Signature based algorithm における F_4 スタイルの簡約アルゴリズムの実装について

Implementation of F_4 Implementation of F_4-like reduction in signature based algorithm

立教大学理学部　　野呂 正行 [*1]
Masayuki Noro
Faculty of Science
Rikkyo University

Abstract

A signature-based algorithm (SBA) computes a Groebner basis of an ideal by attaching a module monomial called signature to an element of the ideal and by finding elements of the ideal with smaller signature first. The algorithm reduces S-polynomials by an intermediate basis as in the Buchberger algorithm. However it is difficult to directly apply a F_4-like reduction of a matrix constructed from many S-polynomials and reducers because only a restricted reduction called regular reduction is allowed. In this article we propose a vectorized polynomial reduction within the framework of the sequential SBA. We report the idea for realizing this method and we show a comparison between the new implementation and the existing SBA implementation in Risa/Asir.

1　概要

signature based algorithm (SBA) は, イデアルの元に signature と呼ばれるある加群単項式を対応させ, 小さい signature を持つイデアルの元から求めていくことで, 最終的にグレブナー基底を計算する. Buchberger algorithm と同様に S 多項式を中間基底で簡約することで algorithm が進行するが, S 多項式の選択順序および簡約に条件がつくため, F_4 アルゴリズムのように, 一度に多くの S 多項式を取り出して, それをを簡約するのに十分な reducer を準備して, 行列上で簡約を行う, という方法が取りにくい. そこで, SBA における S 多項式の選択順序に従って, 毎回 1 つの S 多項式を簡約するが, 簡約自体は S 多項式, reducer をベクトル化して, ベクトルに対する簡約で行う方法を考案した. この方法を実現するためのアイディア, およびこの実装と, 従来の SBA 実装の計算機実験による比較について報告する.

謝　　辞

この研究は JSPS 科研費基盤研究 (C) 21K03377 の助成をうけています。

This work was supported by the Research Institute for Mathematical Sciences, an International Joint Usage/Research Center located in Kyoto University.

[*1] E-mail: noro@rikkyo.ac.jp

2 Signature based algorithm

signature based algorithm を述べるために, 最小限の定義を与える. 一般の $f \in I$ の signature, $\mathfrak{S}$-グレブナー基底の定義を含め, 詳細は [6] を参照してほしい. K を体, $R = K[x_1, \ldots, x_n]$ とする. $I = \langle f_1, \ldots, f_m \rangle$ を R のイデアルとする. M を R の形数 1 の単項式全体とする. $<$ を R の項順序, $\prec$ を $R^\ell = Re_1 \oplus \cdots \oplus Re_\ell$ の加群項順序とする. アルゴリズム中で, 中間基底集合に追加される多項式 f に対し, f の signature $S(f)$ が定義される. $S(f)$ は R^m の加群単項式である. signature が与えられている多項式 g, g' の S 多項式が $cmg - c'm'g'$ $(c, c' \in K,\ m, m' \in M)$ のとき, $mS(g) \neq m'S(g)$ なら $p = (g, g')$ を regular な S ペアと呼ぶ. $mS(g)$, $m'S(g')$ のうち大きい方を $S(p)$ と定義する.

Algorithm 1 signature based algorithm

Input: $F = (f_1, \ldots, f_m)$
Output: $\langle F \rangle$ の $\mathfrak{S}$-グレブナー基底

1: $S(f_i) \leftarrow e_i$ $(i = 1, \ldots, m)$; $Syz \leftarrow \emptyset$; $G \leftarrow \emptyset$
2: **for** $f \in F$ **do** $Update(f, G, D)$
3: **while** $D \neq \emptyset$ **do**
4: $\quad s \leftarrow \min\{S(p) \mid p \in D\}$; $p \leftarrow S(p) = s$ なる $p \in D$; $D \leftarrow D \setminus \{p\}$
5: $\quad$ **if** s を割り切る Syz の元がない **then**
6: $\quad\quad$ **if** $\mathrm{LM}(p) = \min\{\mathrm{LM}(mg) \mid g \in G, m \in M, mS(g) = s\}$ **then**
7: $\quad\quad\quad r \leftarrow RegularReduce(\mathrm{Spoly}(p), G)$
8: $\quad\quad\quad$ **if** $r \neq 0$ **then**
9: $\quad\quad\quad\quad S(r) \leftarrow s$; $Update(r, G, D)$
10: $\quad\quad\quad$ **else**
11: $\quad\quad\quad\quad Syz \leftarrow Syz \cup \{s\}$
12: **return** G

Algorithm 1 において, $RegularReduce$ は, $mS(g) \prec S(p)$ なる $m \in M$, $g \in G$ のみを使って $\mathrm{Spoly}(p)$ を簡約する. また, $Update$ は, 新しく生成された剰余 r から生成されたペアを D に追加する際, 同一 signature のペアのうち, LM が最小のもののみ残す.

