Yang Tutorial

Katsuyoshi OHARA

平成 17 年 11 月 18 日

1 yang とは

yang ではオイラー微分演算子, shift operator, q-shift operator からなる環での計算を行う Risa/Asir のパッケージです. 計算する前に `yang.define_ring` あるいはその変種を用いて, 必ず環を定義します. 同時に扱える環はひとつだけですが, `yang.define_ring` を呼び出すと, 以前の環の定義はスタックにプッシュされるため, `yang.define_ring` と `yang.pop_ring` で挟むことで, サブルーチン的な計算を実現することができます.

yang でできる計算は, グレブナ基底, 正規形, 0 次元イデアルのランク, Pfaff 形式などです. またグレブナ基底は有理関数体係数で計算します.

2 Appell's F_1 を計算してみる.

ここでは, オイラー微分演算子からなる環を定義し, 超幾何方程式系 F_1 のグレブナ基底を計算してみます. 実はオイラー微分演算子のみを含む場合には, `yang_D.rr` を使ったほうが高速になります.

```
ohara:~> asir
[1] load("yang.rr");
[2] yang.define_ring([x1,x2]);
```

環として, $R = K(x_1, x_2)\langle\theta_1, \theta_2\rangle$ が定義されました ($\theta_i = x_i\partial_i$). 環の元を定義しましょう.

```
[3] S1=yang.operator(x1);
<<1,0>>
[4] S2=yang.operator(x2);
<<0,1>>
```

$S_1 = \theta_1, S_2 = \theta_2$ となります.

演算子の内部表現は分散多項式になっていますので, そのままでは $R' = K[x_1, x_2]\langle\theta_1, \theta_2\rangle$ の元しか表せず, R の元は表現できません. したがって, $f \in R'$ と $q \in K[x_1, x_2]$ の対 `[F,Q]` で R の元 $(1/q)f$ を表します.

さらに, R における項順序は asir の分散多項式の項順序に一致します. 既定値は, 全次数逆辞書式順序です. 順序を変更して Groeber 基底を計算したい場合には, 環の元を定義する前に dp_ord で順序を変更しておくべきです.

```
[5] S=S1+S2;
<<0,1>>+<<1,0>>
```

$S = S_1 + S_2$ です. R' における和は, 通常の $+$ で書くことができます.

```
[6] L1 = yang.multi(S1,S+c-1) - x1*yang.multi(S1+b1,S+a);
[7] L2 = yang.multi(S2,S+c-1) - x2*yang.multi(S2+b2,S+a);
```

$L_1 = S_1(S + c - 1) - x_1(S_1 + b_1)(S + a)$ です. R' における演算子の積は `yang.multi` で計算します. ただし, $K[x_1, x_2]$ の元はそのままかけても構いません. いまのところ, `yang.multi` の引数に使えるのは R' の元のみです.

```
[8] G = yang.buchberger([L1,L2]);
[[(-x2^2+(x1+1)*x2-x1)*<<0,2>>+((b2*x1-b2)*x2)*<<1,0>>
+((-a-b2)*x2^2+((a-b1+b2)*x1+c-1)*x2+(-c+b1+1)*x1)*<<0,1>>
+(-b2*a*x2^2+b2*a*x1*x2)*<<0,0>>,-x2^2+(x1+1)*x2-x1],
[(-x2+x1)*<<1,1>>+(-b2*x2)*<<1,0>>+(b1*x1)*<<0,1>>,-x2+x1],
[((-x1+1)*x2+x1^2-x1)*<<2,0>>+
(((-a-b1+b2)*x1+c-b2-1)*x2+(a+b1)*x1^2+(-c+1)*x1)*<<1,0>>
+(-b1*x1*x2+b1*x1)*<<0,1>>
+(-b1*a*x1*x2+b1*a*x1^2)*<<0,0>>,(-x1+1)*x2+x1^2-x1]]
```

R のイデアル $I = \{L_1, L_2\}$ のグレブナ基底 G を計算します. I は `[[L1,1],[L2,1]]` のように表すこともできます. 計算結果は

$$G = \left\{ \begin{array}{l} \dfrac{t_1 S_2^2 + (b_2 x_1 - b_2) x_2 S_1 + t_2 S_2 + (-b_2 a x_2^2 + b_2 a x_1 x_2)}{-x_2^2 + (x_1 + 1) x_2 - x_1}, \\ \dfrac{(-x_2 + x_1) S_1 S_2 + (-b_2 x_2) S_1 + b_1 x_1 S_2}{-x_2 + x_1}, \\ \dfrac{t_3 S_1^2 + t_4 S_1 + (-b_1 x_1 x_2 + b_1 x_1) S_2 + (-b_1 a x_1 x_2 + b_1 a x_1^2)}{(-x_1 + 1) x_2 + x_1^2 - x_1} \end{array} \right\}$$

を意味します. ここで,

$$\begin{aligned} t_1 &= -x_2^2 + (x_1 + 1) x_2 - x_1 \\ t_2 &= (-a - b_2) x_2^2 + ((a - b_1 + b_2) x_1 + c - 1) x_2 + (-c + 1 + b_1) x_1 \\ t_3 &= (-x_1 + 1) x_2 + x_1^2 - x_1 \\ t_4 &= ((-a - b_1 + b_2) x_1 + c - b_2 - 1) x_2 + (a + b_1) x_1^2 + (-c + 1) x_1 \end{aligned}$$

つまり, 計算結果は R の元のリストです. 標準単項式は

```
[9] yang.stdmon(G);
[(1)*<<1,0>>,(1)*<<0,1>>,(1)*<<0,0>>]
```

で求まります. よって I のランクは 3 です.

