`http://www.math.kobe-u.ac.jp/vmkm/2010vm/en/ref.html` から入手できる.

- `lenny-ox`
  Debian lenny x86 をインストールした仮想マシン.
- `lenny64-ox`
  Debian lenny x86_64 をインストールした仮想マシン. 64 bit 環境なので, 大容量メモリを使った計算に適している.
- `lenny-c11`
  `lenny-ox` に M2, SINGULAR, CoCoA, gnuplot, R, polymake, gfan, latte, topcom をインストールした仮想マシン.
- `etch-ox`
  Debian etch x86 をインストールした仮想マシン. etch は lenny の一つ前のリリースであり, 古い環境でなければ動かないソフトのインストール用に用意した.

最初の 4 つは, KNOPPIX/Math の DVD に含まれる iso イメージファイルを使って起動するための仮想マシンである. iso イメージは DVD からなんらかの方法で取り出すか,

`ftp://ftp.math.kobe-u.ac.jp/pub2/knoppix-math-cd/non-free/`

から入手する. 入手した iso イメージを, 仮想マシンの設定メニューから指定して, 仮想マシンを起動する.

最後の 3 つは, OpenXM (Risa/Asir 他) 以外の数学ソフトはインストールされていない. すべて, ユーザが必要なソフトのみをインストールして使うための小さい仮想マシンである. なお, 日本語環境はインストールされていないが, `/opt/to-ja` にあるスクリプト `to-ja.sh` を実行すれば, 日本語 TeX を含む日本語環境が自動的にインストールされる.

3.3 Macaulay2, SINGULAR, CoCoA 上での計算

本節では, 3.1 節で解説したグレブナー基底計算法を, 代数幾何, 可換環論

nosec:efficientbuchberger

のための代表的なソフトウェアである Macaulay2, SINGULAR, CoCoA 上で実際に実行してみる[3)]. これらのうち, Macaulay2, SINGULAR については, 使い勝手の観点から emacs と呼ばれるソフトウェアの中で使うことが推奨されている. emacs については 第 X 章で詳しく解説されているので, それに従って emacs に習熟してから使ってみることをおすすめする.

3.3.1 起動方法, ヘルプ, マニュアル

KNOPPIX/Math の場合, いずれも $\sqrt{x}$ メニュー, または KNOPPIX-Math-Start アイコン, あるいは端末エミュレータから起動する. それぞれいくつか異なるインタフェースをもつのでそれぞれについて説明する.

Macaulay2

Math メニューから起動すると, emacs のバッファ内で Macaulay2 が起動する (図 3.1). 自分で立ち上げた端末エミュレータのシェルからコマンド `M2`

nofig:m2

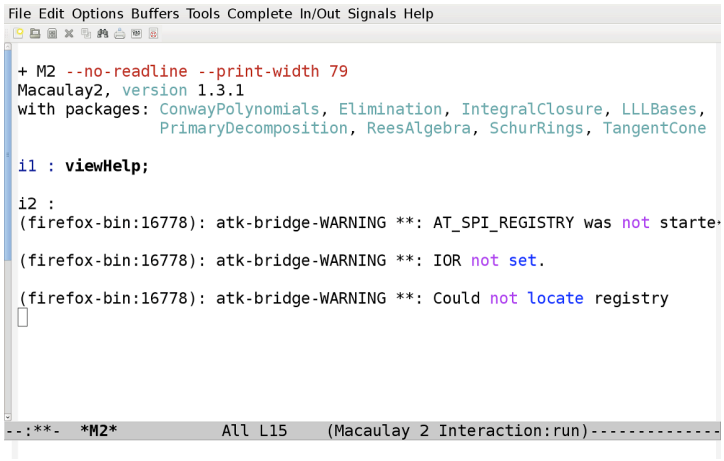

図 **3.1** Macaulay2 nofig:m2

を実行すると, その端末エミュレータ内で Macaulay2 が起動するが, この場

[3)] KNOPPIX/Math 仮想マシン上での実行なので, 計算時間は正確とはいえない.

合, emacs で提供される種々のユーザサポート機能が使えないので, emacs 内で使うことが推奨されている. あるいは, GNU TEXmacs 内での使用も可能である. コマンド `viewHelp` を実行すると, ブラウザが起動する (図 3.2). 最初は,

nofig:m2help

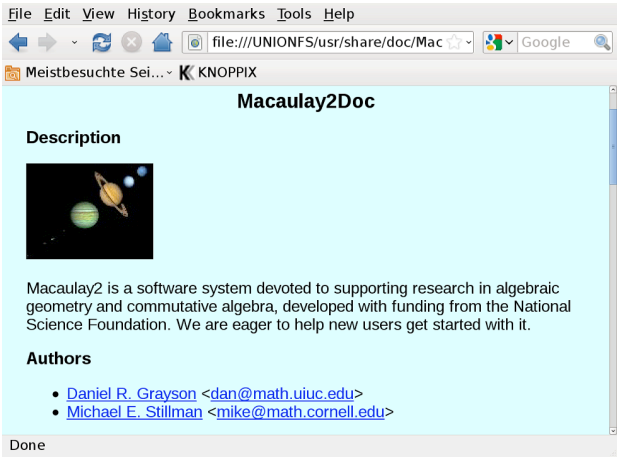

図 **3.2**　Macaulay2 ヘルプブラウザ

nofig:m2help

```
Macaulay 2->getting started->a first Macaulay 2 session
```

などをざっと眺めてみることをお勧めする. 個々のコマンドは, `index` から調べることができる.

SINGULAR

$\sqrt{x}$ メニューから ESingular を起動すると, emacs のバッファ内で SINGULAR が起動する (図 3.3). シェルからコマンド `ESingular` により起動することもできる. ESingular において `help;` を実行すると emacs の info 形式のマニュアルが起動する (図 3.4). 端末ウィンドウ内で `SINGULAR` を実行している場合には, `help;` によりブラウザが起動し, HTML 形式のユーザマニュアルが表示される.

nofig:singular

nofig:singularhelp

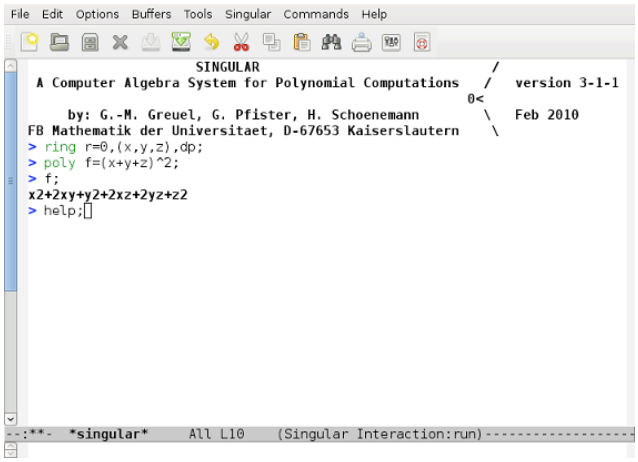

図 3.3 SINGULAR

nofig:singular

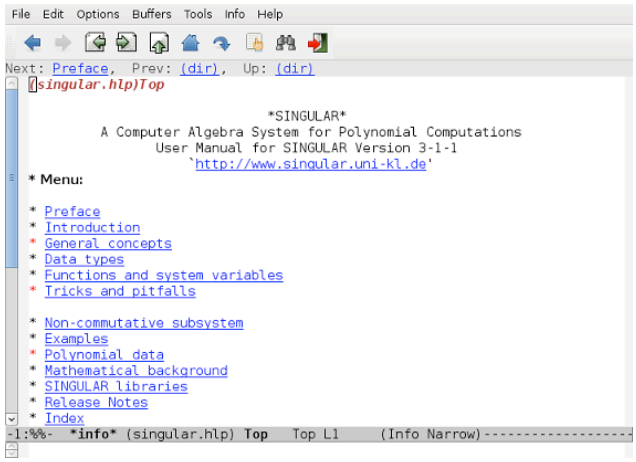

図 3.4 SINGULAR help

nofig:singularhelj

CoCoA

$\sqrt{x}$ メニューから起動すると, 独自の GUI が起動する (図 3.5). 下段に

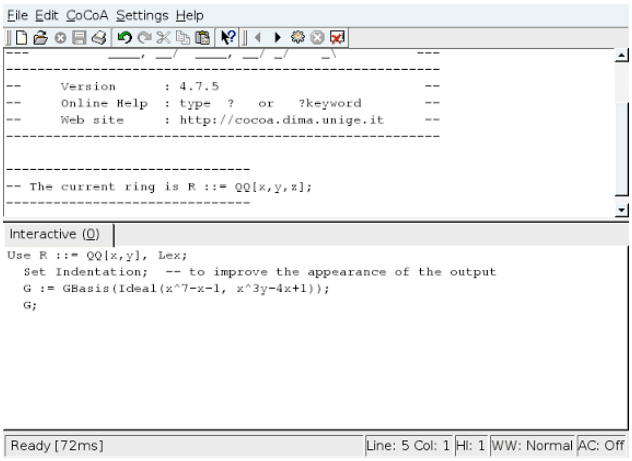

図 **3.5**　CoCoA (xcocoa)

nofig:cocoa

入力し, Ctrl+Enter あるいはツールバーにある実行ボタンを押すと, 実行され, 結果が上段に表示される. シェルからコマンド xcocoa により起動することもできる. なお, CoCoA-4 ではデフォルトでは計算は自分自身が行うが, 計算 (とくにグレブナー基底関連計算) によっては大変遅い場合がある. このような場合, CoCoA-5 のコアとなる CoCoALib の wrapper である CoCoAServer を呼び出して計算を行うことができる. CoCoA-4 の場合, メニューから起動する機能は提供されていないので, CoCoAServer (通常 /usr/local/cocoa-4.7/CoCoAServer) をシェルから直接起動しておく必要がある. GUI の Help->Contents から Online Help が起動でき, それからマニュアルを参照できる (図 3.6).

3.3.2　パッケージ, ライブラリの読み込み, ファイルの読み書き

キーボードから直接コマンドを入れる以外に, あらかじめ作成したファイルを入力とすることができる. 以下で, ファイルとはユーザが作成したファイル, パッケージ (あるいはライブラリ) とは, 各システム付属のプログラム

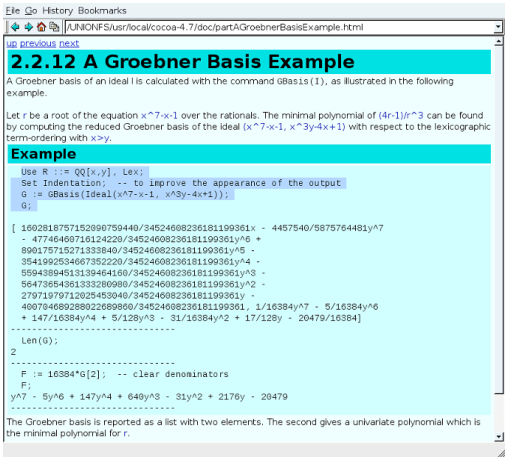

図 **3.6** CoCoA help

nofig:cocoahelp

ファイルのことを指す. ユーザが作成したファイルを読み込む場合, あるいは計算結果をファイルに書き出す場合, ファイルの位置情報を含むパス名を指定する必要がある. 各システムをシェルから起動した場合, 起動時のカレントディレクトリからの相対パスでよい. 例えば, カレントディレクトリにある abc というファイルはそのまま "abc" で, カレントディレクトリのサブディレクトリ pqr にある abc というファイルは "pqr/abc" で指定できる. $\sqrt{x}$ メニューから起動した場合, ユーザのホームディレクトリがカレントディレクトリとなる.

Macaulay2

ファイルの読み込みは load, パッケージの読み込みは loadPackage で行う. ファイルへの書き出しは, ファイル名に対する << によるデータの流し込みの形で行うことができる. 次の例では, out というファイルを新規作成して, リスト L を文字列に変換して書き出し, 改行しファイルを閉じるという操作を行っている.

Macaulay2 : パッケージの読み込み, ファイルの読み書き

```
i1 : loadPackage "Normaliz";
i2 : load "normaliz-example";
i3 : L
              2        2   3     2     2
o3 = {ideal (x , x*y, y , z , x*z , y*z ),...}
o3 : List
i4 : "out"<<toString(L)<<endl<<close;
```

SINGULAR

ファイルの読み込みは `<`, ライブラリの読み込みは `LIB` で行う. 結果のファイルへの書き出しには, ASCII 形式 (`write(":w filename",...)`) と MP 形式 (`write("MPfile:w filename",...)`) の二つの方法がある. 前者の場合, ユーザが可読な形式で書かれるものの, データの構造はすべて捨てられ, カンマ区切りの文字列のリストとなる. 構造ごと保存したい場合には, 後者の形式で保存する. この場合, ユーザ可読ではないが, `read("MPfile:r filename")` で読み込むことができる.

SINGULAR : ライブラリの読み込み, ファイルの読み書き

```
> LIB "primdec.lib";
// ** loaded /usr/share/Singular/LIB/primdec.lib (12508,...)
// ** loaded /usr/share/Singular/LIB/general.lib (12231,...)
> <"primdec-example";
> write(":w asciiout",p);
> write("MPfile:w binaryout",p);
> def pp=read("MPfile:r binaryout");
```

CoCoA

ファイル, パッケージの読み込みは `Source` で行う. インストールディレクトリにあるパッケージを読み込む場合, `CocoaPackagePath()` および `/` をファイル名の前に付ける必要がある. 実際には, CoCoA 付属のパッケージは, その中の関数が呼ばれたときに自動で読み込まれるので, 明示的な読み込みは不要である. GUI の場合には, `File->Open` によりファイルをいった

んバッファに読み込むことができる. 必要があれば編集したあと, キーボード入力の場合と同様に Ctrl+Enter で実行することができる. ファイルへの書き出しは OpenOFile および Print On で行うが, 可読形式で書き出すために, Sprint で文字列に変換してから書き出す.

CoCoA : パッケージの読み込み, ファイルの読み書き

```
Source "gb-example";
G;
D:=OpenOFile("out");
Print Sprint(G) On D;
Close(D);
```

3.3.3 基礎環の宣言, 項順序と多項式の入力

Macaulay2, SINGULAR, CoCoA いずれも, 基礎環を明確に宣言する必要がある. 項順序は基礎環を定義するときに指定する. それぞれのシステムにおいて, 基本的な 3 種類の項順序 : 全次数逆辞書式順序, 純辞書式順序, ブロック項順序 (あるいは消去順序; 3.4.1 節参照) の設定法を説明する.

nosubsec:elimord

Macaulay2

係数体として, 有理数体は QQ, 有限体 $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ は ZZ/p により入力できる. 項順序はデフォルトで全次数逆辞書式順序であり, それ以外の順序を指定する場合には MonomialOrder により指定する.

- 辞書式順序
  例 : QQ[x,y,z,MonomialOrder=>Lex]
  この例は, $x > y > z$ なる辞書式順序を持つ多項式環を宣言している.
- ブロック項順序
  例 : ZZ/37[x,y,z,u,v,MonomialOrder=>{2,3}]
  この例は, $\{x, y\} \gg \{z, u, v\}$ で, 各ブロックに全次数逆辞書式を適用するブロック順序を持つ多項式環を宣言している.

多項式の先頭項は Macaulay2 では lead monomial と呼ばれ, leadMonomial

で取り出せる. その係数は leadCoefficient, 係数つきの先頭項は leadTerm で取り出せる. ある基礎環が設定されている場合, そこに含まれない不定元を使用しようとしても拒否される. 各多項式はそれが生成された時の環に属するため, 異なる環に属する多項式は, たとえ含まれる不定元が一致していても, 一方を他方の環に写す (map を用いる) 必要がある.

Macaulay2 : 基礎環の宣言と多項式の入力

```
i1 : R=QQ[x,y,z];
i2 : f=(x+y+z)^2
      2           2                  2
o2 = x  + 2x*y + y  + 2x*z + 2y*z + z
o2 : R
i3 : g=y+u
stdio:3:4:(1):[0]: error: no method for binary operator
+ applied to objects:
--            y (of class R)
--      +     u (of class Symbol)

i4 : S=QQ[x,y,z,u]
i5 : f+u
stdio:5:2:(1):[0]: error: expected pair to have a method
for '+'
i6 : h=(map(S,R))(f);
i7 : h+u
      2           2                  2
o7 = x  + 2x*y + y  + 2x*z + 2y*z + z  + u
o7 : S
```

Macaulay2 の変数は型宣言は必要なく, 何でも入れられる入れものである.

SINGULAR

基礎環の指定は ring コマンドで行う. 係数体は標数を指定することで指定する. すなわち 0 は有理数体, p (32767 未満の素数) は $\mathbb{F}_p$ を表す. いくつか例を示す.

- 全次数逆辞書式順序
  例： ring r=0,(x,y,z),dp;

- 純辞書式順序
  例： `ring r=0,(a,b,c),lp;`
- ブロック項順序
  例： `ring r=32003,(u,v,w,x),(dp(2),dp(2));`
  `(dp(2),dp(2))` は, $\{u,v\} \gg \{w,x\}$ で, 各ブロックに全次数逆辞書式順序を適用することを意味する.

