数理解析研究所講究録
第1907巻 2014年 122-131

グレブナー基底候補の高速生成法とその検証について

野呂正行
MASAYUKI NORO
神戸大学大学院理学研究科
GRADUATE SCHOOL OF SCIENCE, KOBE UNIVERSITY *

横山 和弘
KAZUHIRO YOKOYAMA
立教大学理学部
FACULTY OF SCIENCE, RIKKYO UNIVERSITY †

Abstract

In [5] we proposed several methods for verifying that a Gröbner basis candidate of an ideal is indeed a Gröbner basis of the ideal. In this article we propose an improved modular algorithm for computing a Gröbner basis candidate. Then we explain the detail of the implementations of these methods and we show some timing data to evaluate these implementations.

1 有理数体上のグレブナー基底候補の高速生成

$\phi_p : \mathbb{Z} \to \mathbb{F}_p$ を標準的射影とする. $\mathbb{Z}_p^0 = \{\frac{b}{a} \mid a, b \in \mathbb{Z}, p \not| a\}$ とする. $\frac{b}{a} \in \mathbb{Z}_p^0$ に対し $\phi_p(\frac{b}{a}) = \frac{\phi_p(b)}{\phi_p(a)}$ により ϕ_p を $\mathbb{Z}_p^0$ に拡張する. $\mathbb{Z}_p^0[X]$ $(X = (x_1, \ldots, x_n))$ 上への拡張も ϕ_p と書く. このとき次が成り立つ.

Theorem 1 $F \subset \mathbb{Z}[X]$, $I = \langle F \rangle$ とし, $G \subset \mathbb{Z}[X]$ を I の簡約グレブナー基底とする. このとき, 有限個の素数 p を除いて $\phi_p(G)$ は $I_p = \langle \phi_p(F) \rangle$ の簡約グレブナー基底となる.

より具体的には, 有理数体上での Buchberger アルゴリズムの実行中に現れるすべての多項式の係数の分母および先頭係数の分子を割らない p に対して, $\phi_p(G)$ は I_p の簡約グレブナー基底となる. この定理より, 有限個の p を除いて $\phi_p(F) \subset \mathbb{F}_p[X]$ の簡約グレブナー基底 G_p は同じ support をもつ多項式からなり, これらを十分たくさん集めて中国剰余定理で結合させれば G が得られることになる. 実際には, G_p が真のグレブナー基底の ϕ_p による像になっているかどうかは分からないため, このようにして得られる多項式集合 G_{can} は G の候補に過ぎないが, 少なくとも次の検証は可能である.

1. G_{can} が $\langle G_{can} \rangle$ のグレブナー基底

   これは G_{can} から作られる S 多項式がすべて G_{can} により 0 に簡約化されることで確認できる.

*noro@math.kobe-u.ac.jp
†kazuhiro@math.kyushu-u.ac.jp

2. $I \subset \langle G_{can} \rangle$

   1. が分かっていれば, これは F の各元が G_{can} により 0 に簡約化されることと同値である.

F が斉次の場合には, 1. から $G_{can} \subset I$ が言える [1]. よって原理的には $G_{can} = G$ を示すことは容易である. 一般には $G_{can} \subset I$ を効率よく示すことは困難であるが, [5] でいくつかの方法を提案した. 本稿の後半でその実行例をいくつか示すが, ここでは G_{can} を効率よく計算するための方法について述べる.

1.1 有限体上でのグレブナー基底計算

次にあげるような工夫があり得る.

1. F_4 アルゴリズムの利用

   G_p の計算は有限体上のグレブナー基底計算のため, F_4 アルゴリズムにおいて係数膨張が起こらず, F_4 の利点である単項式演算の重複の減少の効果が大きく, Buchberger アルゴリズムより効率よくグレブナー基底が計算できることが期待できる.

