計算代数入門

野呂 正行 (神戸大学)

0. Introduction

計算代数 (Computer Algebra)

主として $\mathbf{Z}$, $\mathbf{Q}$ (あるいはより一般の環) 上の多項式を扱うアルゴリズムの研究, 開発

それらを実際に計算機上に実装して計算することを目的とする

1960 年ごろまでは, 計算機の数学応用は数値計算のみ

$\Rightarrow$ LISP があらわれて, 記号(数式)計算が可能になった

ソフトウェア開発上の問題点

残念ながら (今でも) 計算機は代数計算向きに作られていない

整数，有理数の実装

計算機上では，整数は有限の大きさ

32bit ($0 \leq n \leq 2^{32}-1$ または $-2^{31} \leq n \leq 2^{31}-1$,
演算は $\mathrm{mod} 2^{32}$) または 64 bit

$\Rightarrow$ $\mathbf{Z}$ はソフトウェアで実装せざるを得ない (bignum;
最近ではGNU のフリーソフト gmp を使うのが普通)

- $n \in \mathbf{Z}$ は配列で表現

  $n = \pm(b_{l-1}B^{l-1} + \cdots + b_0 B^0)$

  $(B = 2^{32}, 0 \leq b_i \leq 2^{32} - 1)$

- $q \in \mathbf{Q}$ は, $\mathbf{Z}$ へのポインタ二つ (分母分子)

多項式の実装，garbage collection

- $f \in \mathbf{Q}[x]$ は linked list
- $f \in \mathbf{Q}[x_1, \ldots, x_n]$ はより複雑な linked list
- より上位のデータ型は，より複雑な linked list

linked list

いくつかのメモリ領域がポインタでつながっている

メモリの使用方法

行列の数値計算のように，あらかじめ割り付けられない

⇒ 必要になるたびに獲得，不要になったら解放

⇒ 手動では大変なので，通常は garbage collector を使う

効率を意識したアルゴリズム設計

数, 多項式に対する演算： 数学的には自明なものが多い

例 多項式因数分解

できることは分かっている

やり方： 中学生風 or 係数評価が分かっていればしらみつぶし

⇒ 効率を考えれば No

あらゆるアルゴリズムに対して, 効率を意識した設計が必要

⇒ 整数のレベルですでに自明でない (例： GCD)

講義の目標

整数，多項式の四則までは認めて，その上の代数的演算をいくつか解説

1. 多項式 GCD，無平方分解，終結式
2. 多項式因数分解
3. グレブナー基底
4. グレブナー基底の計算法

例を実際に計算して見せる

スライドと黒板の併用 (ゆっくりやった方がよい場合には黒板)

参考書

Knuth, The Art of Computer Programming Vol 2, Addison Wesley

野呂, 計算機代数, Rokko Lectures in Math (pdf が web から入手可能)

野呂, 横山, グレブナー基底の計算 基礎篇, 東大出版会

Cox, Little, O'Shea, Ideals, Varieties and Algorithms, Springer UTM

同, Using Algebraic Geometry, Springer GTM

日比, グレブナー基底, 朝倉書店

丸山, グレブナー基底とその応用

他にもいろいろ (Singular, CoCoA, Macaulay の関係者の書いた本が多数ある)

フリーの数学ソフトウェア

かつては入手, インストールも面倒だったが, 最近は簡単になった

- MAXIMA : 最も古い (Macsyma のフリー化)
- AXIOM : 汎用だが, (少なくとも設計は) 敷居が高い
- Macaulay2, Singular, CoCoA : 代数幾何, 可換環論
- PARI/GP : 数論
- Risa/Asir : いちおう汎用

  多項式因数分解, グレブナー基底計算に最も注力
- Kan/sm1 : 微分作用素環

$\Rightarrow$ 全部 Knoppix/Math に収録

Knoppix/Math について

CD/DVD から起動できる Linux (Live Linux)