SINGULAR : 基礎環の宣言と多項式の入力

```
> ring r=0,(x,y,z),dp;
> poly f=(x+y+z)^2;
> poly g=f+u;
   ? ‘u‘ is not defined
   ? error occurred in or before STDIN line 4: ‘poly g=f+u;‘
   ? expected poly-expression. type ’help poly;’
> ring s=0,(x,y,z,u),dp;
> poly h=imap(r,f);
> h+u;
x2+2xy+y2+2xz+2yz+z2+u
```

Macaulay2 と同様, 基礎環の宣言に含まれない不定元を使うとエラーになる. 宣言されていない不定元と演算したい場合には, その不定元を含む新たな環を宣言し, データを `imap` により新しい環に移す必要がある. Macaulay2 と異なり, SINGULAR の変数は型つきで宣言する必要がある.

ubsubsec:cocoaord

CoCoA

基礎環の指定は `Use` コマンドで行う. 有理数体は `QQ`, 有限体は `ZZ/(p)` (p は 32767 以下の素数) である. 項順序はデフォルトでは全次数逆辞書式であり, 他の項順序は以下のように指定する.

- 純辞書式順序
  例： `Use QQ[x,y], Lex;`
- 消去順序
  例： `Use QQ[x,y,z,u], Elim(x..y);`
  この例では, $\{x,y\}$ を消去するようなある消去順序を指定している. 実

際にどのような項順序かは, Ord コマンドにより定義行列を見ればわかる.

CoCoA：基礎環の宣言と多項式の入力

```
Use R::=QQ[x,y,z];
F:=(x+y+z)^2;
F+u;
ERROR: Undefined indeterminate u
CONTEXT: F + u
------------------------------
Use S::=QQ[x,y,z,u];
B:=BringIn(F);
B+u;
x^2 + 2xy + y^2 + 2xz + 2yz + z^2 + u
------------------------------
Use U::=QQ[x[1..5]],Elim(x[1]..x[3]);
Ord(U);
Mat([
  [1, 1, 1, 0, 0],
  [0, 0, 0, 1, 1],
  [0, 0, 0, 0, -1],
  [0, 0, -1, 0, 0],
  [0, -1, 0, 0, 0]
])
```

CoCoA においては, 不定元は小文字 1 文字, あるいはそれに添字を指定したものである. また, 変数名は大文字で始まる. SINGULAR における imap に相当するコマンドは BringIn である. 最後の消去順序の定義行列を見ればわかるように, ブロック順序ではない項順序が消去順序として指定されていることがわかる.

nosubsec:gbcomp

3.3.4　グレブナー基底の計算

イデアルの入力方法およびグレブナー基底計算について述べる. 次の例は以下でしばしば用いる cyclic-7 (C_7) と呼ばれるベンチマーク用のイデアルであり, 特に有理数体上の計算で困難を生じやすい例である. 係数体は適宜有理数体または有限体として用いる.

noex:cyclic7 例 **3.3.1** (cyclic-7)**.**

$$
\begin{aligned}
&C_7 = \langle c_0 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6,\\
&c_0c_1 + c_1c_2 + c_2c_3 + c_3c_4 + c_4c_5 + c_5c_6 + c_6c_0,\\
&c_0c_1c_2 + c_1c_2c_3 + c_2c_3c_4 + c_3c_4c_5 + c_4c_5c_6 + c_5c_6c_0 + c_6c_0c_1,\\
&c_0c_1c_2c_3 + c_1c_2c_3c_4 + c_2c_3c_4c_5 + c_3c_4c_5c_6 + c_4c_5c_6c_0 + c_5c_6c_0c_1 + c_6c_0c_1c_2,\\
&c_0c_1c_2c_3c_4 + c_1c_2c_3c_4c_5 + c_2c_3c_4c_5c_6 + c_3c_4c_5c_6c_0 + c_4c_5c_6c_0c_1 + c_5c_6c_0c_1c_2\\
&+c_6c_0c_1c_2c_3,\\
&c_0c_1c_2c_3c_4c_5 + c_1c_2c_3c_4c_5c_6 + c_2c_3c_4c_5c_6c_0 + c_3c_4c_5c_6c_0c_1 + c_4c_5c_6c_0c_1c_2\\
&+c_5c_6c_0c_1c_2c_3 + c_6c_0c_1c_2c_3c_4,\\
&c_0c_1c_2c_3c_4c_5c_6 - 1\rangle
\end{aligned}
$$

Macaulay2

Macaulay2 でのイデアルの生成は `ideal` による. グレブナー基底は `gb` で行う. この場合, 項順序は環が知っているので引数はイデアル 1 つのみである. 結果はグレブナー基底というオブジェクトで返される. 生成系は `gens` により行列 (行ベクトル) として取り出せる. 次の例は, cyclic-7 の全次数逆辞書式順序によるグレブナー基底計算である. 計算方法の制御を行うためのスイッチが多数用意されてはいるが, 特に指定を行わなくても比較的高速に計算が終了する. `gens` により生成系が 1×209 行列 `g` として得られる. その i 番目の要素は, `g_i` により取り出せる. i は 0 から始まることに注意する.

Macaulay2 : グレブナー基底計算

```
i1 : R=QQ[c0,c1,c2,c3,c4,c5,c6];
i2 : I=ideal(c0+c1+c2+c3+c4+c5+c6,...);
o2 : Ideal of R
i3 : G=gb I;
i4 : g=gens G;
             1       209
o4 : Matrix R  <--- R
i5 : g_0
o5 = | c0+c1+c2+c3+c4+c5+c6 |
       1
o5 : R
```

SINGULAR

SINGULAR では, イデアルは, `ideal` として宣言された変数に, 多項式のリストを代入することで生成する. グレブナー基底計算は `groebner` コマンドで行う. 結果は `ideal` 変数で受けとればよい. イデアルの生成系は変数に添字 (1 から始まる) をつけることによりアクセスできる. 次の例は, cyclic-7 のグレブナー基底を $\mathbb{F}_{32003}$ 上で計算したものである. SINGULAR の場合, 有限体上での計算は高速だが, 有理数体上の場合しばしば計算が進まなくなる. これは不必要な係数膨張によると考えられるが, 例えばこの例の場合, 手動で斉次化を行っても計算が終了するまでに大変時間がかかる.

SINGULAR : グレブナー基底計算

```
> ring r=32003,(c0,c1,c2,c3,c4,c5,c6),dp;
> ideal i=c0+c1+c2+c3+c4+c5+c6,...;
> ideal g=groebner(i);
> g[1];
c0+c1+c2+c3+c4+c5+c6
> g[2];
c1^2+c1*c3-c2*c3+c1*c4-c3*c4+c1*c5-c4*c5+2*c1*c6+...
```

CoCoA

CoCoA では, イデアルは `Ideal` により生成する. グレブナー基底は

GBasis (被約なグレブナー基底は ReducedGBasis) で計算するが, 一般に大変遅いので, 可能なら CoCoAServer を立ち上げて GBasis5 (あるいは ReducedGBasis5) で計算するほうがよい.

CoCoA : CoCoAServer の起動

```
noro@ubuntu:~$ /usr/local/cocoa-4.7/CoCoAServer
------[   Starting CoCoAServer on port 49344 (0xc0c0)   ]------

Provides operations defined in the following libraries:
  CoCoALib-0.9931 (frobby)
  CoCoALib-0.9931 (groebner)
  CoCoALib-0.9931 (combinatorics)
  CoCoALib-0.9931 (approx)
```

GBasis5 においても, 非斉次イデアルのグレブナー基底計算は係数膨張を起こす可能性があり, 実際に例えば cyclic-7 の計算は大変時間がかかる. しかし, 手動で斉次化してから GBasis5 を呼ぶと大変高速に計算できる. 斉次化は Homogenized により行える.

CoCoA : グレブナー基底計算

```
Use QQ[a,b,c,d,e,f,g,t];
I := Ideal(
abcdefg-1,
abcdef+bcdefg+cdefga+defgab+efgabc+fgabcd+gabcde,
abcde+bcdef+cdefg+defga+efgab+fgabc+gabcd,
abcd+bcde+cdef+defg+efga+fgab+gabc,
abc+bcd+cde+def+efg+fga+gab,
ab+bc+cd+de+ef+fg+ga,
a+b+c+d+e+f+g
);
H:=Homogenized(t,Gens(I));
HG:=ReducedGBasis5(Ideal(H));
-- CoCoAServer: computing Cpu Time = 16.9891
DHG := Subst(HG,t,1);
G := ReducedGBasis5(Ideal(DHG));
-- CoCoAServer: computing Cpu Time = 149.265
```

斉次化して得たグレブナー基底 HG に対し, t に 1 を代入して非斉次化すれ

ば (被約とは限らない) I のグレブナー基底 DHG が得られるが, これを改めて ReducedGBasis5 に入力として与えると, 大変時間がかかるものの, I の被約なグレブナー基底を得る.

nosubsec:initial

3.3.5　イニシャルイデアルの計算

イデアル $I \subset R$ ($R = K[x_1, \ldots, x_n]$) のグレブナー基底 G の先頭項から I のイニシャルイデアル $\mathrm{in}_<(I)$ が得られる. ここでは例として, $R = \mathbb{Q}[x, y, z]$ のイデアル I に対し, 全次数逆辞書式順序でのグレブナー基底からイニシャルイデアル $\mathrm{in}_<(I)$ を計算してみる.

Macaulay2

leadTerm によりイニシャルイデアルの生成系が得られる. グレブナー基底は自動的に計算される.

Macaulay2 : イニシャルイデアルの計算

```
i1 : R=QQ[x,y,z];
i2 : I=ideal(x^2*y^2-z^2,x^3-y*z^2,x^2*z^4-y^2);
o2 : Ideal of R
i3 : J=ideal leadTerm I
             3   2 2   3 2   5   6   2 4
o3 = ideal (x , x y , y z , y , z , x z )
o3 : Ideal of R
```

SINGULAR

lead コマンドは, イデアルに対しては単に生成系の先頭項で生成されるイデアルを返すので, イニシャルイデアルを得るには明示的にグレブナー基底を計算してから lead を適用する必要がある.

SINGULAR : イニシャルイデアル計算

```
> ring r=0,(x,y,z),dp;
> ideal i=x^2*y^2-z^2,x^3-y*z^2,x^2*z^4-y^2;
> lead(i);
_[1]=x2y2
_[2]=x3
_[3]=x2z4
> ideal g=groebner(i);
> lead(g);
_[1]=x3
...
_[6]=x2z4
```

CoCoA

LT は, Macaulay2 と同様に, 自動的にグレブナー基底を計算して正しくイニシャルイデアルを計算する.

CoCoA : イニシャルイデアル計算

```
Use R::= QQ[x,y,z];
I:=Ideal(x^2*y^2-z^2,x^3-y*z^2,x^2*z^4-y^2);
LT(I);
Ideal(x^3, x^2y^2, x^2z^4, y^3z^2, z^6, y^5)
```

3.3.6 商および剰余の計算

nosubsec:remainder

剰余計算の最も簡単な応用は, 多項式 f がイデアル I に属するかどうかのテストである. より一般に, 二つのイデアル I, J に対する $I \subset J$ のテストもしばしば必要になる. これは, J の任意項順序でのグレブナー基底 G を計算しておけば, I の生成系の各元の G による剰余が 0 となることを確かめることに帰着される. ここでは例として, 具体的に与えられた多項式 f およびイデアル I に対し, f のあるベキがイデアル I に属することを剰余計算により確かめてみよう. ここで用いる方法では, $f \notin \sqrt{I}$ を示すことはできない. これをグレブナー基底計算一回で行う方法は, 3.4.3 節で解説する.

nosubsec:radicalmember

Macaulay2

Macaulay2 では, 多項式をグレブナー基底または行列 (イデアルの場合, 生成系を並べた行ベクトル) で割った商および剰余が計算できる. 関連する関数は以下の通りである.

- `remainder`(f, g) : f を g で割った剰余 r を返す.
- `quotient`(f, g) : f を g で割った商 q を返す.
- `quotientRemainder`(f, g) : f を g で割った商 q, 剰余 r に対し sequence (q, r) を返す.

引数 f は行列, g はグレブナー基底または行列である. g がグレブナー基底の場合, 商は 0 が返される. g が行列の場合, $gq + r = f$ を満たす q, r が計算される. 例えば g がイデアルの生成系を並べた行ベクトルの場合, $q_0g_0 + \cdots + q_lg_l = f$ を満たす $q_0, \ldots, q_l$ が列ベクトルとして返される.

Macaulay2 : 商および剰余の計算

```
i1 : R=QQ[x,y,z];
i2 : I=ideal(x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4);
o2 : Ideal of R
i3 : G=gb I;
i4 : g=gens G;
             1       3
o4 : Matrix R  <--- R
i5 : f=y*z-x^3;
i6 : remainder(matrix{{f}},G)
o6 = | -x3+yz |
             1       1
o6 : Matrix R  <--- R
i7 : remainder(matrix{{f^2}},G)
o7 = | 2x2y3z+2x3yz+2y2z2+2xz2 |
             1       1
o7 : Matrix R  <--- R
```

Macaulay2 : 商および剰余の計算 (つづき)

```
i8 : remainder(matrix{{f^3}},G)
o8 = 0
             1       1
o8 : Matrix R  <--- R
i9 : qr=quotientRemainder(matrix{{f^3}},g);
o9 : Sequence
i10 : q=qr_0;
              3       1
o10 : Matrix R  <--- R
i11 : g*q
o11 = | -x9+3x6yz-3x3y2z2+y3z3 |
              1       1
o11 : Matrix R  <--- R
i12 : g*q-f^3
o12 = 0
```

イデアル I による f, f^2, f^3 の剰余を計算して, $f^3 \in I$ を示している. f^3 を G で割ることで商が得られる. さらに得られた商 q を G の生成系に掛けることで, $f^3 = gq$ が確かめられる. この場合には $f \notin I$ だが $f^3 \in I$ である.

二つのイデアルが等しいことのチェックもこれらの関数により判定できるが, 演算子 == を使うこともできる. J==I は, 両辺のイデアルのグレブナー基底が自動的に計算され, 両方向の包含関係が判定されるので, 生成系が異なるイデアルの同一性判定に使える. メンバーシップおよびイデアルの包含関係を調べるには isSubset が使える.

Macaulay2 : 包含関係

```
i1 : R=QQ[x,y,z];
i2 : I=ideal(x*y^2-z^2,x^2*z-x^2,y^2*z^2-x^3);
o2 : Ideal of R
i3 : isSubset(ideal(x^2*z-x^2),I)
o3 = true
i4 : J=ideal(y^4,z^2*y^2,z^4,-y^2*x+z^2,z^2*x,x^2);
o4 : Ideal of R
i5 : isSubset(I,J)
o5 = true
```

この例で, 最初の isSubset は $x^2z - x^2$ が I に属するかどうかを調べている. isSubset は引数としてイデアルを要求するのでイデアルとして渡している. 2 つめの呼び出しは $I \subset J$ かどうかのテストである. このように, 包含関係を調べるにはグレブナー基底を明示的に与える必要がないことに注意しておく.

SINGULAR

- reduce(f,I) : f をイデアル I の生成系で割った剰余を返す.
  I の生成系がグレブナー基底でない場合, $f \in I$ でも剰余が 0 にならない場合がある.
- division(f,I) : f をイデアル l で割った商と剰余を返す.
  自動的にグレブナー基底が計算され, グレブナー基底による剰余が計算される. 商は, I の生成系に対するものが計算される. 結果の 3 番目は, well-order とは限らない場合の除算アルゴリズムにおいて f にかけられた乗数を表す. 通常の項順序では 1 である.

SINGULAR : 商および剰余の計算

```
> ring r=0,(x,y,z),dp;
> ideal i=x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4;
> poly f=y*z-x^3;
> reduce(f,i);
// ** i is no standard basis
-x3+yz
> reduce(f^3,i);
// ** i is no standard basis
-x9+3x6yz-3x3y2z2+y3z3
> division(f^3,i);
[1]:
   _[1,1]=x3y2+2x4+y3z
   _[2,1]=-x7+2x3y3z+3x4yz+2y4z2-2x2z2
[2]:
   _[1]=0
[3]:
   _[1,1]=1
```

reduce はイデアルに対しても使えるので, グレブナー基底 I, J に対し reduce(I, J), reduce(J, I) が両方とも 0 のみからなることをチェックすれば $I = J$ がわかる.

CoCoA

- NF(f,I) : f を イデアル I で割った剰余を返す.
  自動的にグレブナー基底が計算され, グレブナー基底による剰余を返すのでメンバーシップ判定ができるが, グレブナー基底計算が困難な場合, 結果を得るのに大変時間がかかる場合がある.
- DivAlg(f,l) : f を多項式リスト l で割った商と剰余を返す.
  商はリストで返される. 多項式リストがグレブナー基底でない場合には, $f \in I$ でも剰余が 0 にならない場合がある.