2. trace 情報の利用

   正しい結果を与える p を選べば, 必要な S 多項式も有理数体上と F_p 上で一致する. よって, 最初の G_p の計算で必要だった S 多項式のリストを保持しておき, 以降の計算ではその S 多項式のみ生成して Buchberger あるいは F_4 アルゴリズムを実行すれば, 無駄な計算が省ける. もちろん, 最初に選んだ p が失敗の場合には, その後の結果がすべて無駄になるが, 最初に複数回完全な実行を行うなどにより, 失敗の確率を小さくすることはできる.

3. 並列化

   各 G_p の計算は完全に独立に行えるため, 並列化は自明である.

1.2 整数-有理数変換

整数-有理数変換とは, 整数 a, M に対し, $a \equiv \frac{b}{c} \bmod M$ ($ac \equiv b \bmod M$) を満たす整数 b, c, ($|b|, |c| < \sqrt{\frac{M}{2}}$) があれば求める, という計算である. これは, a, M に対する拡張ユークリッド互除法を途中で止めることで行うが, M, a が大きくなると一般に手間が多くなる. しかも, M が十分大きくないと失敗するので, 変換すべき係数が多いとこの部分のコストが大きくなる. これについては次のような工夫が考えられる.

1. 一つの多項式の係数がすべて復元できたら, その係数はそのまま保持する

   M に含まれない法 p での値と比較して等しければ, より確からしい結果と考えられる.

2. 分母の候補を掛けてから変換する

   整数-有理数変換で復元すべき分母が小さい程変換の手間が小さい. よって, もし分母 c の大きい因子を含む整数が推測できれば, その数を掛けてから変換すれば手間が小さくて済む. この乗数としては, ある一つの係数を変換してみて, その分母を用いるというのが有力である.

   計算すべき結果から見れば, 分母にあたるのは, 各基底多項式を整数係数とした場合の先頭係数 (あるいはその因数) である. よって, 多項式ごとにこの乗数を計算しなおす方法も考えられるが,

より有効な方法として, 多項式の全次数 (sugar) が一定の間は, 保持している乗数に, 復元して得た分母を掛けていくという方法がある. この根拠は, F_4 アルゴリズムにおける sugar ごとの計算にある. すなわち, F_4 では, 同一 sugar の S 多項式を行列化してガウス消去を行うが, その結果に現れる各行の先頭係数は, 一つの行列から作られた小行列式と考えられ, 相互に関係していると考えられる. このことから, 同一 sugar の多項式を変換する場合には, 上記のように乗数を更新し, sugar が変わったら乗数を 1 に戻す, という方法が有効ではないかと考えられる.

1.3 計算実験

ベンチマーク問題として有名な cyclic-9 (C_9) を用いて各種の実験を行った. C_9^h は C_9 の斉次化である. IR (整数-有理数変換) は各 CRT (中国剰余定理) ステップごとに行っている. 乗数リセットなし

イデアル	乗数リセット	worker 数	計算時間	CRT	IR	有限体の個数
$\langle C_9 \rangle$	なし	62	5.0×10^4	2.4×10^4	1.5×10^4	1736
$\langle C_9 \rangle$	あり	30	2.0×10^4	2400	740	450
$\langle C_9^h \rangle$	なし	64	7.8×10^4	4.5×10^4	1.7×10^4	640
$\langle C_9^h \rangle$	あり	54	2.9×10^4	9000	3000	486

表 1: グレブナー基底候補計算

の場合, 各 CRT ステップごとに, 乗数 1 からリセットなしに更新していく. 乗数リセットありの場合, sugar ごとに乗数を 1 に戻して更新していく. worker 数がそれぞれ違うので, 全体の計算時間は単なる目安だが, CRT および IR は一つの CPU で行っている. 表から, 必要となった有限体の個数が, リセットありの場合明らかに小さくなっている. これが, CRT および IR の計算時間の短縮につながっている.

2 グレブナー基底候補の検証 1 : ヒルベルト関数の応用

イデアル I の簡約グレブナー基底候補 G が $I \subset \langle G \rangle$ を満たすとする.

