数理解析研究所講究録
1171 巻 2000 年 82-92

Implementation of the F_4 algorithm in Asir

富士通研究所
野呂 正行

平成12 年8 月9 日

1 Introduction

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 Gröbner 基底の計算が dominant step となることが多い. Gröbner 基底の計算法としては Bucbberger 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 アルゴリズムが提案され, その高速性が注目されている. 本稿では, このアルゴリズムの大まかな解説を試みる. 併せて, Risa/Asir 上での実装について述べ, その実験結果を示す.

2 Buchberger アルゴリズムの復習

R を体 K 上の多項式環とする. 係数 1 の単項式を項 (term) と呼ぶ. 項の集合に対し,

1. 任意の項 t に対し, $1 < t$.
2. 項 t_1, t_2 に対し $t_1 < t_2$ なら, 任意の項 s に対し $st_1 < st_2$

なる全順序 $<$ が与えられているとする.

Notation 2.0.1
$f \in R$ の頭項, 頭係数を $HT(f)$, $HC(g)$ と書く.
$f - HC(f)HT(f)$ を $reductum(f)$ と書く.
$T(f)$ で, f に現れる項全ての集合を表す.
f の多項式集合 F による正規形 (の一つ) を $NF(f, F)$ と書く.

定義 2.0.2 $f, g \in R$ に対し, **S-多項式** $Sp(f, g)$ を,

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

($T_{fg} = LCM(HT(f), HT(g))$) と定義する.

命題 2.0.3 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, $Sp(f, g) \xrightarrow[G]{*} 0$

これから次のアルゴリズムが導かれる.

アルゴリズム 2.0.4 (Buchberger[1])
入力 : R の有限部分集合 $F = f_1, \cdots, f_l$
出力 : F で生成されるイデアルの グレブナ基底 G
$D \leftarrow \{\{f, g\} | f, g \in F; f \neq g\}$
$G \leftarrow F$
while ($D \neq \emptyset$) *do* {
  $\{f, g\} \leftarrow D$ の元
  $D \leftarrow D \setminus \{\{f, g\}\}$
  $h \leftarrow NF(Sp(f, g), G)$
  if $h \neq 0$ *then* {
    $D \leftarrow D \cup \{\{f, h\} | f \in G\}$
    $G \leftarrow G \cup \{h\}$
  }
}
return G

このアルゴリズムを実行する場合, D からどの元を選ぶかで, その後の計算の進行の様子が全く異なる場合がある. 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. そのため以下のような種々の改良が提案され用いられてきた.

- sugar strategy

  多項式, S-多項式に仮想的な homogeneous degree を付与し, その小さいものから選ぶ.

- trace lifting

  あらかじめ小標数の有限体上で normal form を計算し, 0 でないものだけ整数上で計算する. (最後にチェックが必要)

- homogenization

  あらかじめ, 入力多項式を斉次化して計算する. trace lifting で係数膨張が起きた場合に有効.

- change of ordering

  他の term order の Gröbner 基底から線形代数的手法で変換する.

これらはそれぞれ有効性が実証されているが, いずれの場合にも $NF(Sp(f,g),G)$ の計算が dominant となる. 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 から t を消去)

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 2.0.5 *(Symbolic preprocessing)*

入力 : 多項式 f, 多項式集合 G
出力 : $Red = \{ah | a : 単項式, h \in G\}$
$T \leftarrow T(f)$
$Red \leftarrow \emptyset$
while $\exists t \in T \exists g \in G\, s.t.\, HT(g)|t$ *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 $\{HT(r)|r \in Red\}$ に含まれないものが, $NF(f, G)$ となる.

3 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 3.0.6 *(Symbolic preprocessing)*

入力 : 多項式集合 F, 多項式集合 G
出力 : $Red = \{ah|a : 単項式, h \in G\}$
$T \leftarrow \cup_{f \in F} T(f)$
$Red \leftarrow \emptyset$
while $\exists t \in T \exists g \in G\, s.t.\, HT(g)|t$ *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 Gröbner 基底を計算した結果に対応する.

$$F' = \{h = poly(B_i) | h \neq 0, HT(h) \notin \{HT(r) | r \in Red\}\}$$
$$Red' = \{h = poly(B_i) | h \neq 0, HT(h) \in \{HT(r) | r \in Red\}\}$$

とおく.

命題 3.0.7 *1.* $h \in F'$ は $G \cup (F' \setminus \{h\})$ に関して正規形
2. $f \in F$ ならば $f \xrightarrow[G \cup F']{*} 0$

[証明] 1: $h \in F'$ ならば $T(h) \cap \{HT(r) | r \in Red\} = \emptyset$. 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. (証明終)
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 3.0.8 *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 Gröbner 基底のうちの, d 次の元全てとなる. これは, homogeneous ideal の場合, 次から言える.

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.