濱田龍義氏 (福岡大) を中心とするプロジェクト

debian (Linux distribution の一つ) ベースで, 数学ソフトウェアを満載

数学会, ICM 2006, その他各所で配布

使い方

Windows PC に CD/DVD を入れて再起動 $\Rightarrow$ Windows が入っている HDD とは無関係に起動

作ったファイルはそのままでは消える $\Rightarrow$ USB メモリに保存

不便な点

再起動が必要 (Windows と共存できない) $\Rightarrow$ VMWare の利用

VMware Player

仮想マシンを走らせるフリーソフト ⇒ Windows 上にインストールしておく

VMare 上でのKnoppix/Math

- 方法 1： 空の仮想マシン上で DVD を起動
  Windows の中で使えるが, ファイルの保存は面倒
- 方法 2： Knoppix/Math をインストールした仮想マシンを使う
  仮想マシンは Windows のフォルダとして存在する
  ⇒ それを起動すれば, Windows 上で, 仮想 HDD つきのKnoppix/Math が使える (メモリは 512MB 必要だが快適)

1 日目. 多項式 GCD

K : 体, $R = K[x] \Rightarrow R$ は PID, すなわち

イデアル $I \subset R$ は適当な $f \in I$ により $I = \langle f \rangle$ と書ける.

より詳しく, R においては剰余つき除算ができる. すなわち,

$fg \in R$ $g \neq 0$ に対し, $q, r \in R$ が存在して

$$f = qg + r, \mathsf{deg}(r) < deg(g)$$

$(r = \mathsf{remainder}(f, g))$

$\Rightarrow$ 互除法ができる.

互除法

$f, g \in K[x]$

while $g \neq 0$ do

$r \leftarrow$ remainder(f, g); $f \leftarrow g$; $g \leftarrow r$

end while

return f

効率上の問題点

K が有限体なら問題なし. $K = \mathbf{Q}$, $K = \mathbf{Q}(y) \Rightarrow$ 途中で係数が不必要に大きくなる

対処法

- 有限体利用せず $\Rightarrow$ subresultant GCD
- 有限体利用 $\Rightarrow$ modular GCD

Subresultant GCD

定義 (擬剰余; pseudo-remainder)

R を UFD とする ($R[x]$ で GCD が意味をもつ)

$f, g \in R[x]$, $g \neq 0$, $\deg(f) \geq \deg(g)$ のとき

$$\mathrm{prem}(f, g) = \mathrm{remainder}(\mathrm{lc}(g)^{\deg(f)-\deg(g)+1} f, g)$$

($\mathrm{lc}(g)$ は g の主係数; $\mathrm{prem}(f, g) \in R[x]$)

互除法における remainder を prem で置き換えると, $Q(R)$ の元は現れないが, やはり係数が巨大化

$\Rightarrow$ 実は, prem の結果を割り切る R の元を与えることができる

Subresultant アルゴリズム

$f_1 \leftarrow f,\ f_2 \leftarrow g,\ g_1 \leftarrow 1,\ h_1 \leftarrow 1,\ j \leftarrow 1$

以下, $n_j = \deg(f_j),\ e_j = n_j - n_{j+1}$

do

(1) $f_{j+2} \leftarrow \mathrm{prem}(f_j, f_{j+1})/(g_j \cdot h_j^{e_j})$

$g_{j+1} \leftarrow \mathrm{lc}(f_{j+1})$

(2) $h_{j+1} \leftarrow h_j^{1-e_j} g_{j+1}^{e_j}$

if $f_{j+2} = 0$ return f_{j+1}

end do

Subresultant アルゴリズムの性質

1. $\forall e_j = 1$ の場合

   n_j は 1 ずつ減り, $f_{j+2} = \mathrm{prem}(f_j, f_{j+1})/\mathrm{lc}(f_j)^2$
2. (1), (2) は整除

   f_j の係数, h_j は, f, g の係数を要素とする行列 S (Sylvester 行列) の minor になるように設計されているから
3. $\mathrm{GCD}(f, g) = 1$ ならば最終的には $f_j = \pm \det(S)$
4. 有限体を用いない方法では最良