2.1 斉次の場合

もし I, したがって G が斉次なら次の定理により, $I = \langle G \rangle$ のチェックは原理的にはやさしい.

Definition 1 p を素数とするとき $G \subset \mathbb{Z}[X]$ が $F \subset \mathbb{Z}[X]$ に対する *p-GB candidate* とは, G の各元の先頭係数が p で割れず, $\phi_p(G)$ が $\langle \phi_p(F) \rangle$ のグレブナー基底であることをいう.

Theorem 2 ([1]) 斉次多項式集合 $G \subset \mathbb{Z}[X]$ が $\langle G \rangle$ の簡約グレブナー基底でとする. 斉次多項式集合 $F \subset \mathbb{Z}[X]$ に対し, $\langle F \rangle \subset \langle G \rangle$ かつ G が F に対する *p-GB candidate* ならば $\langle G \rangle = I$, すなわち G は I の簡約グレブナー基底である.

この定理より, CRT によるグレブナー基底候補 G は, G が $\langle G \rangle$ のグレブナー基底であることを確かめれば正しいことが示せる. ただし, あとで示すように, このチェックのコストが候補生成よりはるかに大きくなる例も存在する.

2.2 非斉次の場合

I が非斉次の場合は次の定理がある. 一般に多項式集合 S の各元を斉次化変数 t で斉次化したものを S^h と表す. 項順序 $\prec$ の斉次化も $\prec^h$ と表す.

Theorem 3 ([5]) $F \subset \mathbb{Z}[X]$ とする. $G \subset \mathbb{Z}[X]$ が F の全次数つき順序 $\prec$ に関する *p-GB candidate* で $I = \langle F \rangle \subset \langle G \rangle$ かつ G は $\langle G \rangle$ のグレブナー基底とする. 自然数 k が $\langle \phi_p(F)^h \rangle : t^k = \langle \phi_p(F)^h \rangle : t^\infty$ を満たすとする. このとき, $F^h \cup \{t^k\}$ の $\prec^h$ に関する簡約グレブナー基底 $H \subset \mathbb{Z}[X]$ の各元の先頭係数が p で割れないならば $I = \langle G \rangle$.

すなわち, 有限体上の saturation 計算により得た k を用いて, F^h に t^k を添加した多項式集合のグレブナー基底を計算すれば, $\langle F \rangle$ のグレブナー基底候補の正当性が保証できる.

以上により, 非斉次な F に対する $\langle F \rangle$ のグレブナー基底を, グレブナー基底候補を経由して求める方法としては次の 2 つが考えられる.

- $\langle F^h \rangle$ のグレブナー基底 G を Theorem 2 により求め, G を非斉次化する.

  斉次化することにより変数は一つ増え, また, 非斉次化すると冗長となる基底が多数生成される場合があるので, 一般には候補 G の計算はコストが大きくなる.

- $\langle F \rangle$ の p-GB candidate G を計算する. この p に対し Theorem 3 の k を計算し, $F^h \cup \{t^k\}$ の p-GB canddate が計算できることを確認すれば, G が I のグレブナー基底であることが保証できる.

  $\langle F^h \cup \{t^k\} \rangle$ のグレブナー基底計算は t^k で割れる項を切り捨てた多項式の計算となるため, $\langle F^h \rangle$ のグレブナー基底計算より容易であることが期待できる. また, この入力自身斉次なので, Theorem 2 が適用できる.

2.3 計算実験

[4] で懸案となっている, C_9 のグレブナー基底候補の正当性検証を 2 つの方法

1. $\langle C_9^h \rangle$ のグレブナー基底候補 G ($C_9^h \subset \langle G \rangle$) が, $\langle G \rangle$ のグレブナー基底であることを示す.
2. $\langle C_9^h \cup t^k \rangle$ ($k = 18$) のグレブナー基底候補 G' が, $\langle G' \rangle$ のグレブナー基底であることを示す. ここで, $k = 18$ は, 有限体上の計算により求めた値である.