CoCoA : 商および剰余の計算

```
Use R::=QQ[x,y,z];
I:=Ideal(x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4);
F:=y*z-x^3;
NF(F^3,I);
0
------------------------------
QR:=DivAlg(F^3,Gens(I));
QR;
Record[Quotients := [0, 0],
Remainder := -x^9 + 3x^6yz - 3x^3y^2z^2 + y^3z^3]
------------------------------
QR:=DivAlg(F^3,GBasis(I));
QR;
Record[Quotients := [-4xy^2z^2 - 2x^2z^2, -2xyz, -x^3 - yz],
Remainder := 0]
```

nosec:tools

3.4 グレブナー基底を用いた種々のイデアル操作

本節では, グレブナー基底を用いた種々のイデアル操作のためのアルゴリ

ズムを紹介する. これらの操作は, システムによってはコマンド一つで行える場合もあるが, その方法を知っていれば, 問題に応じて基本機能を独自に組み合わせることにより, 提供されている機能より効率よく目的の計算を行うことも可能である. 以下, $R = K[x_1, \ldots, x_n]$ とする.

3.4.1　消去順序

nosubsec:elimord

以下で述べるイデアル操作は多くの場合, 変数消去を行う必要がある. そのための項順序が消去順序である. 典型的な消去順序として純辞書式順序があるが, 計算効率上その使用は望ましくない. 多くの場合, ブロック順序 (積順序) と呼ばれる項順序で十分である.

定義 3.4.1 (ブロック順序). 不定元集合 Y, Z ($Y \cap Z = \emptyset$) に対し $K[Y]$, $K[Z]$ 上の項順序 $<_Y$, $<_Z$ が与えられているとき, $K[X]$ ($X = Y \cup Z$) 上の項順序 $<$ を, t_Y, s_Y, t_Z, s_Z をそれぞれ $K[Y]$, $K[Z]$ の単項式とするとき

$$t_Y t_Z < s_Y s_Z \Leftrightarrow t_Y <_Y s_Y \text{ または } (t_Y = s_Y \text{かつ } t_Z <_Z s_Z)$$

で定義すれば, $<$ は $K[X]$ 上の消去順序となる. この $<$ を $Y \gg Z$ なるブロック順序 (積順序) と呼ぶ.

I を多項式環 $K[X]$ ($X = Y \cup Z$, $Y \cap Z = \emptyset$) のイデアルとするとき, $I_Z = I \cap K[Z]$ の生成系は, $Y \gg Z$ なる任意の消去順序 $<$ に関する I のグレブナー基底 G に対し $G_Z = G \cap K[Z]$ により与えられる. さらに, G_Z は I_Z の $<|_{K[Z]}$ に関するグレブナー基底になっている. 消去順序としては辞書式順序も使えるが, 計算効率の問題から, 通常はブロック順序 (あるいはそれに準ずる消去順序; 3.3.3 節参照) を使うのが望ましい.

nosubsubsec:cocoaord

例として $\mathbb{Q}[x, y, z]$ のイデアル $I = \langle x^2 - z, xy - 1, x^3 - x^2 y - x^2 - 1 \rangle$ に対し, $I \cap \mathbb{Q}[z]$ の生成系を計算してみよう.

Macaulay2

$\{x, y\} \gg \{z\}$ で, 各ブロックで全次数逆辞書式順序を適用するブロック順序を設定してグレブナー基底 G を計算したあと, 変数が z のみからなる多項式を G から取り出して Gz としている. Gz を求めるには

selectInSubring を用いる. selectInSubring(i, m) は, 行列 m から i 番目 (この場合は $i \geq 1$) までのブロックに属する変数を含まない列のみを取り出した行列を返す.

Macaulay2 : 消去イデアルのグレブナー基底の計算

```
i1 : R=QQ[x,y,z,MonomialOrder=>{2,1}];

i2 : I=ideal(x^2-z,x*y-1,x^3-x^2*y-x^2-1);
o2 : Ideal of R
i3 : G=gens gb I;
             1       3
o3 : Matrix R  <--- R
i4 : Gz=selectInSubring(1,G)
o4 = | z3-3z2-z-1 |
```

SINGULAR

m を, 変数集合 Y に属する変数の積とするとき, eliminate(I,m) により $I \cap K[X \setminus Y]$ の生成系が計算できる. I が属する環は任意の項順序を設定してよい.

SINGULAR : 消去イデアルのグレブナー基底の計算

```
> ring r=0,(x,y,z),dp;
> ideal i=x^2-z,x*y-1,x^3-x^2*y-x^2-1;
> eliminate(i,x*y);
_[1]=z3-3z2-z-1
```

CoCoA

Elim(v,I) により, v で指定される変数を除いた変数集合 Z に対し $I \cap K[Z]$ を計算する. SINGULAR と同様環の項順序は任意でよい. v としては, 一つの変数または変数リスト ($[x, y]$, $x..z$ など) が指定できる.

CoCoA：消去イデアルのグレブナー基底の計算

```
Use R::=QQ[x,y,z];
I:=Ideal(x^2-z,x*y-1,x^3-x^2*y-x^2-1);
Elim(x..y,I);
Ideal(-1/2z^3 + 3/2z^2 + 1/2z + 1/2)
```

いずれのシステムにおいても, 消去順序に関するグレブナー基底計算が行われている. 上の例はグレブナー計算が容易であるが, 実際の計算においては消去順序のグレブナー基底計算が大変困難な場合がある. 対処法はシステムによりさまざまだが, 共通して使える方法として斉次化がある. 例として, $I = \langle 4a^3 + 3cb^2a^2 - a + 2, -3ca^3 + 3c^2a^2 - a + 3cb^4 + c^3, -6a^2 + (-3c^3 - 4)a + 3, (4cb^2 - 1)a^2 + (db + c)a + b^4, 2a^3 - 3a^2 + (-d^2 - 2e)b + 3\rangle$ に対する $I \cap \mathbb{Q}[d, e]$ の計算を CoCoA で行ってみよう. この計算を上で説明した方法で直接行うと, Elim5 を用いても数十分まっても終了しない. そこで, 斉次化を経由して計算してみる.

CoCoA：斉次化による消去イデアルのグレブナー基底の計算

```
Use R::=QQ[a,b,c,d,e,t];
I:=Ideal(4*a^3+3*c*b^2*a^2-a+2,-3*c*a^3+3*c^2*a^2-a+3*c*b^4+c^3,
-6*a^2+(-3*c^3-4)*a+3,(4*c*b^2-1)*a^2+(d*b+c)*a+b^4,
2*a^3-3*a^2+(-d^2-2*e)*b+3);
H:=Homogenized(t,Gens(I));
J:=Ideal(H);
E:=Elim5(a..c,J);
-- CoCoAServer: computing Cpu Time = 34.3644
-------------------------------
DE:=Subst(E,t,1);
G:=ReducedGBasis5(Ideal(DE));
-- CoCoAServer: computing Cpu Time = 0.5099
```

このように 35 秒ほどですべての計算が終了する.

消去順序に関するグレブナー基底は, 変数消去だけでなく, 有理関数体上のグレブナー基底計算にも応用できる.

nothm:ratgb **定理 3.4.2.** $f_1, \ldots, f_l \in K[X, Y]$ が $K(Y)[X]$ のイデアル I の生成系であ

るとする. $J = \langle f_1, \ldots, f_l \rangle \subset K[X,Y]$ とおくとき, $X \gg Y$ なる消去順序 $<$ に関する J のグレブナー基底は, $<|_{K[X]}$ に関する I のグレブナー基底となる.

noprob:ratgb

問題 3.4.3. 定理 3.4.2 (nothm:ratgb) を示せ.

有理関数体上のグレブナー基底計算は, 有理関数体の演算を用いた Buchberger アルゴリズムにより計算できるが, 定理 3.4.2 (nothm:ratgb) により体 K 上の Buchberger アルゴリズムを応用しても計算できる. 有理関数体上の計算は, 約分に多項式の GCD 計算を必要とするため困難を伴う. このため, 後者のほうが効率よく計算できる場合もあるが, 一般に入力イデアルからそれを判断するのは難しい.

3.4.2 イデアルの和, 積, 共通部分

nosubsec:idealsum

R のイデアル $I = \langle A \rangle$, $J = \langle B \rangle$ に対し, 和 $I + J = \langle A \cup B \rangle$, および積 $IJ = \langle \{ab \mid a \in A, b \in B\} \rangle$ については, 少くともそれらの生成系は定義により容易に得られる. しかし, A, B がグレブナー基底であっても, ここで与えた和, 積の生成系はグレブナー基底とは限らない. グレブナー基底を得るには改めてこれらの生成系を入力としてグレブナー基底計算を行う必要があり, それらが容易であるとは限らないことに注意する. 第 1 章で見たように, イデアルの共通部分は消去イデアル計算により得られる. イデアル $I_1, I_2, \ldots$ の共通部分は, Macaulay2 および SINGULAR では `intersect`$(I_1, I_2, \ldots)$ により, CoCoA では `intersection`$(I_1, I_2, \ldots)$ により計算できる. ここでは Macaulay2 の例を示す. この例では, 二項式イデアル `I` の準素イデアル分解 (3.8 節参照 (nosec:M2program)) を計算し, 分解成分の共通部分が `I` に等しいことを確かめている. この例のように, 引数としてイデアルのリストを与えてもよい.

Macaulay2 : イデアルの共通部分

```
i1 :  R=QQ[a,b,c,d,e,f,g,h,i];
i2 : I=ideal(e*a-d*b,f*b-e*c,h*d-g*e,i*e-h*f);
o2 : Ideal of R
i3 : PD=primaryDecomposition(I);
i4 : J=intersect(PD)
o4 = ideal (f*h - e*i, e*g - d*h, c*e - b*f, b*d - a*e)
o4 : Ideal of R
i5 : I==J
o5 = true
```

3.4.3　根基所属判定

sec:radicalmember

$f \in I$ ならば, f は I の零点集合 $V(I)$ 上で消えるが, 逆は一般には成り立たない. K が代数閉体の場合, f が $V(I)$ 上で消えるための必要十分条件は, Hilbert の零点定理により根基への所属で与えられる.

定義 3.4.4. R のイデアル I に対し, その根基 (radical) $\sqrt{I}$ を $\sqrt{I} = \{f \in R \mid$ ある正整数 m に対し $f^m \in I\}$ により定義する. $\sqrt{I}$ は R のイデアルである.

thm:radmembership

定理 3.4.5. R のイデアル I, $f \in R$ に対し,

$$f \in \sqrt{I} \Leftrightarrow R[t]I + \langle tf - 1\rangle = K[x_1, \ldots, x_n, t]$$
$$\Leftrightarrow R[t]I + \langle tf - 1\rangle \text{ の任意項順序に関する簡約グレブナー基底が } \{1\}$$

noprob:rad

問題 3.4.6. 定理 3.4.5 を示せ. ($\Rightarrow$ は容易. $\Leftarrow$ は, 1 を $R[t]I + \langle tf - 1\rangle$ の元として表して, t に $1/f$ を代入する.)

この定理より $f \in \sqrt{I}$ か否かは $R[t]I + \langle tf - 1\rangle$ のグレブナー基底一つを計算することで判定できる. 項順序は任意なので, 通常は全次数逆辞書式順序など, グレブナー基底が計算しやすい項順序を用いるのがよい.

次の例は, 3.3.6 節の例における $f \in \sqrt{I}$ の判定をここで説明した方法で行ったものである.

Macaulay2

$R[t]I + \langle tf - 1\rangle$ のグレブナー基底が $\{1\}$ なので, $f \in \sqrt{I}$ と判定できる.

Macaulay2 : 根基所属判定

```
i1 : R=QQ[t,x,y,z];
i2 : I=ideal(x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4);
o2 : Ideal of R
i3 : f=y*z-x^3;
i4 : gens gb (I+ideal(t*f-1))
o4 = | 1 |
```

SINGULAR

Macaulay2 と同様, $R[t]I + \langle tf - 1\rangle$ のグレブナー基底が $\{1\}$ なので, $f \in \sqrt{I}$ と判定できる.

SINGULAR : 根基所属判定

```
> ring r=0,(t,x,y,z),dp;
> ideal i=x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4;
> poly f=y*z-x^3;
> ideal j=t*f-1,i;
> groebner(j);
_[1]=1
```

CoCoA

組み込み関数 `IsInRadical` により判定できる. さらに, $f \in \sqrt{I}$ のとき, $f^m \in I$ となる最小指数 m を計算する `MinPowerInIdeal` も利用できる.

CoCoA：根基所属判定

```
Use R::=QQ[x,y,z];
I:=Ideal(x^4*y^2+z^2-4*x*y^3*z-2*y^5*z,x^2+2*x*y^2+y^4);
F:=y*z-x^3;
IsInRadical(F,I);
True
------------------------------
MinPowerInIdeal(F,I);
3
```

3.4.4　イデアル商, saturation

subsec:colonideal

定義 3.4.7. R のイデアル I, J に対し, イデアル商 $I : J$ を

$$I : J = \{f \mid fJ \subset I\}$$

で定義する. $J = \langle f \rangle$ のとき, $I : J$ を $I : f$ と書く.

定義から, 次が容易に得られる.

nothm:colon **定理 3.4.8.**　1. $J = \langle g_1, \ldots, g_l \rangle$ ならば $I : J = \bigcap_{i=1}^{l} (I : g_i)$.

2. $I : g = (I \cap \langle g \rangle)/g$.

$(I \cap \langle g \rangle)/g$ は, $I \cap \langle g \rangle$ の生成系の各元を g で割ったもので生成されるイデアルである. よって, $I : J$ はイデアルの共通部分計算により計算できる.

定義 3.4.9. R のイデアル I, J に対し $I : J^{\infty} = \bigcup_{m=1}^{\infty} (I : J^m)$ である.
$J = \langle f \rangle$ のとき $I : J^{\infty}$ を $I : f^{\infty}$ と書く.

nothm:sat **定理 3.4.10.**　1. $J = \langle g_1, \ldots, g_l \rangle$ ならば $I : J = \bigcap_{i=1}^{l} (I : g_i^{\infty})$ である.

2. $I : g^{\infty} = (I + \langle tg - 1 \rangle) \cap R$ である.

nosubsec:radicalmember nothm:sat
3.4.3 節を参考にして, 定理 3.4.10 を示せ.

よって saturation はイデアルの共通部分計算と, 消去イデアル計算により計

算できる. 次の例では, $\mathbb{Q}[x,y]$ のイデアル $I=\langle x^4-y^5, x^3-y^7\rangle$ に対し, $I:x^k=(I:x^{k-1}):x$ を再帰的に利用して, $I:x^k$ が不変になるまでイデアル商を計算し, それが $I:x^\infty$ に等しいことを確かめている.

Macaulay2

イデアル I,J に対するイデアル商 $I:J$ は `quotient`(I,J), saturation $I:J^\infty$ は `saturate`(I,J) により計算できる.

Macaulay2 : イデアル商, saturation

noprob:sat 問題 **3.4.11.**

```
i1 : R=QQ[x,y];
i2 : I=ideal(x^4-y^5,x^3-y^7);
o2 : Ideal of R
i3 : I1=quotient(I,x);
o3 : Ideal of R
i4 : I2=quotient(I1,x);
o4 : Ideal of R
i5 : I1==I2
o5 = false
i6 : I3=quotient(I2,x);
o6 : Ideal of R
i7 : I2==I3
o7 = false
i8 : I4=quotient(I3,x);
o8 : Ideal of R
i9 : I3==I4
o9 = true
i10 : J=saturate(I,x);
o10 : Ideal of R
i11 : I3==J;
o11 = true
```

SINGULAR

$I:J$ は `quotient`(I,J), saturation $I:J^\infty$ は `elim.lib` で定義される `sat`(I,J) で計算できる. `sat`(I,J) は, $I:J^\infty$ と $I:J^\infty=I:J^m$ を満たす最小整数 m の組を返す.

SINGULAR : イデアル商, saturation

```
> ring r=0,(x,y),dp;
> ideal i=x^4-y^5,x^3-y^7;
> LIB "elim.lib";
...
> list s=sat(i,x);
> s;
[1]:
   _[1]=xy2-1
   _[2]=y5-x4
   _[3]=x5-y3
[2]:
   3
```

```
> ideal g3=quotient(i,x^3);
> g3=groebner(g3);
> g3;
g3[1]=xy2-1
g3[2]=y5-x4
g3[3]=x5-y3
> size(reduce(g3,s[1]));
0
> size(reduce(s[1],g3));
0
```

size コマンドは, リスト中の 0 でない要素の個数を返すので, 上の実行結果は g3 と s[1] が等しいイデアルであることを示す.

CoCoA

$I : J$ は Colon(I,J) または I:J, $I : J^\infty$ は Saturation(I,J) で計算できる. いずれの関数も, 第二引数はイデアルしか受け付けない.

CoCoA : イデアル商, saturation

```
Use R::=QQ[x,y];
I:=Ideal(x^4-y^5,x^3-y^7);
I:Ideal(x^3);
Ideal(xy^2 - 1, y^5 - x^4, x^5 - y^3)
------------------------------
Colon(I,Ideal(x^3));
Ideal(xy^2 - 1, y^5 - x^4, x^5 - y^3)
------------------------------
Saturation(I,Ideal(x));
Ideal(xy^2 - 1, y^5 - x^4, x^5 - y^3)
```

3.4.5　根基計算

nosubsec:radical

nosubsec:radicalmember

3.4.3 節で述べたように, 根基所属判定はグレブナー基底計算 1 回でできるが, イデアル I の根基 $\sqrt{I}$ の計算は以下で見るように容易な計算ではな

い. ここでは, これまで述べたさまざまなイデアル演算の応用として, その概要を紹介する. 定理の証明は, 容易なものは問題とした. それ以外については [NY1] 4.4 節を参照してほしい.