Algorithm 2 $Update(r, G, D)$

1: $New \leftarrow G$ と r から作った regular S ペア
2: **for** $p \in New$ **do**
3: $\quad$ **if** $S(q) = S(p)$ なる $q \in D$ が存在する **then**
4: $\quad\quad$ **if** $\mathrm{LM}(p) < \mathrm{LM}(q)$ **then** $D \leftarrow (D \setminus \{q\}) \cup \{p\}$
5: $\quad$ **else** $D \leftarrow D \cup \{p\}$
6: $G \leftarrow G \cup \{r\}$

Algorithm 1 において最もコストがかかるのは $RegularReduce$ である. そのコストは通常の Buchberger algorithm におけるのと同様, 項比較と係数演算のコストからなるが, 前者のコストを下げる方法として, F_4 algorithm と同様にベクトル形式での簡約を行うことを試みる.

3 SBA における F_4 スタイルの簡約

項順序 $<$, 加群項順序 $\prec$ は次を満たすことを仮定する:
$t, s \in M$ とする.

1. $<$ は次数つき順序, すなわち $\mathrm{tdeg}(t) < \mathrm{tdeg}(s)$ ならば $t < s$.
2. $<$ と $\prec$ は compatible, すなわち $t < s$ ならば $te_i \prec se_i$.
3. イデアルの生成系を $(f_1, \ldots, f_m)$ とするとき, $\mathrm{tdeg}(t\mathrm{LM}(f_i)) < \mathrm{tdeg}(s\mathrm{LM}(f_j))$ ならば $te_i \prec se_j$.

例として $<$ を grevlex, $\prec$ を $<$ 上の Schreyer 順序とすれば, これらの仮定は満たされる.
以下で, signature が te_i であるとき, その全次数 $\mathrm{tdeg}(te_i)$ を $\mathrm{tdeg}(t\mathrm{LM}(f_i))$ と定義する.

3.1 F_4 algorithm の構成

F_4 algorithm においては, 複数の S ペアをまとめてベクトル形式で簡約するが, その際の手順は以下の通りである.

1. ある条件を満たす S ペアを集める.
   通常は, 全次数最小の S ペアを全部取り出す.
2. 取り出した S ペアを全て簡約するのに十分な reducer リストを作る.
   symbolic preprocessing と呼ばれるアルゴリズム (Algorithm 3) で行う.
   簡約に現れる可能性がある単項式のリストも得られる.
3. 単項式のリストに従って, S 多項式, reducer をベクトルに変換する.
4. 得られたベクトルを並べて行列を作り, 簡約する.

$T(p)$ を p に現れる単項式全体のリストとする.

Algorithm 3 $SymbolicPreproc(S, G)$

Input : S ペアの集合 S, 多項式集合 G
Output :$P = \{\mathrm{Spoly}(p) \mid p \in S\}$ の元を G に関する正規形に簡約できる
多項式集合 Red および,
$P \cup Red$ に現れる全ての単項式を並べた列 T

1: $w \leftarrow ()$
2: **for** $p \in S$ **do** $w \leftarrow \mathtt{merge}(w, T(\mathrm{Spoly}(p)))$
3: $Red \leftarrow \emptyset$; $T \leftarrow ()$
4: **while** $w \neq ()$ **do**
5: **if** $\mathrm{LM}(g_i) \mid \mathrm{TOP}(w)$ を満たす g_i が存在する **then**
6: $r \leftarrow \frac{\mathrm{TOP}(w)}{\mathrm{LM}(g_i)} \cdot g_i$; $T \leftarrow \mathtt{merge}(T, T(r))$; $Red \leftarrow Red \cup \{r\}$
7: w から先頭を外して T の末尾に追加する
8: **return** (Red,T)

3.2 SBA で F_4 スタイルの簡約を行う場合の問題点

3 節で述べた $\prec$ に関する仮定の元では, SBA は, S ペアが全次数の小さい順に処理される. SBA において, 全次数 $d-1$ 次の S ペアが全て処理されたとし, S_d を全次数 d 次の S ペア全体とする. これを, F_4 と同様に S_d および reducer 集合を行列に変換して簡約しようとすると, 以下のような点に注意する必要がある.