   $\mathrm{GCD}(f, g) = 1 \Rightarrow$ やはり係数が大きくなる

$\Rightarrow$ 有限体を用いた方法 (modular GCD) について紹介 (黒板で)

２日目．一変数の因数分解（黒板で）

1. 有限体 $\mathbf{F}_p$ 上での既約分解

   Berlekamp, Cantor-Zassenhaus アルゴリズム

2. $\mathbf{Q}$ 上での既約分解

   $\mathbf{F}_p$ 上で作った因子のタネを Hensel lifting $\Rightarrow$ いくつか選んで，$\mathbf{Z}$ 上に引き戻して試し割り

3. 終結式

   f, g が共通零点をもつ条件 $\Rightarrow$ 消去法に使える

   代数拡大上のノルムを与える $\Rightarrow$ $K(\alpha)$ 上の既約分解

4. $K(\alpha)$ 上の既約分解

   ノルムを用いて K 上の分解に帰着

   最小分解体が計算できる

3日目. グレブナー基底

Introduction

体 K 上の n 変数多項式環 $R = K[X] = K[x_1, \ldots, x_n]$ グレブナー基底 (GB) は, R のイデアル I を計算機上で操作する方法を与える.

- $f \in I$ かどうかの判定法 (メンバーシップ)
- I の零点 $V(I)$ の計算
- $\sqrt{I}$ (根基), $I \cap J$, $I : J$, $I : J^\infty$ の計算
- I の分解 ($I = \cap Q_i$, Q_i : 準素; $\sqrt{I} = \cap P_i$, P_i : 素).
- 斉次イデアル I の Hilbert 関数 $H_I(s)$ の計算

内容（黒板で）

1. 項順序および単項式除算による剰余

   典型的な項順序 ： 全次数つき順序，辞書式順序，消去順序

2. グレブナー基底の定義，存在

   Hilbert の基底定理を使ってあっさりと

3. 種々の応用

   R/I の具体的取扱い，support を指定した多項式をイデアル中で探す方法，消去法，イデアルの交わり，イデアル商，saturation，根基，イデアルの分解の基礎

4 日目. グレブナー基底の計算法

定義 (S-多項式) $R = K[X]$, $f, g \in R$ に対し,

$$\mathsf{Spoly}(f,g) = \frac{t}{\mathsf{HM}(f)} \cdot f - \frac{t}{\mathsf{HM}(g)} \cdot g,$$

$$t = \mathsf{LCM}(\mathsf{HT}(f), \mathsf{HT}(g))$$

注意 $\mathsf{Spoly}(f,g)$ は, なるべく小さい次数の単項式を f, g にかけて差をとり, 先頭項がキャンセルするようにしたものである.

定理 (Buchberger) 有限集合 G が $I = \langle G \rangle$ のグレブナー基底 (GB) $\Leftrightarrow \forall g_1, \forall g_2 \in G, \mathsf{Spoly}(g_1, g_2) \xrightarrow[G]{*} 0$

これよりただちに次のアルゴリズムを得る:

Buchberger アルゴリズム (最もシンプルな形)

入力： 多項式集合 $F = \{f_1, \cdots, f_l\}$

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; $D \leftarrow D \setminus \{C\}$

$h \leftarrow \mathsf{NF}_G(\mathsf{Spoly}(f, g))$ の一つ

if $h \neq 0$ then

$D \leftarrow D \cup \{\{f, h\} | f \in G\}$; $G \leftarrow G \cup \{h\}$

end while

return G

シンプルな Buchberger アルゴリズムの特徴

- 停止性

  生成される基底の先頭項が, それまで生成された基底の先頭項で割れない$\Rightarrow$ Dickson の補題 (Hilbert の基底定理) により有限で止まる

- G にどんどん元が付け加わる

  実際には, 冗長な元が多い.