定義 3.4.12 (無平方部分)**.** 多項式 f の既約分解を $f = f_1^{n_1} \cdots f_l^{n_l}$ $(n_1, \ldots, n_l \geq 1)$ とするとき, $f_1 \cdots f_l$ を f の無平方部分と呼ぶ.

定理 3.4.13 (Seidenberg)**.** K を完全体とし I を $K[x_1, \ldots, x_n]$ の 0 次元イデアル (定義 3.5.1 参照) とする. このとき f_i $(i = 1, \ldots, n)$ を $I \cap K[x_i]$ の生成元の無平方部分とすれば, $\sqrt{I} = I + \langle f_1, \ldots, f_n \rangle$.

nodef:extcont **定義 3.4.14** (extension と contraction)**.** I を $K[X]$ のイデアルとする. $U \subset X$ に対し, I で生成される $K(U)[X \setminus U]$ のイデアルを I^e と書き, I の $K(U)[X \setminus U]$ への extension と呼ぶ. また, $K(U)[X \setminus U]$ のイデアル J に対し $K[X]$ のイデアル $J \cap K[X]$ を J^c と書き, J の $K[X]$ への contraction と呼ぶ.

nothm:cont **定理 3.4.15.** J を $K(U)[X \setminus U]$ のイデアルとし, $X \setminus U$ 上の項順序 $<_1$ に関する J のグレブナー基底 $G = \{g_1, \ldots, g_l\} \subset K[U][X \setminus U] = K[X]$ が与えられているとする. このとき, 各 g_i の $<_1$ に関する先頭係数 を $h_i \in K[U]$ とおき, f を $\mathrm{LCM}(h_1, \ldots, h_l)$ の無平方部分とすれば, G で生成される $K[X]$ のイデアル $\tilde{J} = \langle G \rangle \subset K[X]$ に対し, $J^c = \tilde{J} : f^\infty$.

noprob:extcont **問題 3.4.16.** 定理 3.4.15 を示せ.

nocor:cont **系 3.4.17.** $<_1, <_2$ をそれぞれ $X \setminus U, U$ 上の項順序とし, $<$ をそれらで定まるブロック順序とする. $K[X]$ のイデアル I の $<$ に関するグレブナー基底を $G = \{g_1, \ldots, g_l\}$ とする. 各 g_i の $<_1$ に関する先頭係数 を $h_i \in K[U]$ とおき, f を $\mathrm{LCM}(h_1, \ldots, h_l)$ の無平方部分とすれば, $I^{ec} = I : f^\infty$.

この系は定理 3.4.2 の応用であるが, G は I^e のグレブナー基底としては一般に多くの冗長元を持つため, 極小基底あるいは簡約基底にした上で h_i, f を計算すべきであろう.

定義 3.4.18 (極大独立集合)**.** I を $K[X]$ のイデアルとする. $U \subset X$ で, 次を満たすものを I の極大独立集合と呼ぶ.

1. $K[U] \cap I = \{0\}$.
2. 任意の $x \in X \setminus U$ に対し $K[U \cup \{x\}] \cap I \neq \{0\}$.

注意 3.4.19.　1. I の極大独立集合の元の個数の最大数は I の Krull 次元 $\dim I$ に等しい.

2. $\mathrm{in}_< I \cap K[U] = \{0\}$ なる $U \subset X$ (強独立集合) は独立集合である. 項順序が全次数つきなら強独立集合の元の個数の最大数は $\dim I$ に等しいので, $\mathrm{in}_< I$ から極大独立集合を求めることができる.

nothm:satdecomp **定理 3.4.20.** $I : f^\infty = I : f^s$ ならば $I = (I : f^s) \cap (I + \langle f^s \rangle)$.

noprob:satdecomp **問題 3.4.21.** 定理 nothm:satdecomp 3.4.20 を示せ.

othm:radsatdecomp **定理 3.4.22.** I を $K[X]$ のイデアルとし, $U \subset X$ を I に対する極大独立集合とする. このとき系 nocor:cont 3.4.17 の f をとれば $f \notin I$ であり,

$$\sqrt{I} = \sqrt{I^e}^c \cap \sqrt{I + \langle f \rangle}.$$

prob:radsatdecomp **問題 3.4.23.** 定理 nothm:radsatdecomp 3.4.22 を示せ.

アルゴリズム 3.4.24 ($Radical(I)$)**.**
$U \leftarrow I$ の極大独立集合
$f \leftarrow I^{ec} = I : f^\infty$ なる $f \in K[X]$　(系 nocor:cont 3.4.17 により計算する)
$\tilde{J} \leftarrow \sqrt{I^e}$　(0 次元イデアル I^e の根基)
$J \leftarrow \tilde{J}^c$　(定理 nothm:cont 3.4.15 により計算する)
$J' \leftarrow \mathrm{Radical}(I + \langle f \rangle)$
return $J \cap J'$

このアルゴリズムの停止性は, $f \notin I$ より $I + \langle f \rangle$ が真に増大することからネーター性により保証される. また出力が $\sqrt{I}$ に等しいことは定理 nothm:radsatdecomp 3.4.22 により分かる.

3.5 項順序変換

指定された項順序でのグレブナー基底計算を Buchberger アルゴリズムで直接行うと, 中間基底が多数生成されたり, 中間基底の係数が膨張したりして計算が続行できなくなる場合がある. このような場合に, いったん計算しやすい項順序でグレブナー基底を計算しておき, それをもとに目的の項順序でのグレブナー基底を計算するという方法がいくつか考案されている. このような方法を一般に項順序変換 (change of ordering) と呼ぶ. ここでは, 0 次元イデアルのグレブナー基底を線形代数的手法を用いて計算する FGLM アルゴリズムと, 斉次イデアルのグレブナー基底を Hilbert 関数を用いて計算する Hilbert driven アルゴリズムについて簡単に解説する. 項順序変換には, グレブナー fan の概念に基づくグレブナー walk アルゴリズムがあり興味深いが, 予備知識を多く必要とするためここでは述べない.

3.5.1 FGLM アルゴリズム

nosubsec:fglm

noth:zerodim

定理 3.5.1. イデアル $I \subset R$ に対し, 次は同値である.

1. I の Krull 次元 $\dim I = 0$.
2. R/I が K-線形空間として有限次元.
3. I の $<$ に関するグレブナー基底 G が, 各変数 x_i $(i = 1, \ldots, n)$ に対し $\mathrm{in}_<(g_i) = x_i^{m_i}$ なる元 g_i を含む.
4. $\overline{K}$ を K の代数閉包とするとき $V_{\overline{K}}(I)$ が有限集合.
5. 各変数 x_i $(i = 1, \ldots, n)$ に対し, I が x_i の一変数多項式を含む.

定義 3.5.2 (0 次元イデアル)**.** イデアル $I \subset R$ が 0 次元イデアルであるとは, 定理 noth:zerodim 3.5.1 の同値な条件を満たすことをいう.

0 次元イデアルは, 解が有限個であるような代数方程式系を表しており, 実用上多く現れる. また, 第一章で見たように, 純辞書式順序に関するグレブナー基底が求まれば, 一変数方程式の解を求めることにより, 他の変数の値を比較的たやすく求めることができる. しかし, 残念ながら純辞書式順序に関するグレブナー基底を Buchberger アルゴリズムで求めることは多くの場合大変大きい計算コストを必要とする. これから説明する FGLM アルゴリ

ズムは, 0 次元イデアル I に対し, 例えば全次数逆辞書式など, 計算しやすい項順序に関するグレブナー基底 G_0 を求めておき, 目的の項順序に関するグレブナー基底 G_1 を線形代数を用いて計算する方法である.

項順序 $<$ に関して, $\mathrm{in}_<(I)$ に属さない単項式全体を標準単項式集合とよび $SM_<(I)$ と書くことにする. グレブナー基底の定義により $SM_<(I)$ は R/I の K-基底をなす. 特に I が 0 次元イデアルのとき $\dim_K R/I$ は有限だから, $SM_<(I)$ も有限集合である. FGLM においては, 目的項順序 ($<_1$ とする) に関する標準単項式集合 $SM_{<_1}(I)$ を, $<_1$ に関して小さい順に探して並べていく. このとき, 二つの操作が必要となる.

1. 残りの単項式集合の中で, $<_1$ に関して最小なものを見つける.
2. 与えられた単項式集合の元の K 線形和で, I の元であるものがあるかどうかを判定し, あるなら実際その元を求める.

まずアルゴリズムを示す. アルゴリズム中で, $\mathrm{NF}_<(f, F)$ は f をグレブナー基底 F で割った余りを意味する.

noalg:fglm

アルゴリズム 3.5.3 (FLGM アルゴリズム)**.**
Input : 0 次元イデアル I の $<$ に関するグレブナー基底 F
Output : I の $<_1$ に関するグレブナ基底
$G \leftarrow \emptyset$; $h \leftarrow 1$; $B \leftarrow \{h\}$; $H \leftarrow \emptyset$
do
　$N \leftarrow \{u \mid h <_1 u$ かつすべての $m \in H$ に対し $m \not| u\}$
　if $N = \emptyset$ then return G
(1)　$h_1 \leftarrow N$ 中で, $<_1$ に関して最小の単項式
(2)　$E \leftarrow \mathrm{NF}_<(h_1, F) + \sum_{t \in B} a_t \mathrm{NF}_<(t, F)$
　if $E = 0$ を満たす $\{a_t \in K \mid t \in B\}$ が存在する then
　　$G \leftarrow G \cup \{h_1 + \sum_{t \in B} a_t t\}$; $H \leftarrow H \cup \{h_1\}$
　else $B \leftarrow \{h_1\} \cup B$
　$h \leftarrow h_1$
end do

アルゴリズム中で, (1), (2) がそれぞれ上で述べた 1., 2. に対応する. (2) は, $E = 0$ から $SM_<(I)$ の各単項式の係数 $= 0$ が得られ, これらは a_t の線形方程式系をなすので, その解の存在判定を行えばよい. (1) については次の命題により, 有限個の単項式の比較に帰着する.

noprop:fglm **命題 3.5.4.** (1) において, $h_1 \in x_1B \cup \cdots \cup x_nB$.

命題 3.5.4 およびアルゴリズムの正当性の証明は [NY] 5.2 節にある. 証明はここでは述べないが, FGLM アルゴリズムは大抵のシステムに実装されている. また, グレブナー基底による剰余計算と, 線形方程式求解さえあれば実装は可能なので, 自力で実装してみるのもおもしろいだろう. ただし, 上記アルゴリズムはあくまで理解を助けるための例であり, 効率の面から見て問題がある. 特に, (2) に関しては, 無駄な線形方程式求解を繰り返さぬようインクリメンタルに行う必要があろう.

FGLM アルゴリズムによる計算例として, 3.4.1 節の終わりの例に対する辞書式順序グレブナー基底を SINGULAR 上で計算みる. この例の場合, 辞書式順序グレブナー基底には 5000 桁以上の係数が現れる. このような例に対し, 直接 Buchberger アルゴリズムで計算するのは係数膨張のため困難な場合が多い.

SINGULAR : FGLM

```
> timer=1;
> option(redSB);
> ring r=0,(a,b,c,d,e),dp;
> ideal i= 4*a^3+3*c*b^2*a^2-a+2,-3*c*a^3+3*c^2*a^2-a
+3*c*b^4+c^3,-6*a^2+(-3*c^3-4)*a+3,(4*c*b^2-1)*a^2
+(d*b+c)*a+b^4,2*a^3-3*a^2+(-d^2-2*e)*b+3;
> ideal g=groebner(i);
//used time: 57.31 sec
> ring s=0,(a,b,c,d,e),lp;
> ideal j=fglm(r,g);
//used time: 32.55 sec
```

fglm が入力として被約なグレブナー基底を要求するため, オプション redSB を指定している. なお, この一連の操作を自動的に行う stdfglm がライブラ

リ standard.lib (起動時に自動的に読み込まれる) で定義されている. 計算時間を見ると, 全次数辞書式順序グレブナー基底の計算より辞書式順序グレブナー基底の方が高速であり, FGLM アルゴリズムの効果が表れている.

3.5.2 Hilbert driven アルゴリズム

nosubsec:hilb

FGLM アルゴリズムは 0 次元イデアルにのみ適用できる項順序変換であるが, 次に紹介する Hilbert driven アルゴリズムは任意の斉次イデアルに対して適用できる. さらに, 斉次化を経由すれば, 斉次とは限らないイデアルに対しても適用できる. 本節の内容については, [9] 5.3 節に詳しく述べられている.

$R_d = \{f \in R \mid \mathrm{tdeg}(f) = d\}$ とおく. 第 1 章で説明されているように, I が斉次イデアルのとき, よって R/I の Hilbert 関数 $H(R/I; d)$ が, 任意の項順序におけるグレブナー基底により与えられる. 3.1 節で述べたように, Buchberger アルゴリズムにおいては不必要なペアがたくさん現れて計算を困難にする場合があるが, 3.1.3 節で述べたように斉次イデアルに対する Buchberger アルゴリズムを次数順に実行する場合, 既に得られた全次数 d 次以下の基底のどの先頭項でも割れない R_d の単項式の個数が $H(R/I; d)$ の値に等しくなった時点で $(R/I)_d$ の K-基底がすべて得られたことになる. このとき, 残りの全次数 d の S 多項式はすべて 0 に簡約されることが分かる. この方法により, 不必要ペアを除去する方法を Hilbert driven アルゴリズムと呼ぶ.

これまでに紹介した 3 つのシステムいずれも Hilbert 関数, あるいはその母関数 (Hilbert-Poincaré 級数) の計算を提供しており, それを用いた Hilbert driven アルゴリズムによる高速なグレブナー基底計算が可能である. ここでは, 再び SINGULAR による例を紹介する. stdhilb (standard.lib で定義されている) は, 入力イデアルの Hilbert 関数を計算し, それを使った Hilbert driven アルゴリズムにより目的の項順序に関するグレブナー基底を計算する. 入力が斉次イデアルでない場合には, 自動的に斉次化を行って計算する.

SINGULAR : Hilbert driven アルゴリズム

```
> ring r=0,(a,b,c,d,e,f,g),(dp(3),dp(4));
> timer=1;
> option(prot);
> ideal i=-3*a^2+2*f*b+3*f*d,(3*g*b+3*g*e)*a-3*f*c*b,
-3*g^2*a^2-c*b^2*a-g^2*f*e-g^4,e*a-f*b-d*c;
> ideal j=stdhilb(i);
compute hilbert series with slimgb in ring (0),
(a,b,c,d,e,f,g,@),(dp(8),C)
weights used for hilbert series: 1,1,1,1,1,1,1
slimgb in ring (0),(a,b,c,d,e,f,g,@),(dp(8),C)
CC2M[1,1](2)C3M[1,1](2)4M[2,2](5)C5M[5,4](14)C6M[11,5](19)...
NF:118 product criterion:36, ext_product criterion:11
std with hilb in (0),(a,b,c,d,e,f,g,@),(dp(3),dp(4),dp(1),C)
[255:5]2(34)s(33)s3s(34)s4(36)s(38)ss(39)s(42)--s5(44)s(45)...
...
26(10)---shhhhhh27(5)shhhhh28(4)-shhh29-shhh30-shhhhhhh
product criterion:453 chain criterion:41711
hilbert series criterion:912
//used time: 63.30 sec
dehomogenization
simplification
imap to ring (0),(a,b,c,d,e,f,g),(dp(3),dp(4),C)
```

この例では, 7 変数多項式環におけるイデアルから 3 変数消去する計算を Hilbert driven アルゴリズムで実行している. `prot` オプションを指定することで, 計算途中の情報が表示される. `slimgb` (Buchberger アルゴリズムの変種) で計算した全次数逆辞書式順序グレブナー基底により Hilbert 級数を計算し, それを使って消去順序に関する Buchberger アルゴリズムにおいて不要ペアの除去を行っている. 例えば, `std with hilb` の終了直前の `30-shhhhhhh` は, sugar(=全次数) 30 の計算で, 一つ基底が生成された時点 (`s` で示される) で 30 次の基底がすべて得られたことが分かり, 残りの S 多項式はすべて不要として捨てられている (`h` で示される) ことが分かる. 通常の Buchberger アルゴリズムで計算してみると, 特に後半捨てられている S 多項式を 0 まで簡約する計算が大変時間がかかることが観測される.

3.6　加群のグレブナー基底計算

nosec:modulegb

本書では, 多項式環のイデアルのグレブナー基底だけでなく, 多項式環上の自由加群, すなわち多項式環の直和の部分加群を扱う必要がある. 幸いなことに, グレブナー基底の概念および Buchberger アルゴリズムは容易に加群に拡張できる. 本節では, 加群特有の項順序について説明し, そのあと加群に拡張された Buchberger アルゴリズムについて解説する. 本節でも, $R = K[x_1, \ldots, x_n]$ と書くことにする.