4 実装 (I) — Symbolic preprocessing

Risa/Asir では, Buchberger アルゴリズムのための多項式のデータ表現として, 分散表現 (distributed representation) を用いている. これは, 多項式を, (係数, 項) なるペアのリストとして表すものである. symbolic preprocessing の実装は次のようにすればよい.

- 分散表現多項式を, 係数を忘れて項の sorted list とみなす.
- sorted list 同士の merge 関数を用意する.
- S-多項式の集合を, merge 関数で一つの sorted list S とする.
- S の上位の項から順に, それを簡約できる reducer R があれば, S と R を merge して S とする.

5 実装 (II) — ガウス消去

5.1 有限体上の場合

有限体係数の場合, ガウス消去により得られた reducer の正規形の再利用を行わないのなら, 幾つかの簡略化が考えられる. 例えば,

1. 各 S-多項式を reducer 集合で簡約してから, S-多項式のみで構成した行列のみでガウス消去を行う.
2. reducer 集合のみで構成される行列でガウス消去を行ったあと, 1. を行う.

これらの場合, F_4 アルゴリズムの効果としては,

- ベクトル形式で正規形の計算ができる.
- ベクトル形式の reducer が共有できる.

が主要なものとなる.

5.2 有理数体上の場合

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.

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. Hensel 構成は以下のようなアルゴリズムによる.

アルゴリズム 5.2.1
solve_linear_equation_by_hensel(M, B, p)
Input : an $n \times n$ *matrix* M, *an* $n \times m$ *matrix* B, *a prime* p *s.t.* $\phi_p(det(M)) \neq 0$
Output : an $n \times 1$ *matrix* X *s.t.* $MX = B$

$R \leftarrow \phi_p(M)^{-1}$; $c \leftarrow B$; $x \leftarrow 0$; $q \leftarrow 1$; $count \leftarrow 0$
do {
  $t \leftarrow \phi_p^{-1}(R\phi_p(c))$; $x \leftarrow x + qt$; $c \leftarrow (c - Mt)/p$; $q \leftarrow qp$; $count \leftarrow count + 1$
  (ϕ_p^{-1} denotes the canonical inverse image; $(c - Mt)/p$ is an exact division.)
  if $count =$ **Predetermined_Constant** *then* {
    $count \leftarrow 0$
    $X \leftarrow inttorat(x, q)$
    if $X \neq$ **nil** *and* $MX = B$ *then return* X
  }
}

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 Gröbner 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.

6 実験

表 1 は, いくつかのベンチマーク問題に体する有限体上での F_4 実装の比較, 表 2 は, 有理数体上での F_4 実装の比較, 表 3 は, [3] で計算した, odd order replicable function に付随する方程式 (odd order equation) の計算時間の比較 , 表 4 は, F_4 の各ステップで現れる行列のサイズ, 表 5 は, 各ステップでの基底の計算時間, 図 6 は, 中間基底の最大係数のビット長をそれぞれ表す. 表 1, 2 は Pentuim III 700MHz 上での実行結果で, FGb (Faugère による実装) のタイミングデータは換算値である. その他は, Alpha 21164 533MHz 上での実行結果である. (単位は秒)

	C_7	C_8	K_7	K_8	K_9
Asir F_4	11	288	1.8	18	183
Asir Buchberger	82	3626	7.6	77	—
FGb	1.3	33	0.2	1.2	8.6

表 1: 有限体上での F_4 実装の比較

	C_7	K_7	K_8	F_{855}
Asir F_4	1059	92	1053	—
F_4 solve	582	48	510	—
F_4 check	471	43	537	—
Asir Buchberger	490	29	233	377
FGb	11	0.8	6.6	5.7

表 2: 有理数体上での F_4 実装の比較

	total	GaussElim	ChRem	IntRat	Check
Buchberger	264710	—	—	—	—
F_4 homo	32880	6437	6300	5763	13340
F_4 non homo	39970	4832	6143	18240	9950

表 3: odd order equation : 計算時間の比較

7 考察

有限体上の結果については, 未だ FGb と 1 桁以上の開きがあることが分かる. Asir 上でさらに有限体上での F_4 実装を高速化するためには, 以下のような方法が考えられる.

	10	11	12	13	14
F_4 homo	(56,334)	(119,441)	(182,545)	(362,774)	(671,1009)
F_4 non homo	(62,339)	(128,448)	(137,461)	(227,571)	(307,621)

15	16	17	18	19	20
(1195,1359)	(836,688)	(2323,1761)	(895,964)	(204,271)	(446,515)
(955,1149)	(555,508)	(2022,1763)	—	—	—

21	22	23	24	25
(308,374)	—	—	—	—
—	(799,868)	(300,367)	(398,467)	(361,427)

表 4: odd order equation : 各次数の行列のサイズ

	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
205234	10300	4530	—	10700	—	—	—	—	—
1340	1097	359	45	208	150	—	—	—	—
937	902	—	—	—	—	341	132	186	164

表 5: 各次数の基底の計算時間

- 構造化ガウス消去

  ガウス消去すべき行列は一般に疎であり, 現状の Risa/Asir における実装のようにそのままガウス消去するのは効率が悪い. [2] では, あらかじめ行列をスキャンして, 行, 列の入れ換えを行い, 小さいサイズの密行列を取り出してガウス消去する方法を提案している.