1. 逐次的な regular 簡約で行われる剰余計算の保証

   単項式 m が reducer h で簡約できるかどうかは, m が由来する S 多項式の signature に依存する. よって, F_4 の symbolic preprocessing のままでは不十分で, signature を考慮した symbolic preprocessing を行う必要がある.

2. S_d だけで signature の全次数が d の基底が尽くされるとは限らない

   regular 簡約の制限により, $\mathrm{LM}(g)$ が他の $\mathrm{LM}(h)$ $(h \in G)$ で割り切れる g が生成される場合がある. この場合, 新たに d 次の S ペアが生成される場合があり, d 次の処理中に新たな行が行列に追加されることになる. また, symbolic preprocessing で得られた reducer list では新しい S ペアを簡約できない可能性がある.

特に, 2. の理由により, S_d から作った行列から得られた結果を G に追加する方法では algorithm の正当性を示す見通しが立たない. よって, 行列で簡約する方法は行わず, アルゴリズム自体は Algorithm 1 を実行し, 簡約操作のみをベクトルに変換して行う方法を採用することにする. より詳しくは次のようになる:

- S_d の各元の regular 簡約を保証する reducer リストを signature を考慮した symbolic preprocessing で作り, 疎なベクトルに変換しておく.
- Algorithm 1 と同様, 残っている S_d の元のうち最小 signature の元をベクトルに変換して, reducer リストで regular 簡約する. regular 簡約の結果生じた 0 でない剰余はベクトルに変換して reducer リストに追加する.

3.3 SBA における簡約のベクトル化

3.3.1 signature を考慮した symbolic preprocessing

symbolic preprocessing 中に単項式 t が現れたとき, 選ぶべき reducer g は, $\mathrm{LM}(g) \mid t$ だけでなく, t が signature s の S ペアに由来する場合, $s \succ \frac{t}{\mathrm{LM}(g)} S(g)$ も満たす必要がある. さらに, t は複数の S ペアに由来する可能性がある. どの S ペアの簡約に t が現れても, それが逐次的な regular 簡約で簡約できる場合には, 簡約できる g を選んでおく必要がある. このような reducer の見つけ方として, 少なくとも 2 つの方法が考えられる.

方法 1 $\mathrm{LM}(g) \mid t$ かつ $\frac{t}{\mathrm{LM}(g)} S(g)$ が最小の g を探す

t がどの S 多項式の regular 簡約に現れて, 実際に regular 簡約可能でも, この g で regular 簡約できるが, 常に, その時点での G を全てスキャンする必要がある.

方法 2 $s_{min} = \min\{S(p) \mid p \in S_d\}$ とし, $\frac{t}{\mathrm{LM}(g)} S(g) \prec s_{min}$ を満たす g があればそれを使う. ない場合には $\frac{t}{\mathrm{LM}(g)} S(g)$ 最小のものを使う.

この条件でも regular 簡約が完遂できる.

3.3.2 新たに生じた d 次の S ペアの処理

前節で述べたように S_d の元の処理中に新たに d 次の S ペアが生じる可能性があり, その S 多項式中に symbolic preprocessing では現れなかった単項式が現れる可能性があるこのような単項式を簡約できる reducer を探しておく必要がある. S_d から symbolic preprocessing で得られた単項式リストを T, reducer リストを Red とする. $p \notin S_d$ なる S ペアを処理する場合,

- $T(\mathrm{Spoly}(p)) \subset T$ の場合

  $\mathrm{Spoly}(p)$ を regular 簡約するのに十分な reducer がすでにリストにあるので Red はそのまま使える.

- $T(\mathrm{Spoly}(p)) \not\subset T$ の場合

  $\mathrm{Spoly}(p)$ に対する symbolic preprocessing の結果得られた単項式リスト, reducer リストをそれぞれ T, Red にマージする.

3.3.3 F_4 スタイルの簡約を行う SBA

前節で述べた方法を取り入れた SBA が Algorithm 4 である. *SymbolicPreprocSig* (Algorithm 5) は, signature を考慮した symbolic preprocessing を実行する.