- D にどんどんペアがつけ加わる

  剰余が 0 になる S 多項式が多い.

極小グレブナー基底，簡約グレブナー基底

G が I のグレブナー基底とする.

極小グレブナー基底

$f, g \in G$, $\mathrm{HT}(f)|\mathrm{HT}(g) \Rightarrow G \setminus \{g\}$ も I のGB (by 定義). これを用いて冗長元を省く $\Rightarrow$ 極小 GB

相互簡約

F の各元 f を, $\mathrm{NF}_{F\setminus\{f\}}(f)$ で置き換えること

簡約グレブナー基底

極小 GB G を相互簡約したものも I の GB. さらに定数倍して, $\mathrm{HC}(g) = 1$ としたものを簡約 GB と呼ぶ

簡約 GB は集合として一意的

イデアルを identify できる

実際の計算上での問題点

シンプル Buchberger アルゴリズムで計算できるのは簡単な場合のみ．一般に，

- 全次数つき (graded) 項順序での計算は比較的簡単
- 消去順序での計算は困難 (最悪:辞書式順序)

だが，前者でもすぐに行き詰まる場合が多い

理由: 談話会で述べた (剰余が 0 になる S-多項式，不適切な S-多項式選択，係数膨張 etc.)

⇒ ショートカットが有効． 特に，Risa/Asir で実装され，有用性が実証されているものについて詳しく説明する．

1) trace アルゴリズム

素数 p を 入力 F の各要素 f に対し $p \not| \mathrm{HC}(f)$ となるようにとる.

$\phi : \mathbf{Z} \to \mathbf{Z}/\langle p \rangle = \mathbf{F}_p$ を標準的射影とする.

Buchberger trace アルゴリズム (Traverso)

$\phi(G)$ が $\langle \phi(F) \rangle$ の GB となっているような, $G \subset \langle F \rangle$ または **failure** を返す.

trace アルゴリズム： GB 候補生成

$D \leftarrow \{\{f,g\}|f,g \in F; f \neq g\}; \quad G \leftarrow F$

while ($D \neq \emptyset$) do

  $C = \{f,g\} \leftarrow D$ の元; $\quad D \leftarrow D \setminus \{C\}$

  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 $p \not| \mathrm{HC}(h)$ then

      $D \leftarrow D \cup \{\{f,h\}|f \in G\}; \quad G \leftarrow G \cup \{h\}$

    else return **failure** endif

  endif

end while

return G

trace アルゴリズム： 結果のチェック

$\mathrm{HT}(\phi(g)) = \mathrm{HT}(g)$ $(\forall g \in G)$ かつ

$\phi(\mathrm{NF}_G(f,g)) = \mathrm{NF}_{\phi(G)}(\phi(f),\phi(g)))$ $(\forall f, \forall g \in G)$

$\Rightarrow$ アルゴリズム終了後, $\phi(G)$ は $\langle \phi(F) \rangle$ の GB

$G \subset \langle F \rangle$ は成立. すべきチェックは

1. G が $\langle G \rangle$ の GB か?

   G に対して通常の Buchberger 算法を適用

   ここで “儲け” を失う可能性あり

   $\Rightarrow$ あらかじめ簡約化すれば軽減

2. $F \subset \langle G \rangle$ か?