3.6.1　多項式環上の自由加群における項順序

subsec:moduleorder

以下で, 自由加群 R^m および R^m の部分加群を考える. R^m の標準基底を $e_1, \ldots, e_m$ とすると, R^m の元 f は

$$f = c_1 t_1 e_{i_1} + \cdots + c_l t_l e_{i_l} \tag{3.1}$$

($t_1, \ldots, t_l$ は R の単項式, $c_1, \ldots, c_l \in K \setminus \{0\}$) と書ける. 以下 te_i の形の R^m の元を R^m の単項式と呼ぶことにする.

nodef:moduleord

定義 3.6.1 (R^m の項順序)**.** R^m の単項式全体における全順序 $<$ で, 次を満たすものを R^m の項順序と呼ぶ.

1. R^m の単項式 u, v, R の単項式 t に対し, $u < v$ ならば $tu < tv$.
2. R の任意の単項式 t, R^m の任意の単項式 u に対し $u \leq tu$.

注意 3.6.2. 定義 3.6.1 における条件 2. は,

$$R \text{ の任意の単項式 } t,\ i = 1, \ldots, m \text{ に対し } e_i \leq te_i.$$

に置き換えてよい.

nodef:modord

定義 3.6.3 (加群における代表的な項順序)**.** $<$ を R における項順序とするとき, R^m における次の二つの項順序が定義できる.

1. TOP (Term Over Position) 順序
   $te_i <_{TOP} se_j \Leftrightarrow t < s$ または $(t = s$ かつ $i > j)$
2. POT (Position Over Term) 順序

$te_i <_{POT} se_j \Leftrightarrow i > j$ または $(i = j$ かつ $t < s)$

R^m の 0 でない元 f を, (3.1) かつ $t_1 > t_2 > \cdots > t_l$ と表すとき, t_1 を f のイニシャル単項式とよび $\mathrm{in}_<(f)$ と書く.

R^m の単項式集合に対しても Dickson の補題が成立する. よって, R^m の単項式で生成される部分加群は有限生成である.

定義 3.6.4. R^m の部分加群 M に対し, $G = \{g_1, \ldots, g_k\} \subset M$ で, $\langle \{\mathrm{in}_<(f) \mid f \in M\} \rangle = \langle \mathrm{in}_<(g_1), \ldots, \mathrm{in}_<(g_k) \rangle$ を満たすものを M のグレブナー基底という.

R^m においても単項式除算が有限回で停止するので, M の元を M のグレブナー基底で割った余りは 0 となる. よって M のグレブナー基底は M を R 上生成する.

noprob:intbymod

問題 3.6.5. R のイデアル $I = \langle f_1, \ldots, f_l \rangle$, $J = \langle g_1, \ldots, g_m \rangle$ に対し R^2 の部分加群 M を $M = \langle \begin{pmatrix} f_1 \\ 0 \end{pmatrix}, \ldots \begin{pmatrix} f_l \\ 0 \end{pmatrix}, \begin{pmatrix} g_1 \\ g_1 \end{pmatrix}, \ldots \begin{pmatrix} g_m \\ g_m \end{pmatrix} \rangle$ とおく. R の項順序 $<$ に対する POT 順序 $<_{POT}$ に関する M のグレブナー基底を G とし, $G_0 = \{g \in R \mid (0, g) \in G\}$ とおけば G_0 は $I \cap J$ の $<$ に関するグレブナー基底である.

3.6.2 加群における Buchberger アルゴリズム

nosubsec:modulegb

R^m の部分加群のグレブナー基底を求めるために, S 多項式の定義を R^m の元のペアに対して拡張する.

定義 3.6.6 (加群における S 多項式)**.** $f, g \in R^m \setminus \{0\}$ に対し, $S(f, g)$ を次で定義する. $\mathrm{in}_<(f) = te_i$, $\mathrm{in}_<(g) = se_j$, f, g における $\mathrm{in}_<(f)$, $\mathrm{in}_<(g)$ の係数をそれぞれ c_f, c_g とする.

1. $i \neq j$ ならば $S(f, g) = 0$.
2. $i = j$ ならば $S(f, g) = (\mathrm{LCM}(t, s)/c_f t) \cdot f - (\mathrm{LCM}(t, s)/c_g s) \cdot g$.

定理 3.6.7. R^m の部分加群 M に対し, $G = \{g_1, \ldots, g_k\} \subset M \setminus \{0\}$ が M

のグレブナー基底であるための必要十分条件は, 任意の i, j に対し $S(g_i, g_j)$ を G で割った余りが 0 となることである.

この定理により, M のグレブナー基底計算は Buchbeger アルゴリズムにより行うことができることが分かる. また, 不必要ペアの除去, ペアの選び方についてもイデアルの場合とほぼ同様であるので省略する.

3.6.3　syzygy の計算

nosubsec:syzygy

加群に関する計算で最も基本的な計算が, syzygy 計算である.

定義 3.6.8. M を R 加群とする. $f_1, \ldots, f_m \in M$ に対し,

$$\{(h_1, \ldots, h_m) \in R^m \mid h_1 f_1 + \cdots + h_m f_m = 0\}$$

を $(f_1, \ldots, f_m)$ の syzygy と呼び, $\mathrm{syz}(f_1, \ldots, f_m)$ と書く. $\mathrm{syz}(f_1, \ldots, f_m)$ は R^m の部分加群である.

$f_1, \ldots, f_m \in R^l$ に対し, $M = \langle f_1, \ldots, f_m \rangle$ とおく. 一般に syzygy 加群の生成系, あるいはそのグレブナー基底を求めるのは容易ではないが, $\{f_1, \ldots, f_m\}$ が M のグレブナー基底の場合には除算により syzygy 加群の生成系を得ることができる.

nothm:gbsyz

定理 3.6.9. $G = \{f_1, \ldots, f_m\} \subset R^l$ が $M = \langle G \rangle$ のグレブナー基底とする. 0 でない $S(f_i, f_j) = u_i f_i - u_j f_j$ $(u_i, u_j \in R)$ の G による割り算の結果が

$$S(f_i, f_j) = \sum_{k=1}^{m} h_{ijk} f_k, \quad \mathrm{in}_<(h_{ijk} f_k) \leq \mathrm{in}_<(S(f_i, f_j))$$

$(k = 1, \ldots, m)$ であるとする. このとき, $s_{ij} \in R^m$ を

$$s_{ij} = u_i e_i - u_j e_j - \sum_{k=1}^{m} h_{ijk} e_k$$

とおけば $S = \{s_{ij} \mid 1 \leq i, j \leq m,\ S(f_i, f_j) \neq 0\}$ は $\mathrm{syz}(G)$ の生成系である.

注意 3.6.10. 上の S は, R^m のある特殊な項順序 (Schreyer 順序) に関するグレブナー基底になっている (Schreyer の定理).

一般の $(f_1, \ldots, f_m) \in R^l$ の syzygy の計算法については次の二つの方法が知られている.

noalg:syz1 **アルゴリズム 3.6.11** (c.f. [UAG 6] chapt. 5, sect. 3)**.**
入力 : $F = (f_1, \ldots, f_m)$, $f_i \in R^l$ $(i = 1, \ldots, m)$
出力 : $\mathrm{syz}(F)$ の生成系
$G = (g_1, \ldots, g_t) \leftarrow \langle F \rangle$ のグレブナー基底
$C \leftarrow {}^tG = C \cdot {}^tF$ を満たす (t, m) 行列
$D \leftarrow {}^tF = D \cdot {}^tG$ を満たす (m, t) 行列
$S = \{s_1, \ldots, s_u\} \leftarrow \mathrm{syz}(G)$ の生成系
$\{r_1, \ldots, r_m\} \leftarrow I_m - DC$ の各行 (I_m は m 次単位行列)
return $\langle s_1 C, \ldots, s_u C, r_1, \ldots, r_m \rangle$

このアルゴリズムでは, 後で述べる左 D 加群の場合にも適用できるよう, 係数行列を左からかけるように書いた. $\mathrm{syz}(G)$ の生成系は定理 3.6.9 (nothm:gbsyz) により計算できる. 行列 D は, F の各元を G で割った商として計算できる. しかし, 行列 C は, Buchberger アルゴリズムにおいて, S 多項式の剰余だけでなく商の部分も保持しながら計算を実行する必要が生ずるため, 単なるグレブナー基底計算よりコストが大きくなる. また, 得られる結果は必ずしも $\mathrm{syz}(F)$ のグレブナー基底とはならない.

noalg:syz2 **アルゴリズム 3.6.12** (c.f. [UAG 6] Exercise 15 in chapt. 5, sect. 3)**.**
入力 : $F = (f_1, \ldots, f_m)$, $f_i \in R^l$ $(i = 1, \ldots, m)$, 項順序 $<$
出力 : $\mathrm{syz}(F)$ の $<_{POT}$ に関するグレブナー基底
$(e_1, \ldots, e_m) \leftarrow R^m$ の標準基底
$u_i \leftarrow (f_i, e_i) \in R^l \oplus R^m = R^{l+m}$ $(i = 1, \ldots, m)$
$\tilde{G} \leftarrow \langle u_1, \ldots, u_m \rangle$ の $<_{POT}$ に関するグレブナー基底
$S \leftarrow \{h \in R^m \mid (0, h) \in \tilde{G}\}$
return S

noprob:syz2 問題 **3.6.13.** アルゴリズム noalg:syz2 3.6.12 が $\mathrm{syz}(F)$ の $<_{POT}$ に関するグレブナー基底を出力することを示せ.

このアルゴリズムは, F がイデアルの生成系 ($l = 1$) の場合にも, 加群におけるグレブナー基底計算を行わなければならないなど, グレブナー基底計算のコストはアルゴリズム noalg:syz1 3.6.11 と同様大きくなるが, 必要なのは単なるグレブナー基底である. また, 得られる結果 S は $<_{POT}$ に関するグレブナー基底となっている. さらに,

$$G = \{g \in R^l \mid g \neq 0 \text{ かつある } h \text{ に対し } (g, h) \in \tilde{G}\}$$

とおくと G は $\langle F \rangle$ のグレブナー基底であり, また $G = \{g_1, \ldots, g_t\}$ とするとき

$$C = \text{第 } i \text{ 行が } (g_i, h_i) \in \tilde{G} \text{ に対する } h_i \text{ であるような } (t, m) \text{ 行列}$$

とおくと ${}^tG = C \cdot {}^tF$ が成り立つ.

3.7 Risa/Asir 上での計算

これまで説明してきた 3 つのソフトウェアは, 細かい差はあるものの, 項順序が設定された基礎環を定義してその中のオブジェクトに対して諸演算を行うという共通の設計を持つ. 一方で, 本節で解説する Risa/Asir はこれらとは異なる設計のシステムであり, 前節の 3 つとはかなり使い勝手が異なるので, 節を改めて解説することにした.

3.7.1 起動方法

- $\sqrt{x}$ メニュー, または KNOPPIX-Math-Start アイコンから起動する. (openxm) の方を起動すれば, 種々のライブラリファイルを自動的に読みこんで起動する.
- 端末エミュレータから起動する.
  Asir 単体ではコマンドライン編集機能を持たないので, `openxm fep asir` を実行する.

3.7.2 ヘルプ, マニュアル

ヘルプは help("function") で引ける. マニュアルはデスクトップの Math-Doc-Search で引くか, helph() コマンドでブラウザを立ち上げて HTML 形式のマニュアルを見るのが便利である.

3.7.3 ファイルの読み書き

ユーザプログラムファイル, ライブラリファイルの読み込みは load により行う. 環境変数 ASIRLOADPATH で指定されたディレクトリを順に探す. この値は, シェルから openxm env を実行すると見ることができる. コマンド output("file") を実行すると, output() を実行するまで画面への出力がファイルに書き込まれる. これらは可読形式での入出力であるが, データの高速な入出力のための機能もある. bsave(Data,"file") により, データをバイナリ形式でファイルに出力することができる. bsave で書き込んだデータは bload("file") により読み込むことができる. 指定できるデータは 1 つであるが, リストでくくれば複数のデータをまとめて書き込むこともできる.

3.7.4 多項式の入力

Asir ではアルファベット小文字で始まり, アルファベット, 数字, _ (アンダースコア) からなる文字列が不定元である. 不定元を含む多項式が入力された場合, システムが内部的に保持する不定元リストの順序に従い内部形式に変換され保持される. システムが知らない不定元はリストの最後尾に順次追加される. 入力された係数は自動的に有理数と判断される. 入力された多項式は, 内部の不定元順序が変更されない限り, 自由に加減乗算が可能である. しかし, 項順序に関する情報はなんら保持されていないため, グレブナー基底関連計算など, 項順序が必要が計算の都度, 項順序を指定する必要がある. この方式は, 前節で説明したソフトウェアと異なり, ユーザに項順序の存在を常に意識させる点でわずらわしいが, 一方で, 項順序を変えて計算したり, 変数を増やして計算したりするたびに新しい環を生成するという必要がないという利点もある.

多項式の入力

```
[1518] F=(x+y+z)^2;
x^2+(2*y+2*z)*x+y^2+2*z*y+z^2
[1519] G=F+u;
x^2+(2*y+2*z)*x+y^2+2*z*y+z^2+u
```

この表示から分かるように, Asir における多項式は再帰表現により保持されている. 再帰表現とは, 多項式を, 主変数に関する一変数多項式として表現するもので, 係数は, 主変数を含まない多項式である. これに対し, グレブナー基底に関連する計算では, 多項式を単項式の和として表現するのが便利である. これを分散表現と呼ぶ. Asir においては, グレブナー基底関連計算を行う場合, 暗黙あるいは明示的に分散表現への変換を行う. これについては後述する.

3.7.5　項順序

Asir においては, 項順序は変数順序と項順序型により指定される. 変数順序は不定元を並べたリストで表現する. この順序は単項式を指数ベクトルで表示する場合の各指数のインデックスを決める. 例えば, 変数順序が $[x, y, z, u, v, w]$ で与えられた場合, $x^a y^b z^c u^d v^e w^f$ は (a, b, c, d, e, f) で表示される. n 変数の変数リストが与えられているとき, 次のような項順序型が設定できる.

- 単純な項順序型
  整数値で表される. 0 は全次数逆辞書式, 1 は全次数辞書式, 2 は辞書式順序を表す. これらは, 上の変数順序によりベクトルで表された単項式に対し適用される.
- ブロック項順序型
  $[[O_1, n_1], [O_2, n_2], \ldots, [O_l, n_l]]$ なるリストのリストで表される. これは, 変数リストを左から $n_1, n_2, \ldots, n_l$ $(n_1 + \cdots + n_l = n)$ ずつのブロックに分け, i 番目のブロックに単純項順序型 O_i を適用する項順序である. 項順序比較は, 1 番目のブロックから, 大小が決まるまで順に行う. よく使われるのは $[[0, n_1], [0, n_2]]$ なるもので, 先頭の n_1 変数を消去するた

めの消去順序の一つである.

- 行列による項順序型
  $m \times n$ 整数行列 M で表される. これは, 二つの単項式 $e = (e_1, \ldots, e_n)$, $f = (f_1, \ldots, f_n)$ に対し

$$e > f \Leftrightarrow M(e-f) \text{ の } 0 \text{ でない最も上の要素が正}$$

  として定義される. M が項順序を表すためには,
  - 整数ベクトル e に対し $Me = 0 \Leftrightarrow e = 0$
  - 各列の 0 でない最も上の要素が正

  という条件を満たす必要があるが, これを保証するのはユーザの責任である.

項順序型は, dp_ord により設定できる. あるいは関数の引数として与える場合もある.

分散表現への変換

```
[1532] F=x^2*y+y^3*z+x*z+x+1;
y*x^2+(z+1)*x+z*y^3+1
[1533] dp_ord(0)$
[1534] DF0=dp_ptod(F,[x,y,z]);
(1)*<<0,3,1>>+(1)*<<2,1,0>>+(1)*<<1,0,1>>+(1)*<<1,0,0>>
+(1)*<<0,0,0>>
[1535] dp_ord(2)$
[1536] DF2=dp_ptod(F,[x,y,z]);
(1)*<<2,1,0>>+(1)*<<1,0,1>>+(1)*<<1,0,0>>+(1)*<<0,3,1>>
+(1)*<<0,0,0>>
[1537] G=F+u;
y*x^2+(z+1)*x+z*y^3+u+1
[1538] DG=dp_ptod(G,[u,x,y,z]);
(1)*<<1,0,0,0>>+(1)*<<0,2,1,0>>+(1)*<<0,1,0,1>>
+(1)*<<0,1,0,0>>
+(1)*<<0,0,3,1>>+(1)*<<0,0,0,0>>
[1539] dp_ht(DG);
(1)*<<1,0,0,0>>
```

この例では, 多項式 F, G を明示的に分散表現に変換している. dp_ord(0) に

より DF0 は全次数逆辞書式で整列され, dp_ord(2) により DF2 は辞書式に整列される. また, DG は, u が最大の辞書式となるため, <<1,0,0,0>> が先頭となっている. dp_ht は先頭項 (係数 1), dp_hc は先頭係数, dp_hm は係数つきの先頭項を返す.[4] 分散表現多項式間の演算は, 同じ項順序で変換されたものに限られる. Macaulay2, SINGULAR, CoCoA と異なり, これを守るのはユーザの責任である.