Algorithm 4 *SignatureBasadAlgorithm*$F_4(F)$

Input: 多項式集合 $F = (f_1, \ldots, f_m)$
Output: $\langle F \rangle$ の $\mathfrak{S}$-グレブナー基底

```
1:  S(f_i) ← e_i (i = 1,...,m); S ← ∅
2:  for f ∈ F do Update(f, G, D)
3:  while D ≠ ∅ do
4:      d ← min{tdeg(S(p)) | p ∈ D}; S_d ← {p | tdeg(S(p)) = d}; D ← D \ S_d
5:      (R, T) ← SymbolicPreprocSig(S_d, G); VR ← PtoV(R, T)
6:      while S_d ≠ ∅ do
7:          s ← min{S(p) | p ∈ S_d}; p ← S(p) = s なる p ∈ S_d; S_d ← S_d \ {p}
8:          if s を割り切る S の元がない then
9:              if LM(p) = min{LM(mg) | g ∈ G, m ∈ M, mS(g) = s} then
10:                 if T(Spoly(p)) ⊄ T then
11:                     (R_p, T_p) ← SymbolicPreprocSig({p}, G)
12:                     R ← merge(R,R_p); T ← merge(T,T_p); VR ← PtoV(R, T)
13:                 r ← RegularReduceV(Spoly(p), VR, T); S(r) ← s_min
14:                 if r ≠ 0 then
15:                     Update(r, G, D); R ← merge(R, {r}); VR ← merge(VR, PtoV({r}, T))
16:                     S_d ← S_d ∪ {p ∈ D | tdeg(S(p)) = d}; D ← D \ S_d
17:                 else S ← S ∪ {s}
18: return G
```

10-12 行が, 単項式リスト T に含まれない項が生じた場合に reducer リスト R と T を作り直す操作である.

Algorithm 5 $SymbolicPreprocSig(S, G)$

$SymbolicPreprocSig(S, G)$
Input : S ペアの集合 S, 多項式集合 G
Output :$P = \{(\mathrm{Spoly}(p), S(p)) \mid p \in S\}$ の元を G に関する regular 簡約で
正規形に簡約できる多項式と signature のペア集合 Red および,
$P \cup Red$ に現れる全ての単項式を並べた列 T

1: $w \leftarrow ()$
2: **for** $p \in S$ **do** $w \leftarrow \mathtt{merge}(w, T(\mathrm{Spoly}(p)))$
3: $s_{min} \leftarrow \min\{S(p) \mid p \in S\}$
4: $Red \leftarrow \emptyset$; $T \leftarrow ()$
5: **while** $w \neq ()$ **do**
6: $g \leftarrow FindReducerSig(\mathrm{TOP}(w), G, s_{min})$
7: **if** $g \neq 0$ **then**
8: $m \leftarrow \frac{\mathrm{TOP}(w)}{\mathrm{LM}(g)}$; $T \leftarrow \mathtt{merge}(T, T(mg))$; $Red \leftarrow Red \cup \{(mg, mS(g)\}$
9: w から先頭を外して T の末尾に追加する
10: **return** (Red,T)

Algorithm 6 $FindReducerSig(t, G, s_{min})$

Input : 単項式 t, 多項式集合 G, 加群単項式 s_{min}
Output :$\mathrm{LM}(g) \mid t$ かつ $mS(g)$ がなるべく小さい $g \in G$ (ない場合は 0)

1: $s \leftarrow 0$; $g_{min} \leftarrow 0$
2: **for** $g \in G$ **do**
3: **if** $\mathrm{LM}(g) \mid t$ **then**
4: $m \leftarrow \frac{t}{\mathrm{LM}(g)}$
5: **if** $mS(g) \prec s_{min}$ **then return** g
6: **if** $s = 0$ or $mS(g) \prec s$ **then**
7: $s = mS(g)$; $g_{min} = g$
8: **return** g_{min}

$FindReducerSig$ (Algorithm 6) は, 3.3.1 節の方針 2 を実行する. 5 行目を削除すると, 方針 1 を実行する. S 多項式を, ベクトルに変換した reducer リストにより regular 簡約を行うのが $RegularReduceV$ (Algorithm 7) である. reducer リストを, 先頭項のインデックスをキーとする疎なベクトルの配列に変換するのが $PtoV$ (Algorithm 8) である.