   $\langle G \rangle$ が GB とわかっている

   $\Rightarrow$ $\mathrm{NF}_G(f) = 0$ $(f \in F)$ を確かめればよい

2) 斉次化 trace アルゴリズム

trace アルゴリズムの限界：

$\mathbb{Z}$ 上での係数膨張には無力(本番の NF 計算は $\mathbb{Z}$ 上で行うので)

一方で, 斉次化すれば, 係数膨張は起きにくい

$\Rightarrow$ 斉次+trace ではどうか (両方のいいところだけをとる)

斉次化 trace アルゴリズム (in Risa/Asir)

1. $F^h \leftarrow F$ の斉次化： $f^h = h^d f(x_1/h, \ldots, x_n/h)$
2. $G^h \leftarrow F^h$ から trace アルゴリズムで候補生成
   check はしない
3. $G \leftarrow G^h|_{h=1}$から冗長元を省き，相互簡約したもの
   $G \subset \langle F \rangle$ は成り立つ.
   F が斉次でなければ，一般に，冗長元がたくさん出る.
4. G に対してチェックを行う
   高確率で G は $\langle F \rangle$ の GB. step 3. で冗長が省かれているのでチェックも楽.

trace アルゴリズム，斉次化の効果の実例（デモ）

- trace vs non-trace

  $filter$9, $fabrice$24

- 斉次化 trace vs 単なる trace

  jcf26, $assur$44, $fabrice$24

3) 入力が斉次イデアルの場合の相互簡約

$S_d :=$ ある時点での全次数 d 次の S-多項式全体

$G_d :=$ d 次までの S_d の処理が終ったあとの中間基底

入力 F が斉次なら

$G_d =$ 最終的な GB のうち, d 次以下の元となっている

(以後現れる基底は全て $d+1$ 次以上だから)

さらに, G_d の先頭項は, 互いに他を割らない

$\Rightarrow$ G_d をあらかじめ相互簡約してよい

これまで現れた S多項式はやはり 0 に簡約される

Q 上だと, 係数に共通因数が生じて, content が取れる

頻繁な相互簡約 (in Risa/Asir, CoCoA)

d 次の処理中に, 適宜相互簡約をしても OK

$\Rightarrow$ Risa/Asir では 6 個基底が出るごとに 1 回相互簡約

$\Rightarrow$ 実際に, 係数膨張が小さくなる例あり

4) F_4 アルゴリズム (J.C. Faugère)

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 $(\{tg \mid g \in G_{d-1}\})$

$T \leftarrow \mathrm{Span}(S \cup R)$ を簡約して得た基底のうち,
先頭項がR にはいらない元

$D \leftarrow \{(g, h) \mid h \in T\}$; $G \leftarrow G \cup T$

end while

F_4 の特徴

- S 多項式をまとめて簡約

  不適当な S 多項式選択の影響を受けにくい

  「頻繁な相互簡約」の変形とも見なせる

- R は多めに取る : symbolic preprocessing

  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, $h = t\mathsf{HT}(g)$ なる $g \in G$ があれば, $T(t(g - \mathsf{HM}(g)))$ をリストに加え, tg を R に加える

Span($S \cup R$) の基底計算のショートカット

F_4 の主要部 = Span($S \cup R$) の 行簡約形 (rref) 計算

$\Rightarrow$ CRT, Hensel が適用可能

CRT による A の rref 計算

1. $A \bmod p_i$ を $\mathbf{F}_{p_i}$ 上で行簡約 $\rightarrow \left(\begin{array}{c|c} I & M_i \\ \hline O & O \end{array}\right)$
2. $M \equiv M_i \bmod p_i$ なる整数行列 M を CRT で計算
3. 整数-有理数変換で有理数行列 $\tilde{M} \equiv M \bmod \Pi p_i$ を計算
4. A の各行が $\left(\begin{array}{c|c} I & \tilde{M} \end{array}\right)$ の行の集合で張られることをチェック

F_4 実装における問題点

- A は 多数の reducer を含み, 巨大化しやすい

  CRT を行う対象が多い

  reducer に対応する部分も CRT する必要があるが, チェック後は捨てられる

- チェックのコストが大

  多数の行の 0 簡約だが, とにかく数が多い

$\Rightarrow$ Faugère の論文 (1999) 後, あちこちで実装実験が行われたが, 誰も成功しない

5) F_4 風味 Buchberger 算法 (in Risa/Asir)

