわち $\deg_{x_j}(m_i)<d_j$ $(j=1,\ldots,i-1)$ とする. このとき, $G=\{m_1,\ldots,m_n\}$ は, $x_n>x_{n-1}>\cdots>x_1$ なる辞書式順序に関する I のグレブナー基底となっている. $Q[X]/I$ の元 $h(x)$ mod I に対し, $h_0\equiv h \bmod I$, $\deg_{x_i}(h_0)<d_i$ なる h_i は G に関する正規形 $\mathrm{NF}_G(h)$ に等しい.

h_0 は, グレブナー基底を持ち出すまでもなく, 1 変数多項式の剰余計算により与えられる. 例えば, h から, $m_n, m_{n-1}, \ldots, m_1$ という順に剰余を計算していけば得られる. もちろん, m_i による剰余は x_i を主変数として計算する. しかしこの方法は, 効率から見た場合最悪である. なぜなら, m_i による剰余を計算すると, 一般に $j < i$ なる x_j に関する次数が上がる. 特に, m_2 までの剰余を計算した段階の剰余は, x_1 に関する次数が大変高くなっている可能性がある. そこで, i が小さい m_i による剰余計算を優先させる方法が考えられる. この方法は, 上の方法よりは効率がよいと考えられるが, やはり下の順序の変数の次数が急に上がることは望ましくない.

この場合, 実は G による単項簡約を用いれば, 効率よく剰余計算を行うことができる. この際, 各簡約ステップに用いる m_i としては, i の小さいものを優先する必要がある.

例 1

$m_1(x_1) = x_1^7 - 7x_1 + 3,$

$m_2(x_1, x_2) = x_2^6 + x_1x_2^5 + x_1^2x_2^4 + x_1^3x_2^3 + x_1^4x_2^2 + x_1^5x_2 + x_1^6 - 7,$

$m_3(x_1, x_2, x_3) = 63x_3^4 + ((-2x_1^5 - 5x_1^4 - x_1^3 - 7x_1^2 + 3x_1 + 6)x_2^5 + (-5x_1^5 - 5x_1^3 - 3x_1^2 + 7x_1)x_2^4 + (-x_1^5 - 5x_1^4 + 8x_1^2 + 4x_1 - 24)x_2^3 + (-7x_1^5 - 3x_1^4 + 8x_1^3 + 6x_1^2 - 28x_1 + 6)x_2^2 + (3x_1^5 + 7x_1^4 + 4x_1^3 - 28x_1^2 + 14x_1 + 18)x_2 + 6x_1^5 - 24x_1^3 + 6x_1^2 + 18x_1 + 12)x_3^3 + ((x_1^4 - 2x_1^3 - 4x_1^2 + 5x_1)x_2^5 + (x_1^5 - 2x_1^3 + 12x_1^2 + 22x_1)x_2^4 + (-2x_1^5 - 2x_1^4 + 8x_1^3 + 20x_1^2 + 15x_1 - 12)x_2^3 + (-4x_1^5 + 12x_1^4 + 20x_1^3 + 12x_1^2 + 5x_1 + 15)x_2^2 + (5x_1^5 + 22x_1^4 + 15x_1^3 + 5x_1^2 + 25x_1 - 6)x_2 - 12x_1^3 + 15x_1^2 - 6x_1 + 48)x_3^2 + ((5x_1^4 + 2x_1^3 + x_1^2 + 16x_1 - 18)x_2^5 + (5x_1^5 + 6x_1^4 - 4x_1^3 + 18x_1^2 - 13x_1)x_2^4 + (2x_1^5 - 4x_1^4 + 16x_1^3 - 11x_1^2 - 18x_1 + 48)x_2^3 + (x_1^5 + 18x_1^4 - 11x_1^3 - 6x_1^2 + 43x_1 - 6)x_2^2 + (16x_1^5 - 13x_1^4 - 18x_1^3 + 43x_1^2 + 14x_1 - 48)x_2 - 18x_1^5 + 48x_1^3 - 6x_1^2 - 48x_1 + 24)x_3 + (4x_1^5 - x_1^4 + 9x_1^3 - 2x_1^2 - 16x_1 + 6)x_2^5 + (-x_1^5 + 6x_1^4 + 2x_1^3 - 15x_1^2 - x_1 + 45)x_2^4 + (9x_1^5 + 2x_1^4 - 22x_1^3 - 5x_1^2 + 40x_1)x_2^3 + (-2x_1^5 - 15x_1^4 - 5x_1^3 + 33x_1^2 - 8x_1 + 84)x_2^2 + (-16x_1^5 - x_1^4 + 40x_1^3 - 8x_1^2 + 36x_1 - 6)x_2 + 6x_1^5 + 45x_1^4 + 84x_1^2 - 6x_1 - 84$

