グレブナー基底計算の高速化とその応用

野呂 正行 (神戸大学大学院理学研究科/JST-CREST)

平成 22 年 11 月 29 日

1 はじめに

「グレブナー基底計算の高速化」は, CREST 日比プロジェクト 「現代の産業社会とグレブナー基底の調和」のサブテーマのひとつである. そこでは

- Risa/Asir におけるグレブナー基底計算の高速化を一層促進し, 数学理論と数学ソフトウェアのより緊密な連携を実現する.
- D 加群の積分アルゴリズムの高速実装およびその応用を開拓する.

という目標が掲げられている. グレブナー基底については, 既に [7][23] およびその日本語訳など数多くの入門書が出版され, 学部の授業でも取り上げられ, また, 上記 CREST にも見られるように, 幅広い分野の数学の研究にも応用されるようになっている. また, グレブナー基底を実行できるソフトウェアも数多い. しかし, 単にコマンドを実行すると, 簡単に計算が行き詰まることも多い. このような場合に, ユーザがちょっとした前処理を行うことで, 計算が遂行できる場合がある. 本稿では, グレブナー基底計算の基本から始めて, いくつかの高速化技法を紹介し, それらの応用例についても紹介する. 本稿で述べた工夫により, 少しでも多くの方々に, より実用的にグレブナー基底を使っていただくことができれば幸いである.

2 グレブナー基底の基礎

2.1 記号の準備

$R = K[x] = K[x_1, \ldots, x_n]$ を体 K 上の n 変数多項式環 とする. T を R の係数 1 の単項式全体とするとき, T 上の項順序 $<$ とは T の全順序で次を満たすものをいう.

1. すべての $t \in T$ に対し $1 \leq t$ ($(Y,X \setminus Y)$ に関する消去順序)
2. $t, s, u \in T$, $t < s$ ならば $ut < us$

代表的な項順序として, 各変数の指数を順に比較する辞書式順序, まず全次数を比較する全次数辞書式, 全次数逆辞書式順序, 変数集合 X の部分集合 Y に対し, Y の変数を含む単項式は含まないものより常に順序が上であるような順序 ($(Y,X \setminus Y)$ に関する消去順序) などがある.

項順序 $<$ が与えられれば, 多項式 f の先頭単項式 $\mathrm{LT}(f) = \mathrm{LT}_<(f)$ が, f 中の単項式で, $<$ に関して最大のものとして定義できる. ただし, $\mathrm{LT}(f)$ は変数のベキ積 (係数 1) とする. さらに, $\mathrm{LC}(f)$, $\mathrm{LM}(f)$ をそれぞれ $\mathrm{LT}(f)$ の f における係数, $\mathrm{LM}(f) = \mathrm{LC}(f)\mathrm{LT}(f)$ と定義する. これらを用いて, S-多項式 $\mathrm{Spoly}(f, g)$ を

$$\mathrm{Spoly}(f,g) = \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LM}(f)} f - \frac{\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LM}(g)} g$$

で定義する.

2.2 単項式除算

項順序 $<$ を固定する. 有限集合 $F = \{f_1, \ldots, f_m\} \subset R$ による多項式 g の剰余 r とは

$$\begin{aligned} g_1 &\leftarrow g \\ g_2 &\leftarrow g_1 - m_1 f_{k_1} \quad (m_1 \text{ は単項式, } f_1 \text{ のある項} = m_1 \mathrm{LM}(f_{k_1})) \\ &\cdots \\ g_{s+1} &\leftarrow g_s - m_s f_{k_s} \quad (m_s \text{ は単項式, } f_s \text{ のある項} = m_s \mathrm{LM}(f_{k_s})) \end{aligned}$$

で, g_{s+1} のどの項も, どの $\mathrm{LT}(f_k)$ でも割れないときの g_{s+1} と定める. Dickson の補題により, このような除算 (単項式除算) は有限回で停止することが保証される. 商 m_i を f_k ごとに集めれば

$$g = q_1 f_1 + \cdots + q_m f_m + r, \quad \mathrm{LT}(q_k f_k) \leq \mathrm{LT}(g)$$

なる表示を得る. $r = \mathrm{NF}_F(g)$ と書く. 剰余 r は一般には一意性はない.

2.3 グレブナー基底と Buchberger アルゴリズム

定義 2.1 イデアル $I \subset R$ の有限部分集合 G が $<$ に関するグレブナー基底であるとは, 次がなりたつことをいう.

任意の $f \in I$, $f \neq 0$ に対し, $\mathrm{LT}(g) \mid \mathrm{LT}(f)$ なる $g \in G$ が存在する.

定理 2.2 (Buchberger [1]) 1. 任意の $<$ に対し, I のグレブナー基底 $G_<$ が存在する.

2. $I = \langle G_< \rangle$.

3. $G_<$ が $I = \langle G_< \rangle$ のグレブナー基底 $\Leftrightarrow$ 任意の $f, g \in G_<$ に対し $\mathrm{NF}_{G_<}(\mathrm{Spoly}(f,g)) = 0$.

4. $f \in I \Leftrightarrow \mathrm{NF}_{G_<}(f) = 0$.

この定理により, ただちに次を得る.