Algorithm 7 *RegularReduceV*(p, R, T)

Input : 多項式 p, reducer 列 $R = (r_0, \ldots, r_{l-1})$, 単項式列 $T = (t_0, \ldots, t_{l-1})$
Output :p を R で regular 簡約した結果

$p = c_0t_0 + \cdots + c_{l-1}t_{l-1}$ のとき $v \leftarrow (c_0, \ldots, c_{l-1})$
for $i = 0$ to $l - 1$ **do**
    $(r_i, s_i) \leftarrow R_i$
    if $v_i \neq 0$ and $r_i \neq 0$ and $s_i \prec S(p)$ **then**
        $v \leftarrow v - v_i \cdot r_i$ (r_i の非零成分に対してのみ更新)
return $v_0t_0 + \cdots + v_{l-1}t_{l-1}$

Algorithm 8 *PtoV*(R, T)

Input : 多項式と signature のペアの集合 R, 単項式列 $T = (t_0, \ldots, t_{l-1})$
Output :疎なベクトルに変換した reducer 列 $VR = (VR_0, \ldots, VR_{l-1})$

for $i = 0$ to $l - 1$ **do**
    if $\mathrm{LM}(r) = t_i$ なる $(r, s) \in R$ が存在する **then**
        $r/\mathrm{LC}(r) = \sum_{j=1}^{k} c_j t_{i_j}$ とするとき $v \leftarrow ((i_1, c_1), \ldots, (i_k, c_k))$; $VR_i \leftarrow (v, s)$
    else $VR_i \leftarrow (0, 0)$
return VR

4 実験

計算機実験を Risa/Asir 20230315 on MacBookPro M1 Max で行った結果を示す. `nd_sba` (Algorithm 1), `nd_sba_f4` (Algorithm 4), および参考として `nd_f4` (simplify を実装していない F_4) の計算時間を示す. 計算時間は相互簡約なしの時間で単位は秒である. 表 4 が有限体 $\mathbb{F}_{32003}$, 表 2 が有理数体上での計算結果である. 入力イデアルは, ベンチマーク用によく用いられるものである.

`hrandom(n,d,m)` は dense な `n` 変数斉次 `d` 次式 `m` 個で生成されるイデアルである.

4.1 考察

4.1.1 有限体上の結果について

`nd_sba` と `nd_sba_f4` を比較すると, 有限体上では, `nd_sba_f4` が最大 10 倍程度高速である. これは, 項比較が有限体上での計算コストの大きな部分を占めることから期待していた効果である.

いくつかの例に対し, 3.3.1 節で述べた symbolic preprocessing における 2 つの方針の比較も行った. 表 3 において total は全体の計算時間, symb は symbolic preprocessing の計算時間, (find) は symb のうち, reducer を見つけるのにかかった時間を示す. 方針 1 (下限なし) では, 全体の計算時間が短くなる場合があるが, find が極端に大きくなり, total のかなりの部分を占める場合がある. 一方で方針 2(下限あり) では, 全体の計算時間が方針 1 より劣る場合があるが, find の total に対する割合はそう大きくはない. これらの例から見る限り, 方針 2 をデフォルトにするのが安全のように見える.

	nd_sba	nd_sba_f4	nd_f4
eco12	24	4.3	9.6
eco13	220	28	80
eco14	2300	220	770
katsura11	36	4.4	27
katsura12	330	25	220
katsura13	2900	160	1700
noon9	17	6.3	26
noon10	230	61	320
noon11	4100	690	4500

	nd_sba	nd_sba_f4	nd_f4
dl	0.80	0.80	0.23
extcyc6	2.2	0.80	0.28
f855	2.0	1.7	0.18
filter9	2.9	7.2	0.094
ilias13	6.2	2.7	0.36
redeco12	5.6	1.2	3.2
reimer6	4.6	1.5	0.97

	nd_sba	nd_sba_f4	nd_f4
hrandom(8,4,6)	570	74	230
hrandom(8,4,7)	2300	170	840
hrandom(8,4,8)	3100	220	940
hrandom(9,3,8)	280	28	130
hrandom(9,3,9)	270	26	110
hrandom(10,3,10)	6500	460	2400

表 1: 有限体上での計算

	nd_sba	nd_sba_f4	nd_f4
assur44	4.5	2.1	0.92
chemkin	10	3.1	0.65
cohn3	11	6.3	1.8
cyclic7	6.5	2.0	2.3
discret3	230	210	1700
dl	20	8.9	3.6
eco10	4.0	0.87	1.8
eco11	53	8.2	100
eco12	910	110	380
extcyc6	71	20	8.1
f855	75	36	8.0
fabrice24	8.8	4.9	1.0
filter9	220	96	340
i1	2.8	2.1	3.8
ilias13	9100	2600	46
ilias_k_2	54	17	4.7
ilias_k_3	94	40	7.7

	nd_sba	nd_sba_f4	nd_f4
jcf26	48	21	15
jcf26s	16	8.9	1.9
katsura9	16	2.1	13
katsura10	220	21	150
kin1	2.7	2.3	1.8
kotsireas	1.8	0.59	0.32
mckay	340	250	23
noon8	5.3	2.2	14
noon9	97	22	190
pinchon1	43	16	0.62
rbpl	6.6	1.9	4.2
rbpl24	8.9	4.7	1.1
redcyc7	4.6	1.2	2.6
redeco10	1.2	0.21	0.90
redeco11	8.0	1.2	6.3
redeco12	65	7.5	47
reimer6	67	10	46