とする. $m_1(\alpha_1) = 0$, $m_2(\alpha_1, \alpha_2) = 0$, $m_3(\alpha_1, \alpha_2, \alpha_3) = 0$ を満たす α_1, α_2, α_3 により $F = \mathbf{Q}(\alpha_1, \alpha_2, \alpha_3)$ とする. $(\alpha_1 + \alpha_2 + \alpha_3)^{20}$ を, m_1, m_2, m_3 の順に用いる単項簡約により簡約すると 0.1 秒で計算できるが, 逆順で行うと 260 秒かかる. この差は簡約される対象が大きくなると急速に増大する.

1.3 逆元計算

簡約計算と並んで代数体上の効率的計算を困難にするものが, 逆元計算である. 単拡大の場合には, 拡張 Euclid 互除法, 例えば部分終結式アルゴリズムにより, 拡大次数 d に関し $O(d^2)$ の手間で計算できる. この方法は, 逐次拡大に対しても再帰的に適用できるが, 計算効率上種々の問題を生じる.

- 中間式膨張

  $h(x_1, \ldots, x_n) \bmod I$ の逆元を計算する場合, まず変数 x_n に関し $m_n(x_1, \ldots, x_n)$ と拡張 Euclid 互除法を適用すると, $ah + bm_n = r(x_1, \ldots, x_{n-1})$ となる a, b, r を得る. $r \bmod I$ の逆元を $s \bmod I$ とすれば, $h \bmod I$ の逆元は $as \bmod I$ となるが, 計算方法によっては r の逆元が巨大になる場合がある.

- 簡約化との関係

  h を $n-1$ 変数多項式環上の 1 変数多項式と見なして部分終結式アルゴリズムを適用すれば, 係数の除算は整除となるが, これらに現われる変数に対する簡約化が行われないので, 一般に大きな多項式が係数に現われる. 一方で, 係数の簡約化を許せば, それらを表現する多項式が大きくならずに計算できるが, 除算自身が体演算となり, 逆元計算が必要となる.

既に Asir の sp パッケージにおける代数体上の一変数多項式 GCD でも用いられているように, 代数的数の計算にはモジュラー計算が有効である [2], [1]. 逆元のモジュラー計算による計算法としては, 次の 2 つが有力である.

- 中国剰余定理

  十分多くの法 p に対し, 法 p での逆元を計算して中国剰余定理により結合し, 整数-有理数変換により逆元を得る方法である. 法 p で考えると, 係数環が体ではなくなるが, 有限個の p を除いて法 p での逆元は計算できる. この計算が, 互除法のように $O(d^2)$ で計算できるなら, 1 ステップあたり $O(d^2)$ の手間を, 結果の係数の大きさに比例する回数繰り返すことで逆元が計算できる.

- 未定係数法 + Hensel lifting

  逆元を未定係数法により求めることもできる. すなわち, mod$I$ での単項式基底

  $$M = \{x_1^{e_1} \cdots x_n^{e_n} \bmod I \,|\, 0 \leq e_i \leq d_i - 1 (i = 1, \ldots, n)\}$$

  により逆元を $u = \sum_{t \in M} a_t t$ と表し, $hu \equiv 1 \bmod I$ から a_t の線形方程式系を作って解く方法である. この方法は, 拡大次数 $d = \prod d_i$ に関し $O(d^3)$ の解法となってしまうが, 線形方程式系の求解に Hensel lifting+整数-有理数変換を使うことで, $O(d^3)$ の部分を有限体上にのみ限定できる. よって, もし結果が大きい係数をもつ場合には, 実際の計算時間は 1 ステップあたり $O(d^2)$ の Hensel lifting により決定される. また, 中間式を作らず, $\mathrm{NF}_G(th)$ $(t \in M)$ の計算のみで線形方程式ができることから, 上で述べたような中間式膨張や体除算による問題は現われない.