F_4 と Buchberger の hybrid

without trace

$\mathrm{NF}_{G_{d-1}}(S_d)$ を単項簡約で計算 (ここが Buchberger),
剰余を行列化して行簡約 (ここが F_4)

$\Rightarrow$ reducer set R は作らないので行列は小さくなる

でも, $\mathbf{Q}$ 上で簡約すると 0 になる行はまだ残っている

F_4 風味 Buchberger 算法 (in Risa/Asir)

F_4 と Buchberger の hybrid +trace

with trace

$S'_d \leftarrow$ $\mathbf{F}_p$ 上で F_4 を行い, 基底として残る S_d の元

(F_4 による trace)

S'_d に対して, $\mathbf{Q}$ 上で上の計算を行う

$\Rightarrow$ $\mathbf{Q}$ 上で簡約すると 0 になる行はもうない

得られた基底は GB 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

F_4 実装の比較

- F_4 風味 (+trace)

  それなりによい (一般に trace がよいが, 逆の場合もある) が下の 2 つには遠く及ばない (例 $cyclicn$)

- Magma (by A. Steel)

  行簡約を CRT でやるが, チェックなし (バグではなく仕様?)

- SALSA or Maple11 (by Faugère)

  謎の方法でとても高速だがチェックなし (バグではなく仕様?)

問題点

正当性のチェックなしでも, 大抵の場合正しい答えを出す

ソースが公開されていないので, ユーザは信じて使うしかない (一種の偽装)?

6) Modular change of ordering

Change of ordering

直接計算するのが大変な項順序 (辞書式順序など) での GB を, 別の項順序 (全次数辞書式順序など) の GB から計算する.

FGLM, Hilbert driven algorithm, Groebner walk などがある.

Modular Change of ordering

FGLM を modular 化し, さらにそれを一般化

Change of ordering の例： FGLM

I : 0 次元イデアル ($\dim_K K[X]/I < \infty$)

G_0 : 項順序 $<_0$ に関する I の GB

$G \leftarrow \emptyset$; $B \leftarrow \{1\}$; $H \leftarrow \emptyset$

do

$N \leftarrow (x_1 B \cup \cdots \cup x_n B) \setminus H$ の元の倍元

if $N = \emptyset$ return G else $h \leftarrow \min_< N$

if $T(f) \subset (B \cup \{h\})$ なる $f \subset I$ が存在 then

$G \leftarrow G \cup \{f\}$; $H \leftarrow H \cup \{h\}$

else $B \leftarrow B \cup \{h\}$

end do

FGLM の問題点

$T(f) \subset (B \cup \{h\})$ なる $f \subset I$ の存在チェック

行列の簡約を incremental にやっていることになる:

1. $\mathsf{NF}_{G_0}(h_0), \ldots, \mathsf{NF}_{G_0}(h_{i-1})$ からなる行列を行簡約
2. $\mathsf{NF}_{G_0}(h_i)$ をその行列の行で簡約して 0 になったら, 求める f が存在したことになる
3. 0 にならなかったらこの行を行列に追加

$\Rightarrow$ H の元を得るまでに, 行列の係数が巨大化していく

$\Rightarrow$ ショートカット (modular change of ordering)

Compatibility

R : 整域, K : 体, $\phi : R \to K$: 全射準同型

$P = \mathrm{Ker}(\phi)$: 極大イデアル, ϕ : R_P まで拡張

定義

$F \subset R[X]$, $\langle F \rangle \subset Q(R)[X]$.

1. ϕ が $(F, <)$ について permissible $\Leftrightarrow \phi(\mathrm{HC}(f)) \neq 0$ $(\forall f \in F)$ (項順序依存)
2. ϕ が F と compatible $\Leftrightarrow \langle \phi(F) \rangle = \phi(\langle F \rangle \cap R[X])$

   $\langle \phi(F) \rangle \subset \phi(\langle F \rangle \cap R[X])$ は常に成り立つ

   項順序非依存だがチェックしずらい !!