で行った結果を示す. 計算時間は, worker の計算時間の総和である. 実際には並列計算を行っているので, 1 週間程度で計算できているが, worker により計算時間が最大 2 倍程度差がある. すなわち, 負荷分散に問題がある. この表で見る限り, Theorem 3 に基づく方法の方が効率がよくない. これは, 方法 2 における k が 18 と大変大きいことに由来すると考えられる. より多くの例, とくに k の値が小さくなるような例でも比較を行ってみる必要がある.

方法	worker 数	計算時間
1	26	141 日
2	26	162 日

3 グレブナー基底候補の検証 2：生成関係式を作る

F が非斉次の場合, 前節の方法ではいずれも $\phi_p(F^h)$ のグレブナー基底計算が必要となる. しかし, 場合によっては斉次化すると有限体上でもグレブナー基底計算が極端に困難になる場合がある. このような場合でも, $I = \langle F \rangle \subset \langle G \rangle$ を満たすグレブナー基底候補 G に対し, G の各元が F で生成される, すなわち F の $\mathbb{Q}[X]$ 係数一次結合で書けることを示すことで, $G \subset I$, 従って G が I のグレブナー基底であることを示すことができる. 実際には, G のいくつかの元が F で生成されることを示せば, それらを F に添加して改めて種々のグレブナー基底計算法を試すことで, I のグレブナー基底を計算できる場合もある.

3.1 アルゴリズム

ここでは, 多項式 g を多項式集合 F により生成する生成関係式を求める modular アルゴリズムを与える.

Algorithm 1 (生成関係式の計算)
入力 :多項式集合 $F = \{f_1, \ldots, f_m\}$, $\langle F \rangle$ の, $F \subset \langle G \rangle$ を満たす p−GB candidate $G \subset \mathbb{Z}[X]$, $g \in G$
出力 : $g = h_1 f_1 + \cdots + h_m f_m$ を満たす $h_1, \ldots, h_m \in \mathbb{Q}[X]$ または **failure**
$G_p = \phi_p(G) \leftarrow \langle \phi_p(F) \rangle$ の簡約グレブナー基底
$(H_1, \ldots, H_m) \leftarrow \phi_p(g) = H_1\phi_p(f_1) + \cdots + H_m\phi_p(f_m)$ を満たす $H_i = \sum_j a_{ij} t_{ij}$
　　$(i = 1, \ldots, m,\ a_{ij} \in \mathbb{F}_p,\ t_{ij}$ は単項式$)$
$E \leftarrow (\sum_j c_{1j} t_{1j}) f_1 + \cdots + (\sum_j c_{lj} t_{mj}) f_m - g$ (c_{ij} は未定係数)
$\{e_1, \ldots, e_k\} \leftarrow E$ に現れる項 $t_1, \ldots, t_k$ の係数 (c_{ij} の一次式)
$Z \leftarrow e_1 = \cdots = e_k = 0$ の F_p 上の解における自由変数および定数項がない従属変数
$\bar{E} \leftarrow (\sum_{j, c_{1j} \notin Z} c_{1j} t_{ij}) f_1 + \cdots + (\sum_{j, c_{mj} \notin Z} c_{mj} t_{ij}) f_m - g$
$\{\bar{e}_1, \ldots, \bar{e}_l\} \leftarrow \bar{E}$ に現れる項 $s_1, \ldots, s_m$ の係数 ($c_{ij} \notin Z$ の一次式)
if $\bar{e}_1 = \cdots = \bar{e}_l = 0$ が $\mathbb{Q}$ 上で解 $c_{ij} = b_{ij}$ を持つ then
　　return $(\sum_{j, c_{1j} \notin Z} b_{1j} t_{ij}, \ldots, \sum_{j, c_{mj} \notin Z} b_{mj} t_{ij})$
else return **failure**