アルゴリズム 2.3 (Buchberger アルゴリズム [1])
入力：多項式集合 $F = \{f_1, \cdots, f_l\}$, 項順序 $<$
出力：$\langle F \rangle$ のグレブナー基底 G
$D \leftarrow \{\{f,g\} | f, g \in F; f \neq g\}$; $G \leftarrow F$
while ($D \neq \emptyset$) do
　　$C = \{f,g\} \leftarrow D$ の元; $D \leftarrow D \setminus \{C\}$, $h \leftarrow \mathrm{NF}_G(\mathrm{Spoly}(f,g))$
　　if $h \neq 0$ then $D \leftarrow D \cup \{\{f,h\} | f \in G\}$; $G \leftarrow G \cup \{h\}$
end while
return G

G が I のグレブナー基底とするとき, $f, g \in G$, $\mathrm{LT}(f) | \mathrm{LT}(g)$ ならば $G \setminus \{g\}$ も I のグレブナー基底である. これを用いて冗長元を省いて得られる基底を極小グレブナー基底とよぶ. さらに, 極小グレブナー基底 G の各元 g を, $\mathrm{NF}_{G \setminus \{g\}}(g)$ で置き換えても I のグレブナー基底であるという性質は保たれる. このようにして得られた集合の各元をさらに定数倍して, $\mathrm{LC}(g) = 1$ としたものを簡約グレブナー基底と呼ぶ. 簡約グレブナー基底は集合として一意的である.

2.4 グレブナー基底計算の応用例

グレブナー基底の計算の単純な応用として計算できる例を紹介する.

- イデアル所属判定, イデアル同一性判定

  G を I の任意項順序に関するグレブナー基底とするとき $f \in I \Leftrightarrow \mathrm{NF}_G(f) = 0$.

  $I = J \Leftrightarrow I$ と J の簡約グレブナー基底が一致.

- 根基所属判定

  $f \in \sqrt{I} \Leftrightarrow 1 \in R[t]I + \langle tf - 1\rangle$ が成り立つ. ただし t は新しい変数で, 右辺のイデアルは $R[t] = K[x_1, \ldots, x_n, t]$ で考えたものである.

- イデアルと部分環の交わり (消去イデアル)

  $Y \subset X = x_1, \ldots, x_n$ に対し, $Z = X \setminus Y$, $<$ を (Z, Y) に関する消去順序とするとき, R のイデアル I の, $<$ に関するグレブナー基底を G とすれば $G \cap K[Y]$ は $I \cap K[Y]$ の $<|_{K[Y]}$ に関するグレブナー基底である.

- イデアルの共通部分, イデアル商, saturation

  $I \cap J = (R[t](tI) + R[t]((1-t)J)) \cap R$,

  $I : \langle f\rangle = (I \cap \langle f\rangle)/f, \quad I : \langle f_1, \ldots, f_m\rangle = \cap_{i=1}^m I : \langle f_i\rangle$,

  $I : f^\infty = (R[t]I + \langle tf - 1\rangle) \cap R$.

これらのうち, 任意項順序に関するグレブナー基底計算で済むものは比較的容易であるが, 消去順序に関するグレブナー基底計算は一般により困難であり, 何らかの工夫が必要となる場合も多い. これらとは別に, 根基計算, 根基の素イデアル分解, 準素イデアル分解などの計算は, さまざまな状況におけるグレブナー基底計算を多数繰り返して得られるものであり, 部品となるグレブナー基底計算の高速化が必要となる. 例として, Gianni-Trager-Zacharias [4] による準素分解アルゴリズムを紹介する.

アルゴリズム 2.4 (準素分解 (GTZ [4]))
$Q \leftarrow \emptyset$
while $I \neq R$ do
  $Y \leftarrow I$ に関する極大独立集合 $(\subset X)$
  $IK(Y)[X \setminus Y] = Q_1 \cap \cdots \cap Q_k \quad (Q_i \subset K(Y)[X \setminus Y])$
  と準素分解する (0 次元イデアルの準素分解)
  $Q \leftarrow Q \cup \{Q_1^c, \ldots, Q_k^c\} \quad (Q_i^c = Q_i \cap K[X])$
  $Q_1^c \cap \cdots \cap Q_l^c = I : f^\infty = I : f^s$ を満たす f, s を求める
  $I \leftarrow I + f^s \quad (I = (I : f^s) \cap (I + f^s)$ より)
end do

このアルゴリズムは, 次のようなグレブナー基底計算を含む.

- 0 次元準素分解における $J \cap L[x_i]$ の計算

  この場合, 係数体が一般に有理関数体となり, 有理数体, 有限体などの場合より困難になる.

- Q_i^c の計算

  $X \setminus Y$ を消去する消去順序に関するグレブナー基底から得られるある多項式 h に対する saturation $Q_i : h^\infty$ の計算により得られる.

- f, s の計算

  f は,$X \setminus Y$ を消去する消去順序に関するグレブナー基底から得られる. s は $I : f^\infty$ の計算の副産物として得られる.

このように, 消去順序に関するグレブナー基底計算が多数必要であることが分かる.

3 グレブナー基底計算の高速化

最もシンプルな形の Buchberger アルゴリズム (アルゴリズム 2.3) における計算上の問題点として次の点が挙げられる.

- 剰余が 0 になる S-多項式がたくさん現れる.