グレブナ基底が求まったので, R の元 $t = (x_2 - x_1)\theta_1^2$ の正規形を求めましょう. これには yang.nf を用います.

```
[10] T=(x2-x1)*S1*S1;
(x2-x1)*<<2,0>>
[11] yang.nf(T,G);
[(((-a-b1+b2)*x1+c-b2-1)*x2+(a+b1)*x1^2+(-c+1)*x1)*<<1,0>>
+(-b1*x1*x2+b1*x1)*<<0,1>>+(-b1*a*x1*x2+b1*a*x1^2)*<<0,0>>,x1-1]
```

つまり計算結果は $\bmod I$ で

$$t \equiv \frac{((-a-b_1+b_2)x_1+c-b_2-1)x_2+(a+b_1)x_1^2+(-c+1)x_1}{x_1-1}\theta_1 + \frac{-b_1x_1x_2+b_1x_1}{x_1-1}\theta_2 + \frac{-b_1ax_1x_2+b_1ax_1^2}{x_1-1}$$

次に F_1 の Pfaff 形式を計算しましょう.

```
[10] S0 = yang.constant(1);
(1)*<<0,0>>
[11] Base=[S0,S1,S2];
[(1)*<<0,0>>,(1)*<<1,0>>,(1)*<<0,1>>]
[12] Pf=yang.pfaffian(Base,G);
[ [ 0 (1)/(x1) 0 ]
[ (-b1*a)/(x1-1)
(((-a-b1+b2)*x1+c-b2-1)*x2+(a+b1)*x1^2+(-c+1)*x1)/((x1^2-x1)*x2-x1^3+x1^2)
(-b1*x2+b1)/((x1-1)*x2-x1^2+x1) ]
[ 0 (-b2*x2)/(x1*x2-x1^2) (b1)/(x2-x1) ]
[ 0 0 (1)/(x2) ]
[ 0 (-b2)/(x2-x1) (b1*x1)/(x2^2-x1*x2) ]
[ (-b2*a)/(x2-1) (b2*x1-b2)/(x2^2+(-x1-1)*x2+x1)
((-a-b2)*x2^2+((a-b1+b2)*x1+c-1)*x2+(-c+b1+1)*x1)/(x2^3+(-x1-1)*x2^2+x1*x2) ] ]
[13] length(Pf);
2
[14] P1 = Pf[0];
[ 0 (1)/(x1) 0 ]
[ (-b1*a)/(x1-1)
(((-a-b1+b2)*x1+c-b2-1)*x2+(a+b1)*x1^2+(-c+1)*x1)/((x1^2-x1)*x2-x1^3+x1^2)
```

```
(-b1*x2+b1)/((x1-1)*x2-x1^2+x1) ]
[ 0 (-b2*x2)/(x1*x2-x1^2) (b1)/(x2-x1) ]
[15] P2 = Pf[1];
[ 0 0 (1)/(x2) ]
[ 0 (-b2)/(x2-x1) (b1*x1)/(x2^2-x1*x2) ]
[ (-b2*a)/(x2-1) (b2*x1-b2)/(x2^2+(-x1-1)*x2+x1)
((-a-b2)*x2^2+((a-b1+b2)*x1+c-1)*x2+(-c+b1+1)*x1)/(x2^3+(-x1-1)*x2^2+x1*x2) ]
```

計算を説明します. ランクが 3 であることに注意します. 基底を

$$F = \begin{pmatrix} f \\ S_1 f \\ S_2 f \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

ととり, $S_i f_j$ の正規形を計算することで Pfaff 形式を求めています. `Pf` は 3×3-行列のリストで長さは 2 です. 結果は,

$$\frac{\partial}{\partial x_1}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = P_1 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}, \qquad \frac{\partial}{\partial x_2}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = P_2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

ここで,

$$P_1 = \begin{pmatrix} 0 & \frac{1}{x_1} & 0 \\ \frac{-b_1 a}{x_1-1} & \frac{((-a-b_1+b_2)x_1+c-b_2-1)x_2+(a+b_1)x_1^2+(-c+1)x_1}{(x_1^2-x_1)x_2-x_1^3+x_1^2} & \frac{-b_1 x_2+b_1}{(x_1-1)x_2-x_1^2+x_1} \\ 0 & \frac{-b_2 x_2}{x_1 x_2-x_1^2} & \frac{b_1}{x_2-x_1} \end{pmatrix},$$

$$P_2 = \begin{pmatrix} 0 & 0 & \frac{1}{x_2} \\ 0 & \frac{-b_2}{x_2-x_1} & \frac{b_1 x_1}{x_2^2-x_1 x_2} \\ \frac{-b_2 a}{x_2-1} & \frac{b_2 x_1-b_2}{x_2^2+(-x_1-1)x_2+x_1} & \frac{(-a-b_2)x_2^2+((a-b_1+b_2)x_1+c-1)x_2+(-c+b_1+1)x_1}{x_2^3+(-x_1-1)x_2^2+x_1 x_2} \end{pmatrix}$$