このアルゴリズムは, F_p 上で g を F から生成する関係式 $\phi_p(g) = H_1\phi_p(f_1) + \cdots + H_m\phi_p(f_m)$ において H_i の係数を未定係数に置き換えて得られる係数の線形方程式 $e_1 = \cdots = e_k = 0$ をいったん $\mathbb{F}_p$ 上で解いて, 0 にできる未定係数をすべて 0 と置いて改めて得られる線形方程式 $\bar{e}_1 = \cdots = \bar{e}_l = 0$ を $\mathbb{Q}$ 上で解くものである. $e_1 = \cdots = e_k = 0$ は一般には変数の方が多い方程式で, 解は多くの自由変数を持つが, $\bar{e}_1 = \cdots = \bar{e}_l = 0$ は 過剰決定系で有限体上で解が一意となる. $g \in \langle F \rangle$ であったとしても, p で割りきれる係数を 0 としていることからこの方程式が解を持つ保証はない. しかし, $\mathbb{Q}$ 上で解を持てば, それは g を F から生成する係数を与える.

$e_1 = \cdots = e_k = 0$ を直接 $\mathbb{Q}$ 上で解こうとすると, 解が一意でないため, Hensel lifting が使えない. そのため, $\mathbb{F}_p$ 上で 0 にできる係数をすべて 0 にすることで, $\mathbb{F}_p$ 上で解が一意, 従って $\mathbb{Q}$ 上で高々解が 1 つである方程式 $\bar{e}_1 = \cdots = \bar{e}_l = 0$ を強引に作っている. もし, $\mathrm{syz}\phi_p(F)$ のグレブナー基底を求めることができれば, $(H_1, \ldots, H_m)$ をそのグレブナー基底で割った剰余に置き換えることで, $e_1 = \cdots = e_k = 0$ が一意解を持つようにできる. なお, 解が一意でなくても, 多数の有限体上での解の表示を中国剰余定理により結合すれば, $\mathbb{Q}$ 上の解の一般形が求まるが, 解は一組あれば十分なので, 自由変数が多い場合には無駄な計算を多くすることになると考えられる.

3.2 $\mathbb{F}_p$ 上で一意解をもつ線形方程式の $\mathbb{Q}$ 上での求解

$m \times n$ 整数行列 $A = (a_{ij})$ $(m \geq n)$, m 次列ベクトル $b = (b_i)$, n 次未知列ベクトル $x = (x_j)$ に対し $\phi_p(A)x = \phi_p(b)$ が一意解を持つとする. このとき, $Ax = b$ は $\mathbb{Q}$ 上高々一意解を持つ. 解の有無の判定, および解の計算は次のような Hensel lifting で行うことができる. アルゴリズム中で $\mathbb{F}_p = \{0, \ldots, p-1\} \subset \mathbb{Z}$ とみなしている.

Algorithm 2
$A' \leftarrow \det(\phi_p(A')) \neq 0$ なる A の部分 n 次正方行列
$b' \leftarrow b$ から A' と同じ行を取り出した n 次列ベクトル
$x \leftarrow 0$
$r \leftarrow b'$
$k \leftarrow 0$
do
  $y \leftarrow x$ の $\mathrm{mod} p^k$ での整数-有理数変換
  if $y \neq$ **failure** かつ $A'y = b'$ *then*
    if $Ay = b$ *then return* y *else return* **nil**
  $s \leftarrow \phi_p(A')^{-1}\phi_p(r)$
  $x \leftarrow x + p^k s$
  $r \leftarrow \frac{r - A's}{p}$
  $k \leftarrow k + 1$
end do