  G の元の数だけペアが増えるので, S-多項式はどんどん増えるが, 大部分は剰余が 0 である. 剰余が 0 になる S-多項式は, 出力には寄与しないで, 手間だけが増えることになる.

- S-多項式の選択方法が示されていない.

  計算の手間は, 選択される S-多項式に大きく依存する.

- K が無限体の場合の係数膨張

  結果には現れないような係数膨張が計算途中に生ずる.

これらを克服するために, Buchberger をはじめとして, 多くの研究が行われた.

3.1 高速化技法 0 (古典的なもの)

初期 (60 年代から 90 年代はじめ) に行われた研究成果として次のようなものがある.

- 不要な S-多項式の除去 (criterion)

  ペア (f, g) に対し, $\mathrm{LT}(f), \mathrm{LT}(g)$ のみから, 不要な S-多項式を検出する方法が, Buchberger [2] により提案され, これをシステマティックに行う方法が Gebauer-Möller [3] により提案された.

- S-多項式の選択方法 (selection strategy)

  Buchbeger が提案した選択方法は normal strategy と呼ばれ, $\mathrm{LCM}(\mathrm{LT}(f), \mathrm{LT}(g))$ が $<$ で最小のペア (f, g) を選択する方法である. この方法は, 次数つき項順序についてはうまく働くが, 辞書式順序, 消去順序などの場合は効率が悪い. このような場合に仮想的に斉次化を行い, その次数で最小のものから normal strategy で選択する方法 (sugar strategy [6]) が提案された. これは次数つきでない $<$ に効果がある.

これらの方法はほとんどのグレブナー基底計算ソフトウェアに実装されていると考えられる.

3.2 高速化技法 1 (trace アルゴリズム)

高速化技法 0 で計算しても, 有理数体など無限体上での計算の場合に, 途中の計算に大きい係数が現われて計算が困難になる場合がある. このような場合に, 有限体上での Spoly の剰余が 0 なら, 有理数体上での計算を省くという方法で, グレブナー基底候補を計算し, 得られた結果の正当性を後でチェックするという方法がある. この方法は Traverso [5] により最初に提案され, trace アルゴリズムと呼ばれる. $\phi : \mathbb{Z} \to \mathbb{Z}/\langle p \rangle = \mathbb{F}_p$ を標準的射影とする. 次のアルゴリズムは, グレブナー基底候補を計算する.

アルゴリズム 3.1 ($GBCandidate(F, <, p)$)
入力：$F \subset \mathbb{Z}[x]$, $p \not| \mathrm{LC}(f)$ $(\forall f \in F)$
出力：$G \subset \langle F \rangle$ s.t. $\phi(G)$ が $\langle \phi(F) \rangle$ のグレブナー基底, または **failure**
$D \leftarrow \{\{f, g\} | f, g \in F; f \neq g\}$; $G \leftarrow F$
while ($D \neq \emptyset$) do
  $C = \{f, g\} \leftarrow D$ の元; $D \leftarrow D \setminus \{C\}$
  if $\mathrm{NF}_{\phi(G)}(\mathrm{Spoly}(\phi(f), \phi(g))) \neq 0$ then
    $h \leftarrow \mathrm{NF}_G(\mathrm{Spoly}(f, g))$ (定数倍, 整数係数化)
    if $h \neq 0$ かつ $p \not| \mathrm{LC}(h)$ then $D \leftarrow D \cup \{\{f, h\} | f \in G\}$; $G \leftarrow G \cup \{h\}$
    else return **failure**
  endif
end while
return G

アルゴリズム 3.2 (trace アルゴリズム [5])
do
Again: $p \leftarrow p \not| \mathrm{LC}(f)$ $(\forall f \in F)$ なる未使用の素数
  $G \leftarrow GBCandidate(F, <, p)$
  $G \leftarrow G$ の冗長元を省き相互簡約
  if G is not a Gröbner basis of $\langle G \rangle$ then goto Again
  for each $f \in F$ do
    if $\mathrm{NF}_G(f) \neq 0$ then goto Again
  end for
  return G
end do

trace アルゴリズムは, 0 に簡約されるペアの計算を有限体上の計算により効率よく検出, 除去できる可能性はあるが, 簡約グレブナー基底に現われないような係数の大きい中間基底が生成された場合などに起こる $\mathbb{Z}$ 上での係数膨張には無力である. これは, sugar strategy によるペア選択がうまく機能しない場合などに起こりやすいが, このような場合に, 実際に斉次化すれば挙動がよくなることは知られていた. しかし, 斉次化すると 1 変数増えるので, 別の困難を引き起こす可能性がある. そこで, 斉次化と trace アルゴリズムを融合させることを考えた. すなわち, まず入力を斉次化して trace アルゴリズムの候補生成のみを行い, 非斉次化および相互簡約を行ってからチェックを行う方法である.