2 代数体上のグレブナー基底計算

2.1 monic 化

Buchberger アルゴリズムをはじめとするグレブナー基底計算アルゴリズムは任意体上で実行可能であるが, 体によっては係数膨張の問題が生ずるので, 種々の注意が必要となる. F が代数体の場合, 係数の積の簡約化, および, monic 化における体除算の問題があり, 少くとも Risa/Asir においては, これまで代数体上のグレブナー基底計算は提供してこなかった. 代わりに, 次により計算が可能である.

定理 1
$F = \mathbf{Q}[\alpha_1, \ldots, \alpha_l] = \mathbf{Q}[t_1, \ldots, t_l]/I$, $I = \langle m_1(t_1), \ldots, m_l(t_1, \ldots, t_l) \rangle$, とし, $J = \langle B \rangle \subset R = F[x_1, \ldots, x_n]$ を R の真のイデアルとする. $<$ を R の項順序とする. R の項順序 $<_F$ を $<$ および F の $t_1 < \cdots < t_l$ なる辞書式順序からなるブロック順序 $(t_1^{a_1} \cdots t_l^{a_l} <_F x_1^{b_1} \cdots x_n^{b_n})$ とする. このとき, $B_F = B \cup \{m_1, \ldots, m_l\}$ の $<_F$ に関するグレブナー基底 G_F の元のうち変数 $x_1, \ldots, x_n$ を含むものの集合 G は, J の $<$ に関するグレブナー基底となる. G_F が簡約グレブナー基底ならば, G の元の頭項は $t_1, \ldots, t_l$ を含まない.

G_F を計算する場合, 実際のアルゴリズムの実行を観察すると, 生成される中間基底の頭項の t 変数がだんだん減って行き遂に消滅することが分かる. すなわち, 頭項の係数の逆元をかける操作を S-多項式と単項簡約により行っていることになる. このことは S-多項式の数の増大を引き起こす. S-多項式の処理の順序は sugar strategy など, なんらかの基準により機械的に決められるが, S-多項式が増えると, 不適切な順序で処理される可能性も増え, 不必要な係数膨張を引き起こす可能性がある. そこで, 前節の逆元計算を用いて, Buchberger アルゴリズムにおける, 0 に簡約されない元の基底の処理を次のように変更する:

- 通常の処理

  $S(f,g)\xrightarrow[G]{*}h \neq 0$ ならば, $G \leftarrow G \cup \{h\}$

- 変更後の処理

  $S(f,g)\xrightarrow[G]{*}h(x,t) \neq 0$ ならば, $h(x,\alpha)$ を monic 化したもの $\tilde{h}(x,\alpha)$ を作り, $G \leftarrow G \cup \{\tilde{h}(x,t)\}$

このような変更を行っても G_F が計算できることはあきらかであろう.

2.2 trace アルゴリズム

monic 化を行ったとしても, 実際には trace アルゴリズムを適用するのが安全であろう. mod$p$ での trace アルゴリズムの正当性は, $\mathbf{Q}$ 上の簡約化に p で割り切れる分母をもつ有理数が現われないこと, さらに簡約結果の正規形の頭係数が p で割り切れないことが必要となる. 前節の変更を行った場合, 厳密には, 頭係数の逆元計算にも p で割れる分母が現われないことを要求することになるが, この条件を緩めて, 結果を monic 化したあと, p で割れる分母が現われないこと, とすることもできる. 実際, この結果自身は入力イデアルの元であることは確かで, この元があらかじめイデアルの生成元の一つであったと考えればよい.

trace アルゴリズムの実現についてはいくつかのバリエーションが考えられる. たとえば, $\overline{I} = I \bmod p$ が根基イデアルになるような p をとれば, $GF(p)[t]/\overline{I}$ がいくつかの有限体の直和に分解する. I_p を, I の係数を $\mathbf{Q}_{<p>} = \{a/b \mid a,b \in Z, p \not| b\}$ に制限したものとし, ϕ を I_p からある直和成分への射影とすれば, これも trace アルゴリズムを実現する. これにより, trace の計算は, 有限体上での入力多項式集合のグレブナー基底計算となるため, 計算の手間を減らせる可能性がある.