3.7.6 グレブナー基底の計算

本節では, イデアルの入力およびグレブナー基底計算について述べる. 以下で述べる関数はライブラリ gr および noro_pd.rr に定義されているので, これらををロードしておく必要がある. Asir では, イデアルは多項式のリストで表現される. この段階では基礎環にあたるものはまだ未定である. グレブナー基底計算の際に指定される変数リストと項順序型, および係数体を指定する引数により基礎環が係数体を含めて決定される. 主なグレブナー基底計算関数を挙げる.

- nd_gr($Plist, Vlist, Char, Ord$)
  $Plist$ はイデアルを表す多項式リストである. 変数リスト $Vlist$, 項順序型 Ord で指定される項順序をもつ多項式環のイデアル $\langle Plist \rangle$ の簡約グレブナー基底を計算する. $Char = 0$ のとき有理数体係数, $Char$ が素数のとき有限体 $\mathbb{F}_{Char}$ 上で計算する. 結果は多項式のリストである.
- nd_gr_trace($Plist, Vlist, Homo, Prime, Ord$)
  この関数は有理数体上のグレブナー基底を, 有限体上でのショートカット計算を用いて効率よく計算するための関数である.
  $Plist$ はイデアルを表す多項式リストである. 変数リスト $Vlist$, 項順序型 Ord で指定される項順序をもつ多項式環のイデアル $\langle Plist \rangle$ の簡約グレブナー基底を計算する. $Prime$ は 1 を指定しておく. (他の値の場合はマニュアルを参照.) $Homo$ が 1 のとき, 斉次化を経由して計算する. $Homo$ が 0 のとき斉次化を経由しないで計算する. どちらも結果は

[4] 関数名, 用語は, Buchberger の現論文, REDUCE 上の実装, [BW] などで比較的初期に使われていたものを採用しており, 最近のものと異なることに注意されたい.

同一であるが, 中間係数が膨張する可能性がある場合, $Homo = 1$ で計算するのが安全である.

Asir によるグレブナー基底計算

```
[1517] load("cyclic")$
[1527] C=cyclic(7);
[c6*c5*c4*c3*c2*c1*c0-1,...]
[1528] V=vars(C);
[c0,c1,c2,c3,c4,c5,c6]
[1529] nd_gr(C,V,31991,0)$
...
2.303sec + gc : 0.07sec(2.429sec)
[1530] nd_gr(C,V,0,0)$
(5 分待って中断)
[1530] G=nd_gr_trace(C,V,1,1,0)$
...
25.84sec + gc : 7.833sec(34.56sec)
[1531] G[0];
(((2385392266590200071306 62*c6*c4-...
[1532] length(G);
209
```

この例では, cyclic-7 (例 3.3.1) の全次数逆辞書式順序グレブナー基底計算を有限体 $\mathbb{F}_{31991}$ および有理数体上で行っている. 最初の nd_gr は有限体上の計算で, 2 秒程で終っているが, これを有理数体上で行う (2 番目の nd_gr) と, 計算途中で係数膨張のため計算が進まなくなる. 一方で, nd_gr_trace を $Homo = 1$ で実行すると, 25 秒で計算が終了する. この例に限らず, 有理数体上の計算の場合, 常に係数膨張の危険があるため, $Homo = 1$ で計算することを推奨する.

noex:cyclic7

計算結果のリストは, 入力と同一のイデアルを生成するグレブナー基底である. リスト G の i 番目の要素は $G[i]$ (i は 0 から始まる) で取り出せる.

3.7.7 イニシャルイデアルの計算

Asir でイニシャルイデアルを計算するには, まずグレブナー基底を計算し, その各生成元の先頭項を取り出す. このためには, 再帰表現された多項式

を dp_ptod で分散表現に変換し, dp_ht で先頭項を取り出すという操作が必要となる. さらに, 例では dp_dtop により各先頭単項式を再帰表現に逆変換している. また, 0 次元の場合のみ, dp_mbase により, 標準単項式全体を計算できる.

Asir によるイニシャルイデアルの計算

```
[1517] B=[x^2*y^2-z^2,x^3-y*z^2,x^2*z^4-y^2];
[y^2*x^2-z^2,x^3-z^2*y,z^4*x^2-y^2]
[1518] V=[x,y,z]$
[1519]  G=nd_gr(B,V,0,0);
[z^4*x^2-y^2,-y^4+z^6,-y^2*x+y^5,-z^2*x+z^2*y^3,
y^2*x^2-z^2,x^3-z^2*y]
[1520] D=map(dp_ptod,G,V)$ H=map(dp_ht,D)$
[1521] [1522] map(dp_dtop,H,V);
[z^4*x^2,z^6,y^5,z^2*y^3,y^2*x^2,x^3]
[1523] map(dp_dtop,dp_mbase(H),V);
[z^5*y^2*x,z^4*y^2*x,z^5*y*x,z^5*y^2,z*y^4*x,z^3*y*x^2,...]
[1524] length(@@);
52
```

3.7.8　剰余計算

多項式を多項式リストに現れる多項式で割った剰余を計算する関数が p_nf および p_true_nf である. 前者は, 剰余の分母を払って, 整数係数で返すもので, 剰余が 0 か否かの判定に用いる. 後者は, $[num, den]$ なるリストを返す. num は p_nf が返すもので, num/den が真の剰余となる.

剰余計算

```
[1517] B=[u2*u0-2*u2+3,(2*u1-1)*u0^2-u0-2*u2,2*u1^3+u2+4]$
[1518] V=[u0,u1,u2]$
[1519] G=nd_gr(B,V,0,0);
[10*u2^4+126*u2^3+637*u2^2+(586*u1-907)*u2-816*u0^2-...]
[1520] Q=p_nf(u0^5+u1^5+u2^5,G,V,0);
2851262910*u2^3+30078832770*u2^2+(22194374760*u1-...
[1521] QR=p_true_nf(u0^5+u1^5+u2^5,G,V,0);
[2851262910*u2^3+30078832770*u2^2+...,35373600]
```

3.7.9 消去法

nosubsec:elimord
3.4.1 節で述べたように, $K[Z]$ $(Z = X \cup Y, X \cap Y = \emptyset)$ のイデアル I に対する消去イデアル $I_Y = I \cap K[Y]$ の生成系の計算には消去順序によるグレブナー基底を使う. 有理数体上で計算する場合には nd_gr_trace を斉次化ありで使うことをお勧めする. I の消去順序によるグレブナー基底 G から I_Y のグレブナー基底 G_Y を取り出すには, noro_pd.elimination を使う.

次の例では, B が生成するイデアルの $\{u0, u1\} \gg \{u2\}$ なる消去順序によるグレブナー基底を計算し, noro_pd.elimination により $u2$ のみを含む多項式を取り出している.

消去イデアルの計算

```
[1664] B=[u2*u0-2*u2+3,(2*u1-1)*u0^2-u0-2*u2,2*u1^3+u2+4]$
[1665] V=[u0,u1,u2]$
[1666] G1=nd_gr_trace(B,V,1,1,[[0,2],[0,1]])$
[1667] noro_pd.elimination(G1,[u2]);
[8*u2^9+72*u2^8+292*u2^7-2036*u2^6-198*u2^5+20682*u2^4-...]
```

3.7.10 最小多項式の計算

消去法の特別な例として, イデアルと一変数多項式環の共通部分 $I \cap K[z]$ の計算がある. これも前節で述べた一般的な方法で計算できるが, I が 0 次元イデアルの場合, より直接的な方法により $I \cap K[z]$ の単項生成元のみを計算することができる. gr で定義されている minipoly は, 有理数体係数多項式環の 0 次元イデアル I および多項式 f に対し, $m(f) \in I$ を満たすような 0 でない最小次数の多項式 m を計算する.

次の例では, ベンチマーク問題 katsura-7 において, $u7$ の最小多項式を minipoly により求めている. minipoly の引数は, 最初の $(G, V, 0)$ で, グレブナー基底と項順序, $u7$ が, 最小多項式を計算したい多項式, t が, 最小多項式の変数の指定である. 最後の変数は, V に現れないものを指定する必要がある. この例を, 消去順序グレブナー基底計算で行ってみれば, その大変さがよくわかる.

最小多項式の計算

```
[1518] load("katsura")$
[1522] B=katsura(7)$
[1523] V=[u0,u1,u2,u3,u4,u5,u6,u7]$
[1524] G=nd_gr_trace(B,V,1,1,0)$
[1525] minipoly(G,V,0,u7,t)$
[1526] deg(@@,t);
128
```

3.7.11　0 次元イデアルの項順序変換

gr で定義されている tolex は, 0 次元イデアルの任意項順序のグレブナー基底を入力として, 辞書式順序のグレブナー基底を計算する関数である. 使用アルゴリズムはアルゴリズム 3.5.3 を改良したもので, 出力の変数順序のみ変更可能である. 次の例は, katsura-7 の辞書式順序グレブナー基底を項順序変換で計算したものである. これを nd_gr_trace で直接辞書式順序で計算することは, 途中の係数膨張の様子からみてほぼ不可能であろう.

noalg:fglm

項順序変換による辞書式順序グレブナー基底の計算

```
[1524] G=nd_gr_trace(katsura(7),V=[u0,u1,u2,u3,u4,u5,u6,u7]
,1,1,0)$
3.27sec + gc : 1.067sec(4.524sec)
[1525] G2=tolex(G,V,0,V)$
316.4sec + gc : 98.52sec(442.8sec)
```

3.7.12　イデアル演算

ここでは noro_pd.rr で定義されている関数によるイデアル演算の例を紹介する[5)]. いずれも, デフォルトでは入力を有理数体上のイデアルとみなして計算するが, オプション mod=p を与えると $\mathbb{F}_p$ 上の計算を行う.

[5)] これらは, 2011 年現在開発中のパッケージにある関数であり, 必ずしも最良のものとはいえない. あくまで実装例として紹介するものである.

イデアルの共通部分

2つのイデアルの共通部分は noro_pd.ideal_intersection, イデアルのリストに対して共通部分を求めるには noro_pd.ideal_list_intersection を用いる. いずれも, 変数リストと項順序を引数として与える必要がある.

イデアルの共通部分

```
[1640] B=[g*a-f*b,h*b-g*c,i*c-h*d,j*d-i*e,l*f-k*g,m*g-l*h,
       n*h-m*i,o*i-n*j]$
[1708] V=[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o]$
[1709] G=nd_gr(B,V,0,0)$
[1710] PD=noro_pd.syci_dec(B,V)$
[1711] length(PD);
4
[1712] map(length,PD);
[10,5,3,1]
[1713] for(I=0,T=[1];I<4;I++)
  for(J=0,L=length(PD[I]);J<L;J++)
    T=noro_pd.ideal_intersection(T,PD[I][J][0],V,0);
[1649] gb_comp(T,G);
1
```

この例では, noro_pd.syci_dec によりあるイデアルを準素分解し (3.8 節参照), 得られた準素成分の共通部分がもとのイデアルに等しいことをチェックしている. 準素分解の結果は $[PD_0, PD_1, PD_2, PD_3]$ なるリストで, 各 PD_i は $[Q_{ij}, P_{ij}]$ なるペアのリストである. ここで Q_{ij} は P_{ij}-準素イデアルである. この結果から $PD[I][J][0]$ すなわち Q_{ij} の共通部分をとって入力イデアルのグレブナー基底 G と比較している.

根基所属判定

根基所属判定は, 3.4.3 節の通り, $R[t]I + \langle tf - 1\rangle$ のグレブナー基底を計算してもよいし, noro_pd.radical_membership を用いてもよい. 後者の場合, $f \in \sqrt{I}$ ならば 0 が返り $f \notin \sqrt{I}$ ならば $R[t]I + \langle tf - 1\rangle$ の全次数逆辞書式順序に関するグレブナー基底および, t を表す不定元からなるリストが返される.

根基所属判定

```
[1665] B=[(x+y+z)^50,(x-y+z)^50]$
[1666] V=[x,y,z]$
[1667] F=y$
[1668] cputime(1)$
0sec(1.907e-06sec)
[1669] noro_pd.radical_membership(F,B,V);
0
0.2267sec(0.2502sec)
[1670] nd_gr(cons(t*F-1,B),cons(t,V),0,0);
[1]
0.21sec(0.285sec)
```

イデアル商, saturation

多項式によるイデアル商は noro_pd.colon, イデアルによるイデアル商は noro_pd.ideal_colon を用いる. saturation は, 多項式による saturation を計算する noro_pd.sat およびイデアルによる saturation を計算する noro_pd.ideal_sat が実装されている.

イデアル商, saturation

```
[1640] B=[(x+y+z)^50,(x-y+z)^50]$
[1641] V=[x,y,z]$
[1642] noro_pd.sat(B,y,V);
[1]
[1643] noro_pd.colon(B,y^98,V);
[-x-z,-y]
[1644] noro_pd.ideal_colon(B,[(x+y+z)^49,(x-y+z)^49],V);
[-y^49*x-z*y^49,-y^50,-x^2-2*z*x+y^2-z^2]
[1645] noro_pd.ideal_sat(B,[(x+y+z)^49,(x-y+z)^49],V);
[1]
```

nosec:M2program

3.8　Macaulay2 によるプログラミングの例

ここまで, Macaulay2, SINGULAR, CoCoA, Risa/Asir を例にとって, グ

レブナー基底に関連する基本的な計算の実行法を説明してきたが, いずれのシステムを使うにせよ, ある程度本格的なアルゴリズムを実装してみてはじめてそのシステムの特徴が体感できる場合も多い. 本節では, 最近発見されたイデアルの準素分解アルゴリズムを Macaulay2 に実装してみる. Macaulay2 初心者の体験的入門として読んでいただければ幸いである.

3.8.1 イデアルの準素分解

R を体 K 上の n 変数多項式環 $K[X] = K[x_1, \ldots, x_n]$ とする.

定義 3.8.1. 1. R のイデアル P が素イデアルとは

$$fg \in P \text{ ならば } f \in P \text{ または } g \in P$$

が成り立つことをいう.

2. R のイデアル Q が準素イデアルとは

$$fg \in Q \text{ ならば } f \in Q \text{ または } g \in \sqrt{Q}$$

が成り立つことをいう.

3. Q が準素なら $P = \sqrt{Q}$ は素イデアルである. このとき Q は P-準素であるといい, P を Q の付属素イデアルと呼ぶ.

nothm:primdec

定理 3.8.2. 1. R の真のイデアル I は有限個の準素イデアル $Q_1, \ldots, Q_l$ により $I = Q_1 \cap \cdots \cap Q_l$ と表される.

2. 1. において, どの $\sqrt{Q_i}$ も相異なり, どの Q_i も省けない, すなわち $\bigcap_{j \neq i} Q_j \not\subset Q_i$ $(i = 1, \ldots, l)$ のとき

$$\{\sqrt{Q_1}, \ldots, \sqrt{Q_l}\} = \{\sqrt{I : f} \mid f \in R, \sqrt{I : f} \text{ は素イデアル } \}.$$

特にこの集合は I により一意的に定まる.

定義 3.8.3. 定理 3.8.2 の 1. の表示を I の準素分解と呼び, 各 Q_i を準素成分と呼ぶ. さらに 2. の条件が満たされるとき, 最短準素分解とよび, 素イデアル $\sqrt{Q_i}$ を I の付属素イデアルと呼ぶ. 付属素イデアルのうち, 包含関係

に関して極小なものを極小付属素イデアルと呼ぶ. 極小付属素イデアルに対応する準素成分を孤立準素成分と呼ぶ. それ以外の準素成分を埋没準素成分と呼ぶ.

一般に Q_1, Q_2 が P-準素なら $Q_1 \cap Q_2$ も P-準素である. よって, 準素分解が任意に与えられれば, 同じ付属素イデアルをもつ成分をひとまとめにして, 包含関係に関して冗長な成分を除けば最短準素分解が得られる. イデアル I が根基イデアル, すなわち $\sqrt{I} = I$ のとき, その最短準素分解は素イデアルからなる. また, 次が成り立つ.

nothm:isocomp

定理 3.8.4.
R の真のイデアル I の最短準素分解において, 孤立準素成分全体は集合として一意的である.

I の準素分解を計算するアルゴリズムはいくつか知られており, これまで説明してきたシステムにも実装されている. 最近, これらとは異なるアルゴリズムが発見され, 既存のアルゴリズムでは分解が困難な例も分解できることが示された. 以下で, このアルゴリズムについて簡単に解説し, さらにMacaulay2 に実装してみよう.

3.8.2　SYCI アルゴリズム

ここでは, イデアル I に対し, $\sqrt{I}$ の素イデアル分解を計算するアルゴリズム $MinimalAssociatedPrimes(I)$ が与えられていると仮定し, これまで説明してきたさまざまなツールを用いて, あるアルゴリズム (SYCI アルゴリズム) を与える. $MinimalAssocicatedPrimes(I)$ は, nosubsec:radical 3.4.5 節で説明した根基計算において, 0 次元イデアルの根基を計算する部分を, その素イデアル分解に置き換えれば得られる. さらに, 0 次元根基イデアルの素イデアル分解は, 多項式の既約因子分解により実現できる.