アルゴリズム 3.3 (斉次化 trace アルゴリズム [22])
$F^h \leftarrow F$ の斉次化; $<^h \leftarrow <$ の全次数つき斉次化
do
Again: $p \leftarrow$ 未使用の素数
  $G^h \leftarrow GBCandidate(F^h, <, p)$
  $G \leftarrow G^h|_{h=1}$ の冗長元を除いたあと簡約化
  if G is not a Gröbner basis of $\langle G \rangle$ then goto Again
  for each $f \in F$ do
    if $\mathrm{NF}_G(f) \neq 0$ then goto Again
  end for
  return G
end do

3.3 高速化技法 2 (比較的最近のもの)

90 年代以降に提案された高速化技法は, 簡約操作を線形代数におけるガウス消去とみなすことで効率化を行おうとするものが多い. ここでは, J.C. Faugère による F_4 アルゴリズムおよび関連するいくつかのアルゴリズムについて述べる.

3.3.1 頻繁な相互簡約 ([24])

斉次多項式集合 F を入力として, Buchberger アルゴリズムを全次数の小さい S-多項式から処理する場合を考える. D_d を未処理の S-多項式の集合, F_d を次数 d 以下の中間基底とする.

命題 3.4 $D_{d-1} = \emptyset$ となったとき, それ以降 d 次以下の S 多項式は生成されない. さらに, この時点での F_{d-1} は, $\langle F \rangle$ のグレブナー基底のうち $d-1$ 次以下の要素をすべて含む.

命題 3.5 $D_{d-1} = \emptyset$ となった直後の D_d に対し,

$$S_d = \{S(f_i, f_j) \text{ を } F_{d-1} \text{で割った余り} \mid \{i, j\} \in D_d\}$$

とおく. さらに, $R_d = \{f \in R \mid \mathrm{tdeg}(f) = d\}$ の K-基底として全次数 d の単項式全体 $(t_D, \ldots, t_1)$ $(D = \dim_K R_d,\ t_D > t_{D-1} > \cdots > t_1)$ をとり, S_d の元全体にこの基底に関するガウス消去を行って $\mathrm{Span}_K(S_d)$ の基底 S'_d を得たとすれば, S'_d は全次数 d のグレブナー基底の元全体となる.

この命題は, 斉次イデアルのグレブナー基底計算を次数順に行う場合, 既に得られた基底を, 新たに得た基底で簡約して余りに置き換えるなどの操作を自由に行ってよいことを意味する. 実際にこのような操作により, 計算効率が上がることも期待できる. この考えをより一般化すれば, 次項の F_4 アルゴリズムを得る.

3.3.2 F_4 アルゴリズム

J.C. Faugère による F_4 アルゴリズムは, ある全次数の S-多項式をまとめて用意して, これらを簡約するのに十分な reducer を用意し, 行列の掃き出しで簡約することにより, 中間基底をまとめて生成する方法といえる.

アルゴリズム 3.6 (F_4 アルゴリズム [17])
while $D \neq \emptyset$ do
  $S \leftarrow$ 最低次の S 多項式全部; $D \leftarrow D \setminus S$
  $R \leftarrow \mathrm{NF}_{G_{d-1}}(S)$ を計算をするのに十分な reducers $(\{tg \mid g \in G_{d-1}\})$
  $B \leftarrow S \cup R$ に現れる単項式
  $M \leftarrow S \cup R$ を B により行列で表現したもの
  $M' \leftarrow M$ を行基本変形で階段行列に変形したもの
  $T \leftarrow M'$ の行に対応する多項式のうち, 先頭項が R の元の先頭項と一致しないもの全体
  $D \leftarrow \{(g, h) \mid h \in T\}$; $G \leftarrow G \cup T$
end while

F_4 アルゴリズムの実行に必要な reducer の集合は, symbolic preprocessing と呼ばれる次のような方法で生成する.

1. S に現れる単項式を降順に並べてリストにする.

2. リストから最大の元 h を取り出し, $h = t\mathrm{LT}(g)$ なる $g \in G$ があれば, $T(t(g - \mathrm{LM}(g)))$ をリストに加え, tg を R に加える

この方法によれば, 不必要な多項式を追加する可能性はあるものの, S 多項式を簡約するのに十分な多項式集合が得られる. さらに, M の変形に, 行列計算に関する種々の技法を適用できる. 有限体上の場合, 項の比較演算およびリスト演算が大きなコストと占めるため, それらを配列演算に置き換える F_4 アルゴリズムは大変高速となる場合がある. しかし, 有理数体上の場合, 行列の標準形の計算のコストが大きく, 効率化は容易ではない. 中国剰余定理 (CRT) の応用は可能であるが, CRT 版を正直に実装すると, M の標準形の計算だけでなく, 結果のチェックのコストも大きく, Faugère の論文 [17] 後, あちこちで実装実験が行われたが, [17] で報告されたレベルの高速化に成功したとう報告はまだない.

3.3.3 F_4 風 Buchberger アルゴリズム [27]

有理数体上での F_4 アルゴリズムの効率的実装が困難であるため, S-多項式集合を reducer set で単項式除算してから F_4 風行列を作って掃き出しを行う F_4 風の Buchberger アルゴリズムを考案した. これに trace アルゴリズムを応用することもできる. 要点のみ説明する.

- 非 trace 版

  まず $\mathrm{NF}_{G_{d-1}}(S_d)$ を単項除算で計算する. さらに, 得られた剰余を F_4 と同様に行列化して行簡約を行う.