3 Risa/Asir 上での実装

3.1 逐次代数拡大

Risa/Asir には, 逐次代数拡大の元の表現として, 再帰表現多項式をボディ部にもつ Alg があり, 代数体上の因数分解および最小多項式計算に既に用いられているが, 簡約計算, 逆元計算を含むほとんどの関数は, sp 中にユーザ関数として書かれていて高効率は望めない. 特に, 既に述べたような, 単項簡約による簡約を行うのには適していないため, 分散表現多項式をボディ部にもつ逐次代数拡大の表現を新たに定義した.

```
typedef struct oDAlg {
        short id;
        char nid;
        char pad;
        struct oDP *nm;
        struct oQ *dn;
} *DAlg;
```

DAlg はボディ部として, 分散表現多項式である nm および有理数である dn をもつ. 代数的数としては nm/dn を表すが, nm は整数係数の分散表現多項式, dn は整数に限定する. すなわち, ボディ部の有理数係数を通分して表示すると約束するのである. また, 係数体である逐次拡大体を, それを生成する代数的数のリストで指定する関数 set_field() を用意した. この呼び出しにより, 簡約計算, 逆元計算に必要な, 最小多項式リストや単項式基底が計算され, 内部的に係数体情報としてセットされる.

DAlg 型のオブジェクトは, Alg 型からの変換関数 algtodalg() により生成される. その逆変換は dalgtoalg() である.

3.2 グレブナー基底計算

現在の試験的実装は, nd_gr および nd_gr_trace を, 入力多項式集合に最小多項式を追加して実行し, 前節で述べたような変更 (monic 化) を加える, という形で行っている. これに対し, 代数的数を完全に係数に取り込んで, 係数に関して既に述べたような簡約化および逆元計算を行うという方法も考えられる. 前者の場合, 簡約化が $\mathbf{Q}$ 上で行われるので, 既に実装してある, 係数の content 除去が自動的に適用される. sugar 値に, 代数的数の次数が反映されるのを防ぐため, 代数的数に対応する weight を 0 に設定している. しかし, 斉次化については不自然な実装をせざるを得ない. 後者は自然な実装とも言えるが, content 除去に相当する方法を新たに考案する必要があり, 今後の課題とした.

4 タイミングデータ

以下, 特に断らない限り, 項順序は DRL とする.

例 2

$$
\begin{aligned}
Cyc &= \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}\\
f_1 &= \omega c_5c_4c_3c_2c_1c_0 - 1\\
f_2 &= (((((c_5+\omega)c_4+\omega c_5)c_3+\omega c_5c_4)c_2+\omega c_5c_4c_3)c_1+\omega c_5c_4c_3c_2)c_0+\omega c_5c_4c_3c_2c_1\\
f_3 &= ((((c_4+\omega)c_3+\omega c_5)c_2+\omega c_5c_4)c_1+\omega c_5c_4c_3)c_0+c_5c_4c_3c_2c_1+\omega c_5c_4c_3c_2\\
f_4 &= (((c_3+\omega)c_2+\omega c_5)c_1+\omega c_5c_4)c_0+c_4c_3c_2c_1+c_5c_4c_3c_2+\omega c_5c_4c_3\\
f_5 &= ((c_2+\omega)c_1+\omega c_5)c_0+c_3c_2c_1+c_4c_3c_2+c_5c_4c_3+\omega c_5c_4\\
f_6 &= (c_1+\omega)c_0+c_2c_1+c_3c_2+c_4c_3+c_5c_4+\omega c_5\\
f_7 &= c_0+c_1+c_2+c_3+c_4+c_5+\omega
\end{aligned}
$$

Cyc は, cyclic-7 において, c_6 を 1 の原始 7 乗根 ω に置き換えたものである. Cyc の $\mathbf{Q}(\omega)$ 上でのグレブナー基底計算は, 斉次化 trace アルゴリズムにより 22 秒で計算できる. このうち, monic 化の計算時間は 2.2 秒, そのうち逆元計算は 0.2 秒 である.