定義 3.8.5. $I \subset Q$ なるイデアル I, Q に対し, $I = Q \cap (I+J)$ かつ $\sqrt{I:Q} = \sqrt{I+J}$ を満たすイデアル J を (I, Q) に対する saturated separating ideal と呼ぶ.

$\sqrt{I:Q} = \langle f_1, \ldots, f_l \rangle$ とするとき十分大きな m に対する $(\sqrt{I:Q})^m$ あるいは $\langle f_1^m, \ldots, f_l^m \rangle$ などが saturated separating ideal となるが, これらは m のベキが大きくなり過ぎる場合が多く, それを用いた計算の効率を落とす原因になるので次のアルゴリズムを用いる.

noalg:ssi

アルゴリズム 3.8.6 ($SaturatedSeparatingIdeal(I, Q, C)$)**.**
入力 :$I \subset Q$, $\sqrt{C} = \sqrt{I:Q}$ を満たすイデアル I, Q, C
出力 :(I, Q) に対する saturated separating ideal J
$S \leftarrow C$ の生成系
$S_0 = \{f_1, \ldots, f_l\} \leftarrow S \setminus \sqrt{I}$
$J = \{0\}$
for $i = 1$ to l do
  $j \leftarrow 0$
  do $j \leftarrow j + 1$ while $Q \cap (I + J + \langle f_i^j \rangle) \neq I$
  $J \leftarrow J + \langle f_i^j \rangle$
end for
return J

証明は省略するがこのアルゴリズムは停止する. 停止すれば J が saturated separating ideal となることは明らかであろう.

定義 3.4.14 (nodef:extcont) において, $X = \{x_1, \ldots, x_n\}$ の部分集合 U に対し, $K[X]$ のイデアル I を $K(U)[X \setminus U]$ に持ち上げて $K[X]$ に戻す操作 I^{ec} を定義した. この操作を, U を明示して I_U^{ec} と書くことにする.

noalg:sycidec

アルゴリズム 3.8.7 ($PrimaryDecompositionSYCI(I)$)**.**
入力 :イデアル $I \subset R$
出力 :I の準素分解 $(QL_1, QL_2, \ldots, QL_l)$
  $QL_i = (Q_{i1}, \ldots, Q_{in_i})$; Q_{ij} は I の準素成分
$i \leftarrow 1$; $Q_0 \leftarrow R$
do
  $PL_i = \{P_{i1}, \ldots, P_{in_i}\} \leftarrow MinimalAssociatedPrimes(I : Q_{i-1})$
  for $j = 1$ to n_i do

$U_{ij} \leftarrow P_{ij}$ に対する極大独立集合
$f_{ij} \leftarrow (\bigcap_{k \neq j} P_{ik}) \setminus P_{ij}$ の要素
$R_{ij} \leftarrow Q_{i-1} \cap (I : f_{ij}^{\infty})^{ec}_{U_{ij}}$
$J_{ij} \leftarrow SaturatedSeparatingIdeal(R_{ij}, Q_{i-1}, P_{ij})$
$Q_{ij} \leftarrow (R_{ij} + J_{ij})^{ec}_{U_{ij}}$
end for
$QL_i = \{Q_{i1}, \ldots, Q_{in_i}\}$
$Q_i \leftarrow R_{i1} \cap \cdots \cap R_{in_i}$
If $Q_i = I$ then return $(QL_1, \ldots, QL_i)$
$i \leftarrow i + 1$
end do

アルゴリズム 3.8.7 noalg:sycidec について以下が成り立つ.

1. PL_i は, I の付属素イデアル全体から $PL_1, \ldots, PL_{i-1}$ を除いたものの中で包含関係に関して極小なもの全体からなる. 付属素イデアル P が $P \in PL_i$ を満たすとき, P はレベル i であるという.
2. Q_{ij} は I の P_{ij}-準素成分である. 1. より, このアルゴリズムは最短準素分解を与える.
3. Q_i は, レベル i 以下の付属素イデアルに対応する準素成分全体の共通部分であり, Q_i は最短準素分解に依存しない.
4. Q_{ij} を P_{ij}-準素成分とすれば $R_{ij} = Q_{i-1} \cap Q_{ij}$ であり, この R_{ij} も最短準素分解に依存せず, P_{ij} のみで定まる.
5. J_{ij}, Q_{ij} の計算を省いても, PL_i は計算できる. この場合は, I の付属素イデアル全体を計算するアルゴリズムとなる.

これらの証明は省くが, 用いられる個々の計算はすべてこれまで説明したものである. よって, これらを計算する機能が備わっているシステム上では原理的にこれらのアルゴリズムが実装可能である.

3.8.3 Macaulay2 上での実装

新しいコマンド, 関数を紹介しながら, 必要となるサブルーチンを定義していく.

最小次数の多項式を取り出す

```
mindeg = (L) -> (
  f := L#0; df := degree f;
  scan(L, g -> if degree g < df then (f = g; df = degree g));
  return f
)
```

リスト L の中で全次数最小の多項式を選んで返す. この例のように, 関数は $(arg_1, \ldots, arg_l)$ -> $(e_1; \ldots; e_m)$ の形で与える. これに = で関数名を与える. e_i は式であり, 最後の式の値が関数の値になる. あるいは return によりそこで関数実行を終了して明示的に値を返すこともできる. scan($list, function$) は, $list$ の各要素に $function$ を適用するコマンドで, C 言語などの for 文に相当する. $function$ としては, 既に定義されている関数も使えるが, この例のように, 即席の名無しの関数を与えることもできる. 代入文には a=b と a:=b の二種類がある. 前者は単なる代入である. a はデフォルトで大域変数とみなされるので, 関数内で a を宣言なしに用いると, 大域変数が生成されてそこに代入される. 後者は局所変数 a を生成してそこに b が代入される. := は局所変数を初期化する場合に用いる. 初期化せずに単に宣言したい場合は local で宣言する. リスト L の i 番目の要素は L#i で指定する. i は 0 から始まる. リストの長さは #L で与えられる.

$f \in P_1 \setminus P_2$ を満たす元を探す

```
nonmember = (P1,P2) -> (
 S := select(first entries gens P1,f->not isSubset(ideal f,P2));
 return mindeg(S)
)
```

イデアル P_1, P_2 に対し $f \in P_1 \setminus P_2$ を満たす元のうちで全次数が最小のものを返す. select($list, function$) は, $function$ を適用した時 true を返すような $list$ の要素のみを取り出してリストにして返すコマンドである.

entries は行列の各列をリストとしたもののリストを返す. イデアルの場合, gens は行ベクトル (1 行 m 列の行列) を返すので, entries は要素がひとつのリストであり, first はその先頭, すなわち生成系のリストを返すことになる.

無平方部分の計算

```
squarefree = (f) -> value apply(factor f, i -> i#0)
```

組み込みで用意されていてもおかしくない機能だが, 見つからないのでライブラリを参考にして書いた. factor は多項式を既約分解した結果を $Product$ という型のオブジェクトで返す. これは (因子, 重複度) を要素とするリストの一種である. apply($list, function$) は, $list$ の各要素に $function$ を適用した結果をリストにして返す. i->i#0 は, リストの 0 番目の要素 (先頭要素) を取り出すという意味なので, 今の場合は因子のリストが得られることになるが, 入力リストが $Product$ 型なので, apply も結果を同じ型で返す. すなわち入力多項式の無平方部分が分解された形で得られる. value はそれを展開する.

separator の計算

```
separator = (I,PP) -> (
  R := ring I;
  S := new MutableList;
  scan(toList(0..#PP-1),i->S#i=1_R);
  scan(toList(0..#PP-1),i->
     scan(toList(0..#PP-1),j->
       if i != j then
         S#j =  lcm(S#j,squarefree(nonmember(PP#i,PP#j))))
  );
  return S
)
```

どの 2 つも包含関係を持たないような素イデアルのリスト $PP = \{P_1, \ldots, P_l\}$ に対し $f_i \in (\bigcap_{j \neq i} P_j) \setminus P_i$ を満たす $\{f_1, \ldots, f_l\}$ を返す. これらは separator と呼ばれる. $i \neq j$ なる i, j に対し $s_{ij} \in P_i \setminus P_j$ が求まれば, f_i として

$\prod_{j\neq i} s_{ij}$ が使えるが, その無平方部分でもよいので, 積の代わりに squarefree と lcm により順次計算している. このためには, f_i を入れる書き換え可能な入れ物を用意して新しい s_{ij} が計算できるごとに値を更新していくのがよい. このような目的のために $MutableList$ という型が用意されている.

包含関係に関して極小なイデアルの計算

```
removeRedundantComps = (L) -> (
  if #L == 1 then return L;
  S := new MutableList from L;
  for i from 0 to #L-1 do (
     if S#i == 0 then continue;
     for j from 0 to #L-1 do (
         if j == i or S#j == 0 then continue;
         if S#i == S#j or isSubset(S#i,S#j) then S#j = 0
       )
  );
  return toList(select(S,i -> i!=0))
)
```

イデアルのリストを入力として, 包含関係に関して極小な要素のみをリストとして返す. ここでも $MutableList$ を入れものとして用意し, 入力リストで初期化しておき, 他のイデアルを含むものを 0 で置き換えるという操作を全体に適用したあと, 0 にならずに残ったイデアルのみを $List$ 型のオブジェクトとして返す. Macaulay2 では, リストに多くの種類があるが, toList により最も “普通” のリストに変換できる.

$\sqrt{I:J}$ の素イデアル分解

```
colonMinimalPrimes = (I,J) -> (
   local K,PL,S;
   R := ring I; L := {};
   for f in first entries mingens J do (
     if f==1 then K=I else K=I:f;
     if K != ideal(1_R) then L = append(L,K)
   );
   L = removeRedundantComps(L); P := {};
   for K in L do (
     S = apply(first entries gens K,f->squarefree(f));
     PL = minimalPrimes ideal mingens gb ideal S;
     P = join(PL,P)
   );
   return removeRedundantComps(P)
)
```

$J = \langle f_1, \ldots, f_l \rangle$ のとき $I : J = \bigcap_{i=1}^{l} (I : f_i)$ である．$f_1, \ldots, f_l$ は生成系でさえあればよいので, `mingens` を用いて生成元の個数をなるべく少なくする．得られた $I : f_i$ のリスト L に `removeRedundantComps` を適用して素イデアル分解を行うイデアルを減らす．`minimalPrimes` は極小付属素イデアルのリストを計算する Macaulay2 で既に定義されているコマンドであるが, 多少でも効率が上がるように, 無平方化を行ったのちに適用する．得られた素イデアル全体にもう一度 `removeRedundantComps` を適用して冗長性を省く．

saturated separating ideal の計算

```
saturatedSeparatingIdeal = (C,I,Q,Rad) -> (
  local fi;
  S := select(first entries gens C, f->not isSubset(ideal f,I));
  if intersect(I+ideal S,Q) == I then return S;
  I1 := I;
  SSI := {};
  for f in S do (
    fi = f;
    while (intersect(Q,I1+ideal fi) != I) do (fi=fi*f);
    I1 = I1+ideal fi;
    SSI=append(SSI,fi)
  );
  return ideal SSI
)
```

これはアルゴリズム 3.8.6 noalg:ssi をほぼそのまま実装している. $C \setminus \sqrt{I}$ を計算するため $Rad = \sqrt{I}$ を引数として要求している.

I_U^{ec} の計算

```
load "PrimaryDecomposition/Shimoyama-Yokoyama.m2";
myextract = (I,Y) -> (
  R := ring I;
  if #Y == 0 then f = 1_R
  else f := flattener(I,Y#0);
  if f != 1_R then return saturate(I,f)
  else return I
)
```

I の $K(U)[X \setminus U]$ におけるグレブナー基底 $G \subset K[U][X \setminus U] = K[X]$ の各元の先頭係数 ($K[U]$ の元) の LCM f が必要となるが, これはライブラリファイル `Shimoyama-Yokoyama.m2` で定義されている `flattener` を用いる. 求める I_U^{ec} は $I : f^\infty$ に等しい.

準素分解

```
sycidec = (I) -> (
  local PLi,QLi,RLi,Si,Ci,Yi,Ti;
  R := ring I; Qi := ideal(1_R); QL := {};
  for i from 1 do (
    PLi = colonMinimalPrimes(I,Qi);
    Si = separator(I,PLi);
    Ci = apply(Si,f->saturate(I,f));
    Yi = apply(PLi,P->independentSets(P,Limit=>1));
    RLi = apply(Ci,Yi,
      (c,y)->intersect(Qi,ideal gens gb myextract(c,y)));
    if i == 1 then ( Rad := intersect(PLi); QLi = RLi )
    else (
      Ti = apply(PLi,RLi,
        (p,r)->r+saturatedSeparatingIdeal(p,r,Qi,Rad));
      QLi = apply(Ti,Yi,(t,y)->ideal gens gb myextract(t,y))
    );
    QL = append(QL,QLi); Qi = intersect(RLi);
    if Qi == I then return QL
  )
)
```

noalg:sycidec 3.8.7

これまで定義した関数を呼び出して計算するだけなので, アルゴリズム 3.8.7 がほぼそのままプログラムになっている．プログラム中の `Qi` はアルゴリズム中の Q_{i-1} に対応する．`independentSets` はイデアルの極大独立集合のリストを返すが, `Limit=>1` とすると要素ひとつのリストを返す．要素は不定元のリストではなく, 不定元の積となっていることに注意する．`saturatedSeparatingIdeal` の第 4 引数は $\sqrt{R_{ij}}$ でなければならないが, 実は $\sqrt{R_{ij}} = \sqrt{I}$ なので $i = 0$ で計算される $Rad = \sqrt{I}$ が毎回共通に使える．ここでは 3 引数の `apply`$(list_1, list_2, function)$ が使われている．この場合, $list_1$, $list_2$ は同じ長さのリストで, 2 引数関数 $function$ がこれらのリストの要素を引数として呼び出される．

以上の関数を含むファイルを `syci.m2` として `load` により読み込めば準備完了である．

実行例

```
i1 : load "syci.m2";
i2 : R=QQ[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o]$
i3 : I=ideal(g*a-f*b,h*b-g*c,i*c-h*d,j*d-i*e,l*f-k*g,m*g-l*h,
       n*h-m*i,o*i-n*j);
o3 : Ideal of R
i4 : timing (p=sycidec(I);)
o4 = -- 1.95582 seconds
i5 : #p
o5 = 4
i6 : apply(p,i->#i)
o6 = {10, 5, 3, 1}
o6 : List
i7 : intersect(apply(p,intersect))==I
o7 = true
i8 : apply(join(p#1,p#2,p#3),isPrimary)
o8 = {true, true, true, true, true, true, true, true, true}
o8 : List
```

この例の場合, 孤立成分が 10 個, 埋没成分が 9 個得られた. 結果が実際に I の分解になっていることを intersect で確かめ, 得られた埋没成分が実際に準素であることを isPrimary で確かめている. この例で, isPrimary を孤立成分を引数として呼び出すと大変時間がかかるので行っていない. sycidec は 2 秒ほどで計算が終了しているが, primaryDecomposition (組み込み関数) だと 20 時間ほどかかる困難な例である.

このように実際にある程度本格的なプログラムを書いてみると, Macaulay2 の設計思想が見えてくる. 数学的な概念, 対象が自然な形で表現できることを目指していることは明らかであるが, それ以外にも以下の点に気づく.

- グレブナー基底をなるべく表に出さずにプログラムが書ける.
  ほとんどの計算がグレブナー基底なしには遂行できないにも関わらず, 引数にグレブナー基底を必要とするのは, イデアルによる剰余などごく一部であり, 通常はイデアルそのものを入力として渡すことになる. 必要な場合には内部でグレブナー基底が計算されるので, グレブナー基底について深い知識がないユーザでも自然にプログラムが書ける. 今回書

いたプログラムでも, 明示的なグレブナー基底計算の指示は $\sqrt{I:J}$ の素イデアル分解の前処理および, myextract の出力を整形するための呼び出しに限られており, しかもこれらは本質的には必要がない.

- 繰り返しをあらわに記述せずに繰り返しが書ける.
  ライブラリにあるファイルなどを参考にプログラムを書こうとすると, C 言語をベースにした言語でのプログラミングに慣れた人ほど面食らうのではないだろうか. これは, 特に繰り返しを書こうとうする場合に顕著である. すなわち, なるべく for, while などを使わずに apply, scan などを使うことを暗に要求しているように感じられる. 実は, ここで例示したプログラムは, 最初, 大抵の繰り返しを for を使って書いていたものを, あえて Macaulay2 風にしてみようと書き直してみた結果である. しかし, 他のプログラミングの経験がないユーザ (数学者) なら, 自然に要求されているスタイルのプログラミングができるのではないだろうか.

結果として, 数学的な意味がある程度明確な, 他の人が読んでも理解できるプログラムを書くことができる. もちろん, 意味が明確で書くのが容易でも, それは必ずしもプログラムの効率のよさを意味しない. システム開発の観点からは, 効率化を意識した指示 (例えば項順序の指定, 消去アルゴリズムの明示など) なしにそれなりの効率で計算が行われるようになっているかが気になるところである. これについては今回の経験で次のようなことが分かった.