- trace 版

  $\mathbb{F}_p$ 上で F_4 を行い, 基底として残る S_d の元を S'_d とする. すなわち, F_4 アルゴリズムで trace の計算を行う. この S'_d に対して, $\mathbb{Q}$ 上で非 trace 版と同様の計算を行う. こうして得られた基底はグレブナー基底候補なので, 通常の trace アルゴリズムと同じチェックを行う.

3.4 高速化技法 3 (change of ordering)

これまで解説してきたのは, 任意の入力多項式集合で生成されるイデアルのグレブナー基底を計算する方法であったが, 入力多項式集合があらかじめある項順序に関するグレブナー基底である場合, その性質を用いた効率的計算法がいくつか考案されている. 典型的な場合として, ある困難な項順序 $<$ でのグレブナー基底 G を計算する際に, いったん容易な項順序 $<_0$ でのグレブナー基底 G を計算して, そこから出発する場合がある. ここでは, いくつかについて概要を述べるだけにとどめる. 詳細は文献を参照していただきたい.

- FGLM アルゴリズム [10]

  1. 単項式全体からある条件を満たす単項式を除いた集合のうち, $<$ に関して最小のものを見つける.
  2. $\mathrm{NF}_{G_0}(t_i)$ が $\mathrm{NF}_{G_0}(t_0), \ldots, \mathrm{NF}_{G_0}(t_{i-1})$ の線形和で書けたら G の元を得る.

  という手順を繰り返して, G の元をすべて見つけ出す方法である. 2. は未定係数法により線形方程式を解くことに相当する. 0 次元イデアルにのみ適用可能な方法である.

- Modular change of ordering [16]

  有限体上での change of ordering で得た $<$ でのグレブナー基底をテンプレートとして, 未定係数法あるいは trace アルゴリズムによる候補生成により G を求める方法である. この場合, 候補生成が成功すればチェックなしにただちにそれが正しいグレブナー基底であることが保証される.

- Hilbert driven アルゴリズム [13]

  斉次イデアルの場合に, G_0 から計算したヒルベルト関数により, d 次の G の元が全部得られたことが分かったら, 残りの d 次の S-多項式を捨てるという方法である.

- グレブナー walk [15]

  $<_0, <$ の行列表示の先頭 weight ベクトルを結ぶ線分が貫くグレブナー cone におけるグレブナー基底を順に計算していく方法である.

3.5 グレブナー基底の実践的計算法

これまでに述べた高速化技法のうち, 技法 0 はほぼすべての実装で使われていると考えられるが, それ以外については実装によりまちまちである. よって, あらゆる実装について効果のある計算法を述べることは困難であるが, 一般に次のようなことはある程度正しそうである.

- まず有限体上で実験.

  有限体上で計算できないようなら $\mathbb{Q}$ 上ではまず無理であろう.

- 斉次化して計算してみる.

  trace アルゴリズムが実装されていなくても, 斉次化して次数つき項順序でグレブナー基底を計算してから, 非斉次化するという方法で, 直接計算できないグレブナー基底も容易に計算できる場合がある. この場合, weight つき斉次化も有効である.

- 辞書式順序を不用意に使わない

  いくつか変数を消去したいなら, ブロック順序を使うほうが効率よい.

- 手動で change of ordering

  とりあえず, 全次数逆辞書式のグレブナー基底を計算すると, いろいろな手法が使える.

4 高速化技法の応用例

4.1 b 関数の計算

D を n 次元 Weyl 代数 $\mathbb{C}\langle x, \partial_x\rangle$ $(x = (x_1, \ldots, x_n), \partial_x = (\partial_1, \ldots, \partial_n))$ とする. D の左イデアルに対しても Buchberger アルゴリズムが適用可能である.

定義 4.1 ある微分作用素 $L \in D[s]$ に対し $L(x, \partial_x, s)f^{s+1} = b(s)f^s$ を満たすような多項式 $b(s)$ のうちで最小次数のモニックな多項式を $f(x) \in \mathbb{C}[x]$ の大域 b 関数とよび, $b_f(s)$ で表す. $L(x, \partial_x, s)$ の係数を p で解析的とした場合, 局所 b 関数とよび $b_{f,p}(s)$ で表す. $b_f(s)$, $b_{f,p}(s)$ が存在し, 根が負の有理数であることが知られている (Sato, Bernstein, Kashiwara).

定理 4.2 (Oaku[14]) $B_f = \{t-f, \partial_1+\frac{\partial f}{\partial x_1}\partial_t, \ldots, \partial_n+\frac{\partial f}{\partial x_n}\partial_t\}$ とおく. $(-w, w) = (-1, 0, \ldots, 0, 1, 0, \ldots, 0)$ を $(t, x, \partial_t, \partial_x)$ に対する weight, $<_{(-w,w)}$ を, この weight を拡張する単項式順序とするとき, 左 D-ideal $I_f = \langle B_f\rangle$ に対し $\langle b_f(-t\partial_t - 1)\rangle = \mathrm{in}_{(-w,w)}(I_f) \cap \mathbb{C}[t\partial_t]$, ただし $\mathrm{in}_{(-w,w)}(I_f)$ は I_f の各元の weight $(-w, w)$ に関する最大次数を持つ項で生成されるイデアルである.