do の直後において, $A'x \equiv b' \bmod p^k$ かつ $r = \frac{b' - A'x}{p^k}$ が成り立つ. $A'x = b'$ は一意解をもつから有限回で y はこの方程式の解となる. $Ax = b$ の解があるとすれば $x = y$ でなければならないので, もし $Ay \neq b$ でなければ解は存在しない. $z \in \mathbb{F}_p^n$ に対する $\phi_p(A')^{-1}z$ の計算は, $\phi_p(A')^{-1}$ あるいは $\phi_p(A')$ の LU 分解を求めておけば $O(n^2)$ 回の $\mathbb{F}_p$ 演算で計算できる. $A's$ も $O(n^2)$ なので, 解に到達するまでの Hensel lifting の手間は, 解の分母分子の最大桁数を d とすれば $O(dn^2)$ である. 実際には整数-有理数変換の手間がかかるが, 毎回ではなく適当な回数おきに行い, かつ, 分母の候補を掛けてから変換する方法により手間を減らすことができる.

3.3 計算実験

[5] で述べた, Romanovski et al. [2] による例に対する本節の方法の応用について述べる. ここで問題となるイデアル $I = \langle F \rangle$ は, 8 変数で 10 桁程度の係数を持つ非斉次多項式で生成されるイデアルであり, CRT により, $I \subset \langle G \rangle$ を満たすグレブナー基底候補は容易に計算できる. しかし, F を斉次化すると, 有限体上でもグレブナー基底の計算が困難となる[1]. G のすべての元に対し本節の方法を適用するという方法もあり得るが, 事前の実験により, G 中のある 2 つの元を F に添加すれば, 適当な方法 (たとえば斉次化を用いた Buchberger アルゴリズム) によりグレブナー基底が計算できて G と等しくなる. よって, これら 2 つの元

$$g_{60} = 349a_{31}a_{40} + 333b_{31}a_{40} - 333b_{04}a_{13} - 349b_{04}b_{13}$$
$$g_{61} = 555b_{31}a_{40} + 349a_{04}a_{13} - 555b_{04}a_{13} - 349b_{31}b_{40}$$

[1] Risa/Asir では数日たっても結果が得られない.

が I に属することがわかれば十分である. 本節の方法により, $g_{60}, g_{61} \in I$ を示すことができた. これらは support がほぼ同一で, 計算時間などもほぼ同等なので, g_{60} の計算におけるデータサイズ, 計算時間などのみを示す.

行列サイズ (前処理前/後)	計算時間	Hensel lifting	結果の係数サイズ
182300× 440525 / 117585× 116556	2 日 (1CPU)	5030 ステップ	分母分子約 20000 桁

4 Risa/Asir における実装

本稿で述べた方法を効率よく実現するため, Risa/Asir の改良および新機能の実装を行った. それについて述べる. これらのうち, Risa/Asir 本体組み込みの機能は 20140224 版で利用できるが, それらを呼び出すユーザ言語ファイルは 2014 年 2 月現在まだ公開の準備ができていない. マニュアルも用意してなるべく早く公開する予定である.

4.1 グレブナー基底候補の高速生成

- 有限体上でのグレブナー基底計算

  組み込み関数 `nd_f4` により計算を行う. 最初に, オプション `gentrace=1` を指定して, 計算すべき S ペアのリストを含むデータ (NZ とする) を取り出す. 以降の計算では, オプション `trace=`NZ を指定することで, NZ で指定された S ペアのみが生成される. また, オプション `dp=1` を指定することで, 出力は分散多項式のリストとなるため, 中国剰余定理による結合の際に無駄なデータ変換が起こらない.

- 中国剰余定理による結合

  すべてユーザ言語で記述されている. すでに係数全体が整数-有理数変換できた多項式に対しては, それ以上中国剰余定理による結合は行わない.

- 整数-有理数変換

  一つの整数の変換は組み込み関数 `inttorat` で行う. 分母を掛ける, あるいはリセットする等の処理を行うユーザ定義関数が用意されている.

- 並列計算

  `ox_launch` あるいは `ox_launch_nox` により worker として `ox_asir` のプロセス番号リストを渡すことにより, それらを用いた並列計算を行うことができる. この並列計算は完全に独立に行うことができる.