Compatibility は, 任意の GB で判定可能

定理

$G \subset \langle F \rangle \cap R[X]$ が $\langle F \rangle$ の $<$ に関する GB とする.
もし ϕ $(G, <)$ と permissible で $\phi(G) \subset \langle \phi(F) \rangle$ ならば

1. $\phi(G)$ は $\langle \phi(F) \rangle$ の GB.

2. ϕ は F と compatible.

Compatible な GB 候補

定義

$G \subset \langle F \rangle \cap R[X]$ が $\langle F \rangle$ の, $<$ に関する (ϕ, F)-compatible な GB 候補$\Leftrightarrow$ ϕ が $(G, <)$ につき permissible で $\phi(G)$ が $\langle \phi(F) \rangle$ の GB.

定理

R が PID で

1. ϕ が F と compatible,

2. G が $I = \langle F \rangle$ の (ϕ, F)-compatible GB 候補

ならば G は I の GB.

Modular change of ordering (骨組)

$\langle F \rangle$ の $<$ に関する $\mathbf{Q}$ 上 の GB 計算 ($F \subset \mathbf{Z}[X]$)

$G_0 \leftarrow \langle F \rangle$ の $<_0$ に関する GB

do

$p \leftarrow (G_0, <_0)$ について permissible な未使用の素数

$G \leftarrow \langle G_0 \rangle$ の (ϕ, G_0)-compatible な GB 候補

もし G が存在したら return G

end do

GB 候補 G の探し方

- Buchberger or F_4+trace (チェックなし)
- 未定係数法+modular (modular FGLM; 0 次元でなくても可)

未定係数法による GB 候補計算

$\overline{G} \leftarrow \langle \phi(G_0) \rangle$ の $<_0$ に関する簡約 GB, $G \leftarrow \emptyset$

for each $h \in \overline{G}$ do

$a_t \leftarrow t \in T(h)$ に対応する不定元

$H \leftarrow \sum_{t \in T(h)} a_t \mathsf{NF}_{<_0, G_0}(t) \quad (a_{\mathsf{HT}_<(h)} = 1)$

$C \leftarrow H$ の X に関する係数 (a_t の一次式)

if $\{f = 0 | f \in C\}$ が解 $S_h = \{a_t = c_t\}$ を持つ

then $G \leftarrow G \cup \{ \sum_{t \in T(h)} c_t t \}$

else return **failure**

end do

return G

精密化

定理

1. $F \subset R[X]$ で ϕ は F と compatible.
2. $\overline{G} = \{\overline{g}_1, \ldots, \overline{g}_s\}$ $(\mathsf{HT}(g_1) < \cdots < \mathsf{HT}(g_t))$ が $\langle \phi(F) \rangle$ の簡約 GB.
3. $H = \{h_1, \ldots, h_t\}$ $(\mathsf{HT}(h_1) < \cdots < \mathsf{HT}(h_s))$ が I の簡約 GB.
4. モニックな $g_i \in I \cap R_P[X]$ が存在して $\phi(g_i) = \overline{g_i}$ $(i = 1, \ldots, s', s' \leq s)$

$\Rightarrow g_i = h_i$ $(i = 1, \ldots, s')$.

精密化の応用 : 消去法

精密化の意味

modular template の逆像を順序の下の方から求めていく

$\Rightarrow$ 求まったところまでが真の GB と対応している.