$\mathrm{in}_{(-w,w)}(I_f)$ は B_f の斉次化のグレブナー計算から計算できる. B_f の斉次化はある項順序に関するグレブナー基底なので, modular change of ordering が適用できる. また, $\langle b_f(-t\partial_t-1)\rangle = \mathrm{in}_{(-w,w)}(I_f)\cap\mathbb{C}[t\partial_t]$ の計算に機械的に消去イデアル計算を適用すると, 一般に大変困難であるが, b 関数の存在が保証されているため, FGLM 的に計算することが可能である. すなわち, $J=\mathrm{in}_{(-w,w)}(I_f)$ の グレブナー基底 G が分かっていれば, 未定係数法が使える.

アルゴリズム 4.3 (未定係数法による $b_f(s)$ の計算 [21])
Input : イデアル J のグレブナー基底 G
Output : $s=t\partial_t$ の J を法とする最小多項式
$i \leftarrow 0$
do
　if $\mathrm{NF}_G(s^i)+\sum_{j=0}^{i-1} a_j \mathrm{NF}_G(s^j)=0$ を満たす $a_{i-1},\dots,a_0\in\mathbb{K}$ が存在する
　then return $s^i+\sum_{j=0}^{i-1} a_j s^j$
　else $i \leftarrow i+1$
end do

4.2 syzygy 計算

多項式ベクトル $F=(f_1,\dots,f_s)$ の syzygy $\mathrm{syz}(F)$ を計算する場合, F がグレブナー基底なら F の S-多項式の F による商から $\mathrm{syz}(F)$ の基底が容易に計算でき, しかもそれはある項順序 (Schreyer 順序) に関するグレブナー基底となっている. しかし, 一般の F については容易ではない. これに関しては 2 つのアルゴリズムが知られている.

アルゴリズム 4.4
Input : $F=(f_1,\dots,f_s)$, $f_i\in R^l$ $(i=1,\dots,s)$
Output : $\mathrm{syz}(F)$ の生成系
$G=(g_1,\dots,g_t) \leftarrow \langle F\rangle$ の $<$ に関する グレブナー基底
$C \leftarrow (t,s)$-行列 s.t. ${}^tG = C\cdot {}^tF$,　$D \leftarrow (s,t)$-行列 s.t. ${}^tF = D\cdot {}^tG$
$S=\{s_1,\dots,s_u\} \leftarrow R^t$ の有限集合 s.t. $\mathrm{syz}(G)=\langle S\rangle$
$\{r_1,\dots,r_s\} \leftarrow I_s - DC$ の行全体, (I_s : s 次単位行列).
return $\langle s_1C,\dots,s_uC,r_1,\dots,r_s\rangle$

アルゴリズム 4.5 (Caboara-Traverso のアルゴリズム)
Input : $F=(f_1,\dots,f_s)$, $f_i\in R^l$, R^l における項順序 $<$
Output: $\mathrm{syz}(F)$ の $(POT,<)$ に関するグレブナー基底 S,
$\langle F\rangle$ の $<$ に関するグレブナー基底 $G=(g_1,\dots,g_t)$,　(t,s)-行列 C s.t. ${}^tG = C\cdot {}^tF$
$(e_1,\dots,e_s) \leftarrow R^s$ の標準基底
$m_i \leftarrow (f_i,e_i)\in R^l\oplus R^s = R^{l+s}$ $(i=1,\dots,s)$
$\tilde{G} \leftarrow \langle m_1,\dots,m_s\rangle$ の $(POT,<)$ に関するグレブナー基底
$S \leftarrow \{h\in R^s \mid (0,h)\in\tilde{G}\}$
$G \leftarrow \{g\in R^l \mid g\neq 0$ かつある $h\in R^s$ に対し $(g,h)\in\tilde{G}\}$
$C \leftarrow (g_i,h_i)\in\tilde{G}$ に対し第 i 行が h_i であるような (t,s)-行列
return (S,G,C)

アルゴリズム 4.4 は f_i が属する自由加群でのグレブナー計算で済むが, 入力からグレブナー基底を生成する行列 C の計算を計算する必要があり, この計算は Buchberger アルゴリズムに更に負荷をかけ

る可能性がある. また得られるのは単なる基底であり, グレブナー基底が必要な場合にはさらに手間がかかる. 一方で, アルゴリズム 4.5 では階数が大きい自由加群での グレブナー基底計算が必要となるが, 通常の Buchberger アルゴリズムの実行結果が得られれば十分であり, syz(F) の グレブナー基底が得られるという利点がある. さらに次の自明な定理が成立する.

定理 4.6 $f_i \in R^l$ $(i = 1, \ldots, s)$, R^s の標準基底 $(e_1, \ldots, e_s)$ に対し $m_i = (e_i, f_i) \in R^s \oplus R^l = R^{s+l}$ とおくと, $M = (m_1, \ldots, m_s)$ は $\langle M \rangle$ の, 任意の POT 順序に関するグレブナー基底である.

すなわち, アルゴリズム中のグレブナー基底計算は change of ordering であり, アルゴリズム 4.5 において $GBCandidate$ が成功すれば, チェックなしにグレブナー基底候補がグレブナー基底となる.