3 $\mathcal{A}$-超幾何微分差分系を計算してみる

$\mathcal{A}$-超幾何微分差分系については専用の関数が用意されています. まず行列 A と パラメータベクトル β が与えられたとき, オイラー微分演算子の形で方程式系を求める必要があります.

```
[1] load("yang.rr");
[2] A=[[1,1,1],[0,1,2]];
[3] B=[s1,s2];
[4] GKZ=[A,B];
[[[1,1,1],[0,1,2]],[s1,s2]]
[5] yang.define_gkz_ring(GKZ);
[[x1,x2,x3,s1,s2],[0,0,0,1,1],[0,0,0,-1,-1]]
```

```
[6] E = yang.gkz(GKZ);
[[(1)*<<1,0,0,0,0>>+(1)*<<0,1,0,0,0>>+(1)*<<0,0,1,0,0>>+(-s1)*<<0,0,0,0,0>>,
(1)*<<0,1,0,0,0>>+(2)*<<0,0,1,0,0>>+(-s2)*<<0,0,0,0,0>>,
(1)*<<1,0,0,0,0>>+(-x1)*<<0,0,0,1,0>>,(-x2)*<<0,0,0,1,1>>+(1)*<<0,1,0,0,0>>,
(-x3)*<<0,0,0,1,2>>+(1)*<<0,0,1,0,0>>],[x1,x2,x3]]
```

yang.define_gkz_ring で微分差分環を定義する. 演算子の分散多項式表示のうち, 最初の 3 つは x_i に関するオイラー微分演算子, あとのふたつは s_i に関するシフト演算子である.

yang.gkz は $\mathcal{A}$-超幾何微分差分系を出力する. E は $\mathcal{A}$-超幾何微分差分系である.

トーリックイデアルを計算するには次のようにする.

```
[7] IA=yang.gkz_toric(GKZ);
[(x1*x3)*<<0,2,0,0,0>>+(-x2^2)*<<1,0,1,0,0>>+(-x1*x3)*<<0,1,0,0,0>>]
```

IA はトーリックイデアルの生成元をオイラー微分演算子の形で書いたものである. 通常の偏微分演算子による表示を得るには,

```
[8] yang.compute_toric_kernel(GKZ);
[[-_x2*_x0+_x1^2],[_x0,_x1,_x2,_t1,_t2]]
```

あるいは yang.gkz_toric_partial も使える.

4 API リファレンス

yang.define_ring(Ring)

環を定義し、yang の内部データ構造を初期化する。以前の定義はスタックに積まれる。環定義における変数の並び順によって、変数順序が定まるので注意すること。

Ring の 定義

```
Ring := '[' ( Vars | RingDef )  ']'
Vars := Variable [ , Variable ]*
RingDef := RingEl [ , RingEl ]*
RingEl := Keyword , '[' ( Vars | Pairs ) ']'
Keyword := "euler" | "differential" | "difference"
Pairs := Pair [ , Pair ]*
Pair := '[' Variable , ( Number | Variable ) ']'
```

yang.pop_ring()

以前の環定義を取り出す。現在の環定義は破棄される。

yang.operator(Variable)

Variable に対応する演算子を取り出す。演算子は分散表現単項式で与えられる。

yang.constant(Number)

Number の環における表現を取り出す。この関数は yang.pfaffian で与える基底を生成するのに有用である。

yang.multi(DPolynomial, DPolynomial)

演算子同士の積を計算する。

yang.nf(RDPolynomial, Ideal) (別名: yang.reduction)

RDPolynomial を Ideal で簡約する。Ideal が グレブナ基底になっている場合には、正規形になる。

```
RDPolynomial := DPolynomial | '[' DPolynomial , Polynomial ']'
```

yang.buchberger(Ideal)

```
Ideal := '[' RDPolynomial [ , RDPolynomial ]* ']'
```

イデアル Ideal のグレブナ基底を計算する。変数順序は環定義で定まり、項順序は分散表現多項式の表現に依存している。したがって、入力 Ideal をつくるまえに項順序を定めておく必要がある。

yang.stdmon(Ideal)

グレブナ基底 Ideal の標準単項式を計算する。

yang.pfaffian(Base, Ideal)

グレブナ基底 Ideal の生成するイデアルの定める微分方程式系に対応する、Pfaff 方程式系を求める。Pfaff 方程式系の解の基底には Base を用いる。(この関数は要改良である!!)

```
Base := '[' DMonomial [ , DMonomial ]* ']'
```