$\Rightarrow$ 消去法にそのまま応用可能

例 最小多項式

$\overline{m}_f(t) \leftarrow F_p$ 上での f の最小多項式

もし $m_f(f) \in I$, $\phi(m_f) = \overline{m}_f$, なる permissible な $m_f(t)$ が見つかれば, $m_f(t)$ が f の最小多項式

Risa/Asir における種々の GB 関連関数

- 多項式操作関数

  dp_ptod(), dp_dtop(), dp_ht(), dp_hc(), dp_rest()

- non-trace GB 計算

  nd_gr(), nd_f4()

- trace GB 計算

  nd_gr_trace(), nd_f4_trace()

- modular change of ordering

  tolex(), tolex_gsl(), minipoly()

- イデアル分解

  primedec(), primadec()

nd_gr(), nd_f4()

nd_gr($Poly$,V,$Char$,$Order$)

trace なし, $\mathbf{Q}$, $\mathbf{F}_p$, $\mathbf{Q}(t_1,\ldots,t_m)$ 上の GB 計算

$Poly = [f_1, f_2, \ldots, f_k]$, f_i : V を変数とする多項式

$V = [x_1, x_2, \ldots, x_n]$: 変数

$Char$: 0 なら $\mathbf{Q}$ 上, 素数 p なら $\mathbf{F}_p$ 上

$Order$ = 0 (全次数逆辞書式), = 1, (全次数辞書式),
= 2 (辞書式), $[[O_1, N_1], [O_2, N_2], \ldots]$ (積順序;
O_i は 0, 1, 2, N_i は変数の個数)

nd_f4($Poly$,V,$Char$,$Order$)

$\mathbf{Q}(t)$ 上の計算ができない以外は同じ

nd_gr_trace(), nd_f4_trace()

nd_gr_trace($Poly$,V,$Homo$,$Char$,$Order$)

$\mathbf{Q}$, あるいは $\mathbf{Q}(t)$ 上の Buchberger trace 算法

$Homo$: 1 なら, 斉次化 trace, 0 なら単なる trace

$Char$: 1 $\Rightarrow$ 成功するまで繰り返し

-1 $\Rightarrow$ システムが選んだ素数で候補生成 (お試し用)

$-p$ $\Rightarrow$ (p : 素数) $\mathbf{F}_p$ 上で候補生成. 失敗したら 0 を返す(modular change of ordering 用)

nd_f4_trace($Poly$,V,$Homo$,$Char$,$Order$)

$\mathbf{Q}$ 上の F_4 風味 Buchberger 算法

minipoly() (in lib/gr)

`load("gr");` が必要

`minipoly`(G,V,$Order$,F,T)

F の $\mathbf{Q}[X]/\langle G\rangle$ における最小多項式 $m(T)$ を計算する.

G は項順序 $Order$ に関する, 0 次元イデアル $\langle G\rangle$ の GB

$T \notin V$ なる T を選ぶ

辞書式順序 GB の順序最小元のみの計算であり, 高速

tolex() (in lib/gr)

`load("gr");` が必要

`tolex(`G`,`V`,`$Order$`,`W`)`

$\langle G \rangle$ の変数順序 W の辞書式順序に関する GB を計算

G は項順序 $Order$ に関する, 0 次元イデアル $\langle G \rangle$ の GB

例 $eco9$ (shape base の例)

$G = \{x_1 - f_1(x_9), \ldots, x_8 - f_8(x_9), f_9(x_9)\}$

$\Rightarrow$ ほぼ解けたと言える形 (0 次元ならほとんどの場合こうなる)

primedec(), primadec() (in lib/primdec)

`load("primdec");` が必要

`primedec(`B`,`V`)`

$\mathbf{Q}$ 上での素分解： $[P_1, \ldots, P_l]$ を返す

P_i は多項式リストで, $\langle P_i \rangle$ は孤立付属素イデアル

$\sqrt{\langle B \rangle} = \cap_i \langle P_i \rangle$

`primadec(`P`,`V`)`

$\mathbf{Q}$ 上での準素分解： $[[Q_1, P_1], \ldots, [Q_l, P_l]]$ を返す

P_i, Q_i は多項式リストで, $\langle Q_i \rangle$ は準素成分

$\langle P_i \rangle = \sqrt{Q_i}$, $\langle B \rangle = \cap_i \langle Q_i \rangle$