アルゴリズム 4.7 ([25])
Input : $F = (f_1, \ldots, f_s)$, $f_i \in R^l$ かつ f_i は整数係数, R^l における項順序 $<$
Output :syz(F) の $(POT, <)$ に関するグレブナー基底 S
　　$\langle F \rangle$ の $<$ に関するグレブナー基底 $G = (g_1, \ldots, g_t)$,　(t, s)-行列 C s.t. ${}^tG = C \cdot {}^tF$
$(e_1, \ldots, e_s) \leftarrow R^s$ の標準基底
do
restart:　$p \leftarrow ((m_1, \ldots, m_s)$ の $(POT, <)$ に関して permissible な素数
　　$\tilde{G} \leftarrow GBCandidate((m_1, \ldots, m_s), (POT, <), p)$
　　if $\tilde{G} =$ **failure** goto restart
　　$S \leftarrow \{h \in R^s \mid (0, h) \in \tilde{G}\}$
　　$G \leftarrow \{g \in R^l \mid g \neq 0$ かつある $h \in R^s$ に対し $(g, h) \in \tilde{G}\}$
　　$C \leftarrow (g_i, h_i) \in \tilde{G}$ に対し第 i 行が h_i であるような (t, s)-行列
end do
return (S, G, C)

チェックが不要ということから, アルゴリズム 4.7 は, 有理数体上の計算ではほぼ確実にアルゴリズム 4.5 より高速である. [25] にいくつかの例のタイミングデータがある.

4.3　高速化技法だけでは不十分な例 : 準素分解

最後に紹介する例は, 準素分解アルゴリズムである. 準素分解アルゴリズムは 90 年代までに GTZ [4], EHV [8], SY [12] の 3 つのアルゴリズムが提案されているが, いずれを用いるにせよ, さまざまな係数体, 項順序でのグレブナー基底計算を多数必要とする. それぞれに対し, 部品であるグレブナー基底計算の高速化を行うことにより全体の性能を向上させることはできるが, アルゴリズム自体の性質により, それぞれ分解が困難な入力のタイプがあり, グレブナー基底計算の高速化のみでは不十分である. 実際に, これら既存の方法では分解できない例を最近いくつか見つけ, それをきっかけとして新しいアルゴリズムを提案することができた.

アルゴリズム 4.8 (準素分解の新アルゴリズム [26])
Input : イデアル $I_{in} \subset k[X]$
Output : I_{in} の無断のない準素分解
$L_{all} \leftarrow \emptyset$; $Q_{all} \leftarrow k[X]$; $I_t \leftarrow I_{in}$
Restart : $L \leftarrow \emptyset$; $Q \leftarrow k[X]$; $I \leftarrow I_t$; $C = \{0\}$
while $I_t \neq k[X]$ *do* (この点で常に $I_{in} = Q_{all} \cap I$, $I = Q \cap I_t$)
　$L_t \leftarrow I_t$ の孤立準素成分;　$Q_t \leftarrow \bigcap_{J \in L_t} J$

if $Q \subset Q_t$ *goto* **Restart** *else* $Q \leftarrow Q \cap Q_t$
if $Q_{all} \not\subset Q_t$ *then* $\{\ L \leftarrow L \cup L_t;\quad Q_{all} \leftarrow Q_{all} \cap Q_t;\quad L_{all} \leftarrow L_{all} \cup L_t\}$
if $Q_t = I_t$ *or* $Q = I$ *or* $Q_{all} = I_{in}$ *break*
if $I : Q = C$ *goto* **Restart**
else $\{J \leftarrow SeparatingIdeal(I, Q, (I : Q));\ I_t \leftarrow I + J;\quad C \leftarrow I : Q\}$
end do
return $RemoveRedundancy(L_{all})$

アルゴリズム 4.8 は, 入力イデアルの孤立準素成分を最初に計算するという意味で SY (Shimoyama-Yokoyama) アルゴリズムの応用といえるが, 大きな特徴は分離イデアル (separating ideal), すなわちこれまでに見つかった準素成分の交わり $Q = Q_1 \cap \cdots \cap Q_l$ に対し $I = Q \cap (I + J)$ を満たすイデアル J, のうちなるべく (集合として) 大きいものを求めるという点にある. $I = Q \cap (I + J)$ から $J \subset I : Q$ がすぐにわかるので, J は $I : Q$ の生成系から構成するのが自然である. ここで, J を大きくとる理由は, J, すなわち $I + J$ が大きいほど $I + J$ から現われる無駄な準素成分が少なくなるであろう, という単純な発想に基づく. 実際, J に相当するものは GTZ では $J = \langle f^s \rangle$, SY では $J = \langle f_1^{s_1}, \ldots, f_l^{s_l} \rangle$ という形をしているが, GTZ, SY による分解が困難な例では, これらが無駄な成分を生み出している. 分離イデアルの計算法としては発見的な方法しか提案できていないが, ある種の入力についてはかなりの効果を発揮している.