- 簡約された reducer の再利用

  [2] では, 既に得られている簡約された reducer を, reducer として用いるという方法を提案している. この方法によれば, 確かに各ガウス消去の手間が軽減されることが期待できるが, 大量に現れる簡約された reducer をどう保持するかが問題となる.

- 原始根表現の利用

  有限体 $GF(p)$ の演算として, Risa/Asir では通常の法 p での加減乗除算を行っているが, この方法では, 乗算のたびに, 整数乗算, 除算それぞれ一回が必要となる. p がある

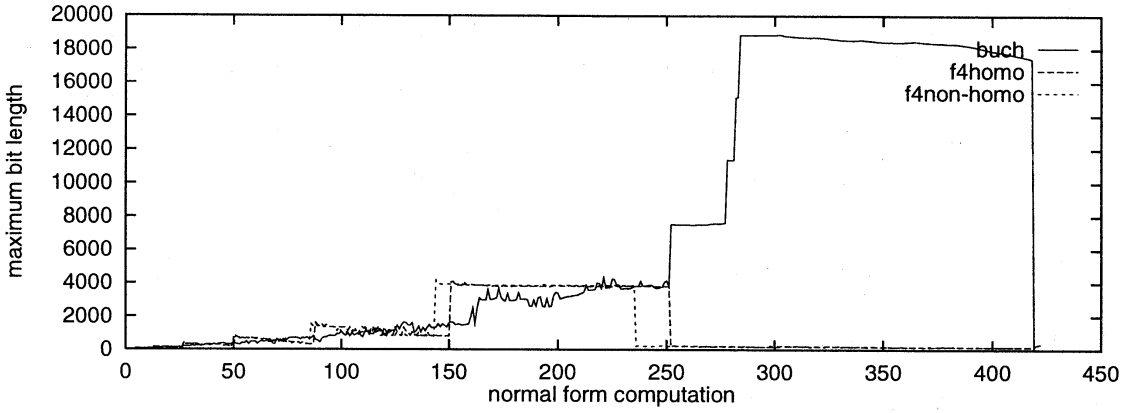

図 1: 中間基底の最大係数のビット長

程度小さい (例えば 2^{16} 未満) の場合, 法 p での原始根を使って $GF(p)^{\times}$ を指数で表現した表を保持することで, 乗算は加算, 加算は表参照で行える. この方法は, Faugère 自身が旧版の Gb (Buchberger アルゴリズムによるグレブナ基底計算プログラム) で用いていて, FGb でも用いられている可能性がある.

有理数体上の結果は, FGb とは全く比較にならない. これに関しては未だ理由が不明である. 一方で, odd order equation に関しては, Asir 上での Buchberger アルゴリズムによる計算に比べて, Asir 上での F_4 実装により, 計算時間が 1/8 程度に減少している. これは,

- 16 次の reduced bases の係数が小さくなるため 16 次の計算が 1/100 程度になったこと,
- 小さい係数を持つ多項式による reduction で済むため, 17 次以降も計算時間が短縮したこと

による. とは言え, odd order equation については FGb on PentiumPro 200MHz で 54 秒という結果が報告されている. これは reduced でも巨大な 15 次の基底の生成, チェックが 50 秒程度でできるということで, 現在の Risa/Asir の実装では考えられないことである. ちなみに Asir では Chinese remainder に 329 段, Check に 10^4 秒かかっている. 現状では, reducer の置き換えが効いている以外に考えられない.

8 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. 高速化の理由としては,

- 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」さら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. ただし, この場合には, 行列の各行の 0 簡約チェックが必須である.

といったことが考えられるが, Risa/Asir の実装がまだ甘いか, あるいは何か重大な見落としがあるかで, FGb 程に高速な実装には至っていない. いずれにせよ

- sparse matrix に対する Gauss 消去の効率化
- memory を大量消費しない工夫
- 有限体上での原始根表現の利用

といったことに留意して効率化を目指す必要がある.

参考文献

[1] Buchberger, B., Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Doctoral Dissertation Math. Inst. University of Innsbruck, Austria(1965).

[2] Jean-Charles Faugère, A new efficient algorithm for computing Groebner bases (F_4), Journal of Pure and Applied Algebra (139) 1-3 (1999), 61-88.

[3] Noro, M., J. McKay, Computation of replicable functions on Risa/Asir. Proc. PASCO'97, ACM Press (1997), 130-138.