表 2: 有理数体上での計算

	方針 2		方針 1	
	total	symb(find)	total	symb(find)
eco13	28	4.0(1.3)	23	4.4(2.9)
eco14	220	23(6.8)	180	26(17)
katsura12	25	3.1(0.59)	22	3.1(1.1)
katsura13	160	14(2.2)	140	15(5.7)
noon9	6.3	1.9(0.93)	9.4	5.1(3.9)
noon10	61	12(6.8)	100	53(46)
noon11	690	100(64)	1300	770(710)

表 3: symbolic preprocessing の比較

4.1.2 有理数体上の結果について

有理数体上でも, やはり `nd_sba_f4` が多くの例で, 2 倍以上高速であるが, これは, 有理数体上での計算は項比較より係数の計算コストが大きな部分を占める場合が多いことから予想外の結果である. 理由として考えられるのは, `nd_sba` が, 最初に見つけた regular 簡約できる reducer をそのまま使うのに対し, `nd_sba_f4` では複数の S 多項式を regular 簡約できるような reducer を探しに行くという点である. `nd_sba` でこれに相当する方法として, 毎回 signature が最小である reducer を求める方法を実装してみた. 表 4 は, 表 2 で用

	最初にみつけた reducer	signature 最小 reducer	`nd_sba_f4`
chemkin	10	5.7	3.1
dl	20	16	8.9
eco11	53	37	8.2
eco12	910	430	110
filter9	220	360	96
ilias13	9100	5300	2600
katsura10	220	140	21

表 4: reducer の見つけ方による `nd_sba` の計算時間の比較

いた例のうち, この比較で大きな差がついたもののみを集めたものであるが, `filter9` のようにかえって遅くなるものや, 高速化するものの, `nd_sba_f4` とは依然として大きな差があるものばかりであり, `nd_sba_f4` の高速性にはまだ他の理由があるようである.

なお, ここで用いた例のうち, `filter9`, `ilias_k_2`, `ilias_k_3` において, ある全次数の処理中に, S 多項式中に新たな単項式が現れて, symbolic preprocessing のやり直しが生じた.

4.2 まとめ

SBA において F_4 スタイルの行列簡約を取り入れた先行研究としては, F4/5 [2] と MatrixF5 [4, 5] がある. 前者は単に, signature を考慮した symbolic preorocessing を行うことのみを提案しており, termination の保証はなく, 正当性についての言及はない. 後者は, 全次数 d の Macaulay 行列を作って regular 簡約に相当する操作を行列簡約として実行する. これは主として計算量評価のしやすさから用いられているが実用

的とは言い難い. ここで述べた方法は, 停止性, 正当性が保証された, 逐次的な SBA の枠組みを崩さず, 各 regular 簡約をベクトル形式で行えるように前処理を追加するものである. 行操作に制限がつき, また, 途中で行, あるいは列が追加されるなど, F_4 における行列簡約と異なる点は多いが, ある種の例に対しては大きな効果があることがわかった.

参 考 文 献

[1] A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5), in Proc. ISSAC2002, 75-83, 2002.

[2] M. Albrecht, J. Perry, F4/5 (2010). https://arxiv.org/abs/1006.4933v2.

[3] A. Arri, J. Perry, The F5 criterion revised, J. Symb. Comp., 46, 1017-1029 (2011). (A revised version : https://arxiv.org/abs/1012.3664v6)

[4] C. Eder, J.-C. Faugère, A survey on signature-based algorithms for comptuing Gröbner bases, J. Symb. Comp., 80, 719-784, 2017.

[5] T. Vaccon, Matrix-F5 algorithms over finite-precision complete discrete valuation fields, J. Symb. Comp., 80, 329-350, 2017.

[6] 野呂正行, 横山和弘, Risa/Asir における signature-based アルゴリズムの実装について, RIMS 講究録 No.2185, 139-148 (2021).