Ideal	Total	孤立成分	埋没成分	M2
I_2	0.9	1	2 (2)	$> 1d$
I_3	17	1	4 (8)	$> 1d$
$A_{2,3,5}$	5	10	9(9)	21h
$A_{2,3,6}$	133	18	23(23)	–
$A_{2,3,7}$	3540	32	56(56)	–
$A_{2,3,8}$	146h	57	131(131)	–
$A_{2,4,4}$	31	15	17(21)	–
$A_{2,4,5}$	12700	35	61(68)	–

表 1: アルゴリズム 4.8 による計算時間 (秒), 準素成分の個数

表は, 局所 b 関数計算に現れたイデアル I_2, I_3 および (m, n) 行列の隣接する k-minor で生成されるイデアル A_{kmn} のいくつかに対し, その準素分解を計算した計算時間, 孤立成分, 埋没成分の個数を表す (例についての詳細は [26] を参照). カッコ内の数は, 無駄な成分を除去する前の埋没成分の個数を表す. 表から, これらの例に関してはアルゴリズム 4.8 は無駄な成分を出しにくいことが分かる. また, M2 は Macaulay2 (SY アルゴリズム) による計算時間である. $A_{2,3,5}$ の計算経過を見ると, 無駄な成分の混入が計算時間の増大を招いているらしいことが分かる.

参考文献

[1] B. Buchberger, An Algorithm for Finding a basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal (German). PhD Thesis, University of Innsbruck, Institute for Mathematics (1965).

[2] B. Buchberger, A Criterion for Detecting Unnecessary Reductions in the Construction of Gróbner bases. In Proc. EUROSAM'79, LNCS 72, Springer-Verlag, 3-21 (1979).

[3] R. Gebauer, H. M. Möller, On an installation of Buchberger's algorithm, Journal of Symbolic Computation 6, 275-286 (1988).

[4] P. Gianni, B. Trager, G. Zacharias, Gröbner basis and primary decomposition of polynomial ideals. *J. Symb. Comp.* **6**, 149-167 (1988).

[5] C. Traverso, Gröbner trace algorithms. In Proc. ISSAC'88, LNCS **358**, Springer-Verlag, 125-138 (1988).

[6] A. Giovini, T. Mora, G. Niesi, L. Robbiano, C. Traverso, " One sugar cube, please " or selection strategies in the Buchberger algorithm, In Proc. ISSAC 1991, ACM Press, 49-54 (1991).

[7] D. Cox, J. Little, D. O'Shea, Ideals, Varieties, and Algorithms. Springer-Verlag (1992).

[8] D. Eisenbud, C. Huneke, W. Vasconcelos, Direct methods for primary decomposition. *Invent. Math.* **110**, 207-235 (1992).

[9] T. Becker, V. Weispfenning, Gröbner Bases, Springer (1993).

[10] Faugère et al., Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *J. Symb. Comp.* 16/4, 329-344 (1993).

[11] W. Adams, P. Loustaunau, An Introduction to Gröbner Bases. Graduate Studies in Mathematics, Vol. 3, AMS (1994).

[12] T. Shimoyama, K. Yokoyama, Localization and Primary Decomposition of Polynomial ideals. *J. Symb. Comp.* **22**, 247-277 (1996).

[13] C. Traverso, Hilbert functions and the Buchberger algorithm. *J. Symb. Comp.* **22**, 355-376 (1996).

[14] T. Oaku, Algorithms for b-Functions, Restrictions, and Algebraic Local Cohomology Groups of D-Modules. *Advances in Applied Mathematics* **19**, 61-105 (1997).

[15] S. Collart, M. Kalkbrener, D. Mall, Converting bases with the Gröbner walk. *J. Symb. Comp.* **24**, 465-469 (1997).

[16] Noro. M., Yokoyama, K., A Modular Method to Compute the Rational Univariate Representation of Zero-Dimen sional Ideals. *J. Symb. Comp.* **28**, 1, 243-263 (1999).

[17] J. C. Faugère, A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra* (139) 1-3, 61-88 (1999).

[18] D. Eisenbud, D. Grayson, M. Stillman, B. Sturmfels (Eds.), Computations in Algebraic Geometry with Macaulay 2. Algorithms and Computation in Mathematics **8**, Springer-Verlag (2000).

[19] M. Kreuzer, L. Robbiano, Computational Commutative Algebra 1. Springer (2000), Computational Commutative Algebra 2. Springer (2005).

[20] G.-M. Greuel, G. Pfister, A Singular Introduction to Commutative Algebra. Springer (2002).

[21] M. Noro, An Efficient Modular Algorithm for Computing the Global b-function. Mathematical Software, Proceedings of the first international congress of mathematical software, Beijing, Edited by A. M. Cohen, X. S. Gao, N. Takayama, World Scientific, 147-157 (2002).

[22] 野呂 正行, 横山 和弘, グレブナー基底の計算 基礎篇, 東京大学出版会 (2003).

[23] D. Cox, J. Little, D. O'Shea, Using Algebraic Geometry. GTM Vol. 185, Springer (2005).

[24] M. Noro, An efficient implementation for computing Groebner bases over algebrai c number fields, In Proc. ICMS2006, LNCS4151, Springer, 99-106 (2006).

[25] M. Noro, Modular Algorithms for Computing a Generating Set of the Sygyzy Module. In Proc. CASC2009, LNCS 5743, 259-268 (2009).

[26] M. Noro, New Algorithms for Computing Primary Decomposition of Polynomial Ideals. In Proc. ICMS2010, LNCS 6327, 233-244 (2010).

[27] M. Noro et al., A computer algebra system Risa/Asir. `http://www.math.kobe-u.ac.jp/Asir`, `http://www.openxm.org` (1994-2010).