4.2 グレブナー基底候補の検証

- チェックすべき S ペアの生成

  `nd_gr` は, オプション `splist=1` が指定されると, 入力多項式集合に対する, 種々の criteria 適用後の S ペアのリストを返す.

- 指定された S ペアのチェック

  `nd_gr` と `nd_f4` は, オプション `check_splist=S` (S は番号のペアのリスト) を指定すると, それらの S ペアから作られる S 多項式が入力された多項式集合ですべて 0 に簡約されれば 1, されなければ 0 を返す.

- 並列計算

  プロセス番号リストを渡すと, チェックすべき S ペアを適当に分割して, worker に渡す. 各 worker は担当分の S ペアがすべて 0 に簡約されるかどうか調べる. この並列計算も完全に独立に行えるが, ペアの分割の仕方によっては, 計算時間にかなりの差がでる場合がある.

4.3 生成関係式の計算

Algorithm 1 の記号をそのまま用いる.

- 入力多項式からグレブナー基底を生成する行列の計算

  組み込み関数 `nd_btog`$(F, V, Char, Ord, Data)$ により計算する. $Data$ は, `nd_gr`$(F, V, Char, Ord$ `|gentrace=1,gensyz=1)` の出力で, 各中間基底がそれ以前に計算された中間基底からどのように計算されたかを表すデータである. これを入力として, グレブナー基底が F からどのように生成されるかを表す行列を計算する. この組み込み関数自体は $\mathbb{Q}$, 有限体両方に対して実装されているが, 実際には $\mathbb{Q}$ 上での計算は大変非効率である. 有限体上で計算した行列の各列の転置が Algorithm 1 の $(H_1, \ldots, H_m)$ である.

- $e_1, \ldots, e_k$ の計算

  一般に h_i は巨大な多項式で, この計算を単純に多項式の積として行うと膨大な計算時間, メモリを必要とする. 実際には h_i は未定係数多項式なので, 係数間の計算は単なる未定係数の定数倍で, 同じ項の係数の計算は単なるリストへの追加である. よって, 次の手順で行う.

  1. E に現れる項 $t_1, \ldots, t_k$ をまず求める.
  2. $h_i f_i$ の各項がどの t_i の係数になるかを, 指数ベクトルのハッシュ値を用いて高速に探し, リストについかする.

- $e_1, \ldots, e_k$ の係数をファイルに格納する

  Asir の組み込み関数 `put_int` を繰り返し実行して, $\mathbb{F}_p$ 上の疎行列としてファイルに格納する.

- $\mathbb{F}_p$ 上での解の計算

  この計算は C 言語で書かれたプログラムで実行する. 疎行列としてガウス消去する. すなわち各行はガウス消去後も疎なベクトルとして表現される. 上三角化した行列, 行交換情報がファイルに出力される. その後, 別の C プログラムにより 0 にできる要素を 0 にした解をファイルに出力する.

- $\bar{e}_1, \ldots, \bar{e}_l$ の計算

  上で計算した解をもとに, 未定係数多項式を作り直し, $\bar{e}_1, \ldots, \bar{e}_l$ を求める.

- $\bar{e}_1 = \cdots = \bar{e}_l = 0$ の $\mathbb{Q}$ 上での求解

  Algorithm 2 を適用するため, A から A' を抜き出す操作が必要である. これは最初のガウス消去でも得られるはずだが, 現在はもう一度あらたにガウス消去を実行している. そのあと, 抜き出した正方行列に対して改めて有限体上で LU 分解を行っている. これも明らかに非効率的で, 最終的には 1 回のガウス消去ですむようにしたい.

  Hensel lifting は, LU 分解を用いて逆行列を掛ける組み込み関数 `lusolve_main` を呼び出しながら, ユーザ言語で書いたプログラムで実行している.