ω の最小多項式を添加して $\mathbf{Q}$ 上で計算する場合 220 秒かかる.

例 3

$$
\begin{aligned}
Cap &= \{f_1, f_2, f_3, f_4\}\\
f_1 &= (2ty-2)x-(\alpha+\beta)zy^2-z\\
f_2 &= 2\beta\alpha^4zx^3+(4ty+\beta)x^2+(4zy^2+4z)x+2ty^3-10y^2-10ty+2\alpha^2+\beta^2\\
f_3 &= (t^2-1)x+(\beta\alpha^4+\beta^3\alpha^3)tzy-2z\\
f_4 &= (-z^2+4t^2+\beta\alpha+2\beta^3)zx+(4tz^2+2t^3-10t)y+4z^2-10t^2+\beta\alpha^3
\end{aligned}
$$

$$m_1 = u^7 - 7u + 3$$
$$m_2 = u^6 + \alpha u^5 + \alpha^2 u^4 + \alpha^3 u^3 + \alpha^4 u^2 + \alpha^5 u + \alpha^6 - 7$$

Cap は *Caprasse* [4] の係数をランダムに代数的数に置き換えたものである. α, β は $t^7 - 7t + 3$ の 2 根で, α の $\mathbf{Q}$ 上の最小多項式が $m_1(u)$, β の $\mathbf{Q}(\alpha)$ 上の最小多項式が $m_2(u)$ である. *Cap* の $\mathbf{Q}(\alpha, \beta)$ 上でのグレブナー基底計算は trace アルゴリズムで 589 秒 (内 monic 化 36 秒), $\mathbf{Q}$ 上では斉次化しても, 1 時間待っても終了しない.

例 4
例 1 で定義される F は, $f(x) = x^7 - 7x + 3$ の最小分解体である. $f(x)$ の F 上因数分解に現れる, F 上の 2 つの 2 次式の GCD 計算 (GCD は 1 次式) を F 上のグレブナー基底計算により行うと, 0.8 秒で終了する. 逆元計算 1 回分が計算時間の大部分を占める. 一方, これを $\mathbf{Q}$ 上で行うと, 60 秒かかる.

5 おわりに

本稿で提案した方法により, 代数体上のグレブナー基底計算が, 単に最小多項式を添加して有理数体上で計算するより高速に計算できる場合があることが示せた. しかし, このような計算が困難であることには変わりない. より本質的な改良, 新アルゴリズムも必要であろう.

今後の予定として以下を挙げておく.

- DCGB との関係

  佐藤ら [3] による DCGB も, 同様に代数体上のグレブナー基底を与える. この方法との比較は興味ある話題であろう.

- change of ordering, RUR

  実際の求解に必要になる, FGLM や RUR の計算を代数体上に拡張することは必要であろう. この場合にも, 線形方程式求解を代数体上で行うか, 有理数体上で行うかの選択が生ずる. これらの比較も今後の課題である.

- 代数体演算の実装の効率化

  今回の実装では, 代数体の表現として, Asir 旧来の分散多項式型である DP を用いた. これは, nd 系関数が現状では再入不可であるというのが主な理由であった. nd の開発動機が DP 型の計算効率への不満であったことを考えれば, この部分の改良は意味がある. 特に, triangular form による簡約化に特化したデータ型および関数を書くことにより, より一層の効率化が達成できるであろう.

参 考 文 献

[1] M.v. Hoeij, M. Monagan, A Modular GCD algorithm over Number Fields presented with Multiple Extensions. Proc. ISSAC'02 (2002), 109-116.

[2] 野呂, 逐次代数拡大体上での 1 変数多項式の GCD について. 数理研講究録 920 (1996), 1-8.

[3] Y. Sato, A. Suzuki, Discrete Comprehensive Gröbner Bases. Proc. ISSAC'01 (2001), 292-296.

[4] SymbolicData. http://www.SymbolicData.org.