1. イデアルの共通部分の計算が高速である.
   マニュアルに詳しい記述はないが, 第 1 章で述べられているアルゴリズムを用いればグレブナー基底計算は 1 回で, 結果はグレブナー基底で得られる (従って生成元が多い) はずだが, intersect ではグレブナー基底が 2 回計算され, 結果は mingens を適用したように見える形で得られる. このことは, 別種のアルゴリズムが適用されていることを意味していると考えられる.
2. イデアル商の計算が高速である.
   これに関してはマニュアルに記述があり, この章で述べた方法とは異な

る, syzygy を用いた方法がデフォルトになっていて, これが高速性の要因のようである.

3. イデアルの極小付属素イデアル計算が遅い
   これは残念な発見であるが, アルゴリズム 3.8.7 の実行でボトルネックとなるのが常に $minimalPrimes$ であった. マニュアルによれば characteristic set を用いた実装のみを提供しているようで, この方法は効率が入力によりムラがあり, 計算が滞ってしまう例があるのはやむを得ない. よって, 今回のプログラムをより実用的にするには, 極小付属素イデアル計算の他の方法を実装する必要がある.

以上のように, 少なくとも今回必要になった計算において大変重要な 1., 2. に関しては, ブラックボックス的に用いても快適に計算できるような工夫がなされていることがわかった. 同時に, 3. のような弱点も見えたが, この機能に関しては, Macaulay2 に限らず他のシステムにおいてもまだ満足な実装がなされてはいないので,[6] よりよい実装に挑戦してみる動機になるのではないか.

3.9 章末問題

1. $\mathbb{Q}[x,y,z,u]$ のイデアル H, I, J を $H = \langle 2zx+3zy, 2x^2u+zx^2+zx, zxu-2zx\rangle$, $I = \langle 2zx^2+3zyx, 4zx^2-9zy^2, 2x^3u+zx^3+zx^2, -zx^2u+2zx^2, 2x^3u+3yx^2u\rangle$, $J = \langle y^2, x^3, 4x^2u-3zy, zx^2, yx^2, zxu+3zy, z^2u\rangle$ で定義する.
   a. H, I, J の包含関係を調べよ.
   b. $V(H)$, $V(I)$, $V(J)$ の包含関係を調べよ.
2. $\mathbb{Q}[x,y,z]$ のイデアル I を $I = \langle 3x^2yz^2+3z+(-2x+2)y+2x, 3yz^5+(-xy^2+2)z-2y^4+2y, xy^3z^3-2yz^2-z-2y+x^2\rangle$ で定義する.
   a. I が 0 次元イデアルであることを示せ. (例えば, $\mathrm{in}_<(I)$ に入らない単項式が有限個であることを示す, あるいは $I\cap\mathbb{Q}[x]$, $I\cap\mathbb{Q}[y]$, $I\cap\mathbb{Q}[z]$

[6] SINGULAR には characteristic set 以外の方法による実装もあり選択できるが, やはり計算が滞る場合がよくある.

を実際に計算する, など.)

b. $\dim_{\mathbb{Q}} \mathbb{Q}[x,y,z]/I$ を求めよ.

c. I の $x > y > z$ なる辞書式順序での簡約グレブナー基底が $\{g_0(z), x - g_1(z), y - g_2(z)\}$ という形であることを確かめよ.

3. $\alpha = 3^{\frac{1}{5}}$, $\beta = 5^{\frac{1}{3}}$ とする.

a. $\alpha + \beta$ の $\mathbb{Q}$ 上の最小多項式を求めよ. ($\mathbb{Q}[x,y,z]$ のイデアル $I = \langle x^5 - 3, y^3 - 5, z - (x+y)\rangle$ に対し, $I \cap \mathbb{Q}[z]$ を求める.)

b. $\frac{1}{\alpha+\beta}$ を α, β の有理数係数多項式で表せ. ($\mathbb{Q}[x,y,t]$ のイデアル $J = \langle (x+y)t - 1, x^5 - 3, y^3 - 5\rangle$ の $t\{x,y\}$ なる消去順序に関するグレブナー基底を求める.)

4. 本章で説明した各システム上で, 以下の各項を行う方法を調べよ.

a. 実行中断および続行, 復帰.

b. 繰り返しおよび途中脱出.

c. リストの操作 (生成, 結合, 要素の取り出しなど).

d. 多項式の GCD, 因数分解.

e. 関数定義.

5. アルゴリズム 3.8.7 を SINGULAR 上で記述せよ.

3.10　問題の略解

3.10.1　本文中の問題

問題 **3.1.7**

定理 1.3.3 (1.5) の右辺の $S(g_j, g_k)$ を, S' に属する S_{pq} から作られる S-多項式の単項式係数の線形和に置き換えることができる. あとは, それらの S-多項式に対する (1.6) の表示により, 定理の証明がそのまま適用できる.

問題 **3.1.11**

$k < i < j$ かつ $T_{jk} = T_{ij}$ とすると, $T_k \mid T_{ij}$ より $T_{ijk} = T_{ij}$. よって $S_{ij} = S_{jk} + \frac{T_{ij}}{T_{ki}} S_{ki}$. 定義により $S_{jk} \prec S_{ij} \Leftrightarrow T_{jk} < T_{ij}$ または ($T_{jk} = T_{ij}$ かつ ($k < j$ または ($k = j$ かつ $j < i$))) であるが, $T_{jk} = T_{ij}$ かつ $k < j$ だ

から $S_{jk} \prec S_{ij}$. $S_{ki} \prec S_{ij} \Leftrightarrow T_{ki} < T_{ij}$ または $(T_{ki} = T_{ij}$ かつ $(i < j$ または $(i = j$ かつ $k < i)))$ であるが, $T_{ki} \mid T_{ij}$ より $T_{ki} \leq T_{ij}$ であり, $i < j$ だから $S_{ki} \prec S_{ij}$. よって S_{ij} が最大順位となる. 他の場合も同様に確かめられる.

問題 noprob:homogb
3.1.19

略.

問題 noprob:ratgb
3.4.3

J の $<$ に関するグレブナー基底を $G \subset K[X,Y]$ とすれば, 明らかに $G \subset I$ である. $f \in I$ とすると, $h \in K[Y]$ が存在して $hf \in J$. よって, $g \in G$ が存在して $\mathrm{in}_<(g) \mid \mathrm{in}_<(hf)$. $<$ が $X \gg Y$ なる消去順序だから $\mathrm{in}_<(g) = t_X t_Y$, $\mathrm{in}_<(hf) = s_X s_Y$ $(t_X, s_X \in K[X]$, $t_Y, s_Y \in K[Y])$ とすれば $t_X \mid s_X$ かつ t_X, s_X はそれぞれ $K(Y)[X]$ における $<|_{K[X]}$ に関する g, f の先頭項である. よって G は I の $<|_{K[X]}$ に関するグレブナー基底となる.

問題 noprob:rad
3.4.6

$f \in \sqrt{I}$ ならばある正整数 m が存在して $f^m \in I$. このとき $1 = (1-t^m f^m)+t^m f^m = (1-tf)(t^{m-1}f^{m-1}+\cdots+1)+t^m f^m \in \langle 1-tf\rangle + R[t]I$. 逆に $1 \in R[t]I + \langle 1-tf\rangle$ とすればある $f_1, \ldots f_l \in I, a_1(t), \ldots, a_l(t) \in R[t]$, $b(t) \in R[t]$ が存在して $1 = a_1(t)f_1 + \cdots + a_l(t)f_l + b(t)(1-tf)$. ここで, $t = 1/f$ を代入すれば R の商体における等式 $1 = a_1(1/f)f_1 + \cdots + a_l(1/f)f_l$ を得る. 両辺に, $f^m a_1(1/f), f^m a_l(1/f) \in R$ となるような f のベキをかければ R における等式 $f^m = (f^m a_1(1/f))f_1 + \cdots (f^m a_l(1/f))f_l$ を得る. よって $f^m \in I$.

問題 noprob:sat
3.4.11

前問とほとんど同様に示せるので略.

問題 **3.4.16** noprob:extcont

$h \in J^c$ とすれば, $h \in J$ より h を G で割った余りは 0. このとき $h = a_1g_1 + \cdots + a_lg_l$ $(a_1, \ldots, a_l \in K(U)[X \setminus U])$ と書けるが, 除算の定義により $a_1, \ldots, a_l$ の分母はすべて $h_1, \ldots, h_l$ のベキ積となる. よって十分大きい m をとれば $f^m h \in \tilde{J}$. よって $h \in \tilde{J} : f^\infty$. 逆に $h \in \tilde{J} : f^\infty$ ならばある m が存在して $f^m h \in \tilde{J}$. よって $h \in J$ かつ $h \in K[X]$ より $h \in J^c$.

問題 **3.4.21** noprob:satdecomp

I が右辺に含まれるのは明らかである. $h \in I : f^s$ かつ $h \in I + \langle f^s \rangle$ とすると $hf^s \in I$ かつある $a \in I, b \in R$ が存在して $h = a + bf^s$. このとき $hf^s = af^s + bf^{2s} \in I$ より $bf^{2s} \in I$. よって $b \in I : f^{2s}$ が成り立つ. ここで $I : f^{2s} = I : f^\infty = I : f^s$ より $b \in I : f^s$. よって $bf^s \in I$ が成り立ち, $a + bf^s \in I$ すなわち $h \in I$.

問題 **3.4.23** noprob:radsatdecomp

$I^{ec} = I : f^\infty = I : f^s$ を満たす s をとると, 定理 3.4.20 nothm:satdecomp より $I = (I : f^\infty) \cap (I + f^s) = I^{ec} \cap (I + f^s)$. よって $\sqrt{I} = \sqrt{I^{ec}} \cap \sqrt{I + f^s}$. $\sqrt{I^{ec}} = \sqrt{I^e \cap R} = \sqrt{I^e}^c$ および $\sqrt{I + f^s} = \sqrt{I + \langle f \rangle}$ より $\sqrt{I^{ec}} \cap \sqrt{I + f^s} = \sqrt{I^e}^c \cap \sqrt{I + \langle f \rangle}$.

問題 **3.6.5** noprob:intbymod

まず $G_0 \subset I \cap J$ を示す. $h \in G_0$ に対し, $\begin{pmatrix} 0 \\ h \end{pmatrix} \in M$ より $\begin{pmatrix} 0 \\ h \end{pmatrix} = \sum_i c_i \begin{pmatrix} f_i \\ 0 \end{pmatrix} + \sum_j d_j \begin{pmatrix} g_j \\ g_j \end{pmatrix}$ とかける. このとき $\sum_i c_i f_i = -\sum_j d_j g_j$ かつ $h = \sum_j d_j g_j$ だから $h \in J$ かつ $h \in I$. 次に, G_0 が $I \cap J$ の $<$ に関するグレブナー基底であることを示す. $h \in I \cap J$ とすると, $h = \sum_i c_i f_i = \sum_j d_j g_j$ とかける. この c_i, d_j に対し, $\sum_i c_i \begin{pmatrix} f_i \\ 0 \end{pmatrix} - \sum_j d_j \begin{pmatrix} g_j \\ g_j \end{pmatrix} = \begin{pmatrix} 0 \\ -\sum_j d_j g_j \end{pmatrix} = \begin{pmatrix} 0 \\ -h \end{pmatrix}$ より $v = \begin{pmatrix} 0 \\ h \end{pmatrix} \in M$. よって $w = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} \in G$ が存在して

$\mathrm{in}_{<_{\mathrm{POT}}}(w)|\mathrm{in}_{<_{\mathrm{POT}}}(v)$. $<_{POT}$ の定義より $h_1 = 0$ だから $h_2 \in G_0$ で, $\mathrm{in}_<(h_2) \mid \mathrm{in}_<(h)$ より G_0 は $I \cap J$ のグレブナー基底である.

問題 noprob:syz2
3.6.13

前問とほぼ同様に示せるので略.

3.10.2 章末問題

問題 1

a. それぞれグレブナー基底を計算して包含関係を確かめると $I \subset J$, $I \neq J$ $I \subset H$, $I \neq H$, J と H に包含関係はないことが分かる.

b. a. より $V(J) \subset V(I)$, $V(H) \subset V(I)$ が分かる. さらに根基所属を調べることにより $V(I) = V(H)$, $V(H) \neq V(J)$ も分かる.

問題 2

a. I の $x > y > z$ なる全次数逆辞書式順序 $<$ でのグレブナー基底には, 先頭項が x^7, y^7, z^7 であるような元が含まれるので I は 0 次元である.

b. $\mathrm{in}_<(I) = \langle x^7, y^6x, y^7, zy^6, z^2y^5, z^3x^4, z^4x^3, z^7, yx^5, y^2x^4, y^4x^2, zx^5, zyx^4, z^3y^2x, z^3y^3, z^4yx, z^5x, z^5y, zy^3x, z^2yx^2\rangle$ で, これに含まれない単項式は $\{x^6, y^3x^3, y^5x, y^6, zy^2x^3, zy^5, z^2x^4, z^2y^4, z^3x^3, z^4x^2, z^4y^2, z^6, x^5, yx^4, y^2x^3, y^3x^2, y^4x, y^5, zx^4, zyx^3, zy^2x^2, zy^4, z^2x^3, z^2y^2x, z^2y^3, z^3x^2, z^3yx, z^3y^2, z^4x, z^4y, z^5, x^4, yx^3, y^2x^2, y^3x, y^4, zx^3, zyx^2, zy^2x, zy^3, z^2x^2, z^2yx, z^2y^2, z^3x, z^3y, z^4, x^3, yx^2, y^2x, y^3, zx^2, zyx, zy^2, z^2x, z^2y, z^3, x^2, yx, y^2, zx, zy, z^2, x, y, z, 1\}$ の 66 個である. よって $\dim_{\mathbb{Q}} \mathbb{Q}[x,y,z]/I = 66$.

c. 実際に計算してみれば $\{g_0(z), x - g_1(z), y - g_2(z)\}$ なる形のグレブナー基底を持つことが分かる. ただし, g_1, g_2 の係数はある程度大きくなるので, 計算のしかたによっては大変時間がかかる場合があるので注意すること.

問題 3

a. $\alpha + \beta$ の最小多項式は, $\alpha^5 = 3$, $\beta^3 = 5$ という関係式のみを用いて示すことができる $m(\alpha + \beta) = 0$ という有理数係数の関係式のうち $m(z)$ の次数が最小であるものと考えれば, $I = \langle x^5 - 3, y^3 - 5, z - (x + y)\rangle$ に対し

$I \cap \mathbb{Q}[z] = \langle m(z) \rangle$ を満たす $m(z) \in \mathbb{Q}[z]$ を求めればよいことになる. 結果だけ示す:
$m(z) = z^{15} - 25z^{12} - 9z^{10} + 250z^9 - 1350z^7 - 1250z^6 + 27z^5 - 10125z^4 + 3125z^3 - 1350z^2 - 5625z - 3152.$
b. $\mathbb{Q}(\alpha, \beta) = \mathbb{Q}[\alpha, \beta]$ より, $\frac{1}{\alpha+\beta}$ は α, β の $\mathbb{Q}$ 係数多項式 $g(\alpha, \beta)$ として表すことができる. このとき, $(x + y)g(x, y) \equiv 1 \bmod \langle x^5 - 3, y^3 - 5 \rangle$ が成り立つ. すると, イデアル $J = \langle (x + y)t - 1, x^5 - 3, y^3 - 5 \rangle$ に対して $t - g(x, y) \in J$ より J の $t \gg \{x, y\}$ なる消去順序に関するグレブナー基底は $t - g(x, y)$ を含む. 計算結果は
$g(x, y) = \frac{1}{3152}((-15y^2 + 125y + 9)x^4 + (-125y^2 - 9y + 75)x^3 + (9y^2 - 75y + 625)x^2 + (75y^2 - 625y - 45)x + 625y^2 + 45y - 375).$

問題 4

略.

問題 5

略.

参考文献

NY [1] 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).

GM [2] R. Gebauer, H. M. Möller, On an installation of Buchberger's algorithm, Journal of Symbolic Computation 6, 275-286 (1988).

SUGAR [3] A. Giovini, T. Mora, G. Niesi, L. Robbiano, C. Traverso, "One sugar cube, please" or selection strategies in the Buchberger algorithm, Proc. ISSAC 1991, ACM Press, 49-54 (1991).

BW [4] T. Becker, V. Weispfenning, Gröbner Bases, Springer (1993).

AL [5] W. Adams, P. Loustaunau, An Introduction to GröbnerBases. Graduate Studies in Mathematics, Vol. 3, AMS (1994).

UAG [6] D. Cox, J. Little, D. O'Shea, Using Algebraic Geometry. GTM Vol. 185, Springer (2005).

MAC2 [7] 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).

SINGULAR [8] G.-M. Greuel, G. Pfister, A Singular Introduction to Commutative Algebra. Springer (2002).

COCOA [9] M. Kreuzer, L. Robbiano, Computational Commutative Algebra 1. Springer (2000), Computational Commutative Algebra 2. Springer (2005).