4.4 F_4 における整数演算の GMP 化

Risa/Asir においては, 多倍長整数演算を自主開発のプログラムで行ってきた. しかし, 整数演算ライブラリである GMP の性能向上はめざましく, GMP を利用している Macaulay2, SINGULAR などに比較して, Risa/Asir におけるほぼ同一の計算 (グレブナー基底計算など) の効率がかなり落ちる場合があることがわかっていた. 今回, GMP 化の効果を測るため, 有理数体上の F_4 における, S 多項式を中間基底で除算する部分にのみ GMP を使用してみた.

Risa/Asir の `nd_f4` においては, sugar が d の S 多項式を次のように処理している.

1. $S_d \leftarrow$ sugar が d の S 多項式
2. $R_d \leftarrow S_d$ の元の, 現在の中間基底 G_{d-1} による剰余
3. $H_d \leftarrow R_d$ が生成する部分空間の基底
4. $G_d \leftarrow G_{d-1} \cup H_d$

ここで, 2. の計算は, S_d の一つの元をベクトルで表示し, 同じくベクトルで表示された G_{d-1} の元の定数倍を繰り返し引くことで行われる. 計算は分母を払いながら行うので, この計算は, 整数ベクトルのスカラー倍の和である. また, 3. の計算は, 有限体上で計算したガウス消去の結果を中国剰余定理で結合し, 適当な間隔で整数-有理数変換を行ったあと, 結果のチェック (これも, ベクトルのスカラー倍の和の繰り返し) を行うものである. これらの計算に現れる多倍長整数演算を, GMP で行うよう変更した. いくつかの例について変更前後の計算時間を示す. この表で分かるように, GMP 化は大きな効果

入力	K_9	C_7	C_8[1)]	$McKay$	$ilias13$
変更前	131	20	540	360	600
変更後	99	16	320	101	250

を発揮する.

5 おわりに

これまで述べたように, アルゴリズムのレベルでの工夫および実装上の工夫, 並列化により, グレブナー基底候補生成および候補の検証が, かなり大規模な問題に対しても実用的な計算時間で可能になってきた. 今後はさらなる効率化のため, 以下のような改良を行う予定である.

---

1) 斉次化 trace アルゴリズムで実行した.

- 生成関係式を作る機能の組み込み化, 並列化

  現状では, ところどころ外部の C プログラムを呼び出す試験実装であり, 各 C プログラムも含めて並列化できていない. これらを組み込み化し, 並列実行も可能にしたい.

- F_4 の並列化

  F_4 のステップ 2, ステップ 3 などは明らかに並列化可能である. これらを並列化した `nd_f4` による直接計算と, 候補生成+検証に計算の効率比較が興味深い.

- 並列グレブナー基底チェックの負荷分散の最適化

  現状では S 多項式のリストを機械的に同程度の個数に分けているが, 各 worker の計算時間に大きな差が出る場合ある. F_4 の利点を生かすためには, ある程度の個数の S 多項式をまとめて worker に渡す必要があるため, 計算が終了した worker に順次, 次の仕事を送るという形はとりにくい. これをどのようにうまく負荷分散するかが検討課題である.

- Risa/Asir の完全 GMP 化

  GMP 化が有効であることが分かったので, Risa/Asir 全体の多倍長整数演算をすべて GMP に置き換えたい.

参 考 文 献

[1] Arnold, E. (2003). Modular algorithms for computing Gröbner bases. *J. Symb. Comp.* **35**, 403-419.

[2] Romanovski, V., Chen, X., Hu, Z. (2007). Linearizability of linear systems perturbed by fifth degree homogeneous polynomials. *J. Phys. A: Math. Theor.* **40**, 22, 5905-5919.

[3] Idrees, N., Pfister, G., Steidel, S. (2011). Parallelization of modular algorithms. *J. Symb. Comp.* **46**, 672-684.

[4] Steidel, S. (2013). Groebner bases of symmetric ideals. *J. Symb. Comp.* **54**, 72-86.

[5] 野呂正行, 横山和弘 (2013). グレブナー基底候補の正当性検証について. 数理解析研究所講究録 **1843**, 38-50.