

1 変数代数的局所コホモロジー類用パッケージ **taji_alc**

1 変数代数的局所コホモロジー類用パッケージ taji_alc
1.0 版
2007 年 11 月

庄司卓夢，田島慎一

1 1 変数代数的局所コホモロジー類

1.1 1 変数代数的局所コホモロジー類用のパッケージ `taji_alc` について

この説明書では 1 変数代数的局所コホモロジー類用のパッケージ `taji_alc` について説明する. 数学的解説や背景については, 解説記事 “1 変数代数的局所コホモロジー類用に対する Risa/Asir 用パッケージ `taji_alc`” (Risa/Asir Journal (2007)) およびその参考文献を参照.

1.2 1 変数代数的局所コホモロジー類用の関数

本セクションの関数を呼び出すには,

```
import("taji_alc.rr")$
```

を実行してプログラムをロードする.

1.2.1 `taji_alc.cpfid`

`taji_alc.cpfid(num,den)`

:: 有理関数 num/den の部分分数分解を求める.

return *switch* が 0 か 1 ならば, [[[分子],[分母の因子, 重複度]],...],... なるリスト.

switch が 10 か 11 ならば, [[分子],[分母の因子, 重複度]],... なるリスト.

num (有理関数の分子の) 多項式

den (有理関数の分母の) 多項式

または (有理関数の分母を $\mathbb{Q}$ 上で既約分解した) [[因子, 重複度],...] なるリスト

switch オプション指定

case 0 : complete な部分分数分解を返す. (分子は有理数係数多項式)

case 1 : complete な部分分数分解を返す. (分子は整数係数化リスト)

case 10 : 分母を冪展開しない部分分数分解を返す. (分子は有理数係数多項式)

case 11 : 分母を冪展開しない部分分数分解を返す. (分子は整数係数化リスト)

default : case 0

- `taji_alc.cpfid()` は, proper な有理関数を対象とする. 入力値が proper でない場合でも正常に動作するが, 多項式として出てくる部分は表示しない.
- 部分分数分解は, 冪展開をする complete なタイプと, 冪展開をしないタイプの 2 つのタイプがある. `taji_alc.cpfid()` で採用しているアルゴリズムでは, 前者が先に求まる. 後者は, 前者のデータをホーナー法で足し上げて求める.
- *den* は, リストでの入力望ましい. (多項式で入力すると, 簡約化の処理が生じるため重くなる.) ただしその場合には, 既約チェック, 有理式の約分, 整数係数化は行わないので注意する. 入力値はユーザ側が責任をもつ.

```
[235] taji_alc.cpfid(x^3-x-1,x^4+2*x^3+2*x^2+2*x+1);
[[[1/2*x-1,[x^2+1,1]]],[[-1/2,[x+1,2]],[1/2,[x+1,1]]]]
[236] taji_alc.cpfid(x^3-x-1,x^4+2*x^3+2*x^2+2*x+1|switch=1);
[[[x-2,2],[x^2+1,1]]],[[-1,2],[x+1,2]],[[1,2],[x+1,1]]]]
[237] taji_alc.cpfid(x^3-x-1,x^4+2*x^3+2*x^2+2*x+1|switch=10);
[[1/2*x-1,[x^2+1,1]],[1/2*x,[x+1,2]]]
```

```
[238] taji_alc.cpfid(x^3-x-1,x^4+2*x^3+2*x^2+2*x+1|switch=11);  
[[[x-2,2],[x^2+1,1]],[[x,2],[x+1,2]]]
```

参照

ChangeLog

1.2.2 taji_alc.snoether

`taji_alc.snoether(num,den)`
 :: 有理関数 num/den が定める代数的局所コホモロジー類のネーター作用素を求める.

`return` [[因子, ネーター作用素],...] なるリスト.
 ネーター作用素は, 係数を高階の部分から降順に並べたリスト

`num` (有理関数の分子の) 多項式

`den` (有理関数の分母の) 多項式
 または (有理関数の分母を Q 上で既約分解した) [[因子, 重複度],...] なるリスト.

`switch` オプション指定

case 0 : ネーター作用素を [有理数係数多項式,...] なるリストで返す.

case 1 : ネーター作用素を [整数係数化リスト,...] なるリストで返す.

case 10 : ネーター作用素を [[整数係数多項式,...], 整数] なるリストで返す.

case 20 : ネーター作用素を [[整数係数化リスト,...], 整数] なるリストで返す.

default : case 0

- `taji_alc.snoether()` は, den を Q 上で既約分解し, 各因子に対応するネーター作用素を返す.
- den は, リストでの入力が見やすい. (多項式で入力すると, 簡約化の処理が生じるため重くなる.)
 ただしその場合には, 既約チェック, 有理式の約分, 整数係数化は行わないので注意する. 入力値はユーザ側が責任をもつ.
- 戻り値の型は `switch` で選択できる.

case 10 は, ネーター作用素の各係数全体を通分し, その分母部分と階乗の積をリストで分けた表現である. わかりやすいが, 通分値と係数部分とで約分できる部分がある (特に高階の部分に多い) ので, 冗長性をもっている.

case 20 は, 階乗の部分で全体をくくり (リストで分け), ネーター作用素の各係数を個別に通分しリスト化する. 階乗の部分と係数部分とで約分できる部分がある (特に低階の部分に多い) ので, 冗長と言えなくもない (case 10 よりはまし) が, 数学的な構造が綺麗に見える表現である.

```
[296] taji_alc.snoether(1, [[x^3-x-1,3]]);
[[x^3-x-1, [9/529*x^2-27/1058*x+11/1058, -81/529*x^2-9/529*x+135/529, -4905/12167*x^2+4563/12167*x+3270/12167]]]
```

```
[299] taji_alc.snoether(1, [[x^3-x-1,3]]|switch=1);
[[x^3-x-1, [[18*x^2-27*x+11,1058], [-81*x^2-9*x+135,529], [-4905*x^2+4563*x+3270,12167]]]]]
```

```
[297] taji_alc.snoether(1, [[x^3-x-1,3]]|switch=10);
[[x^3-x-1, [[414*x^2-621*x+253, -3726*x^2-414*x+6210, -9810*x^2+9126*x+6540], 24334]]]
```

```
[298] taji_alc.snoether(1, [[x^3-x-1,3]]|switch=20);
[[x^3-x-1, [[18*x^2-27*x+11,529], [-162*x^2-18*x+270,529], [-9810*x^2+9126*x+6540,12167]],2]]]
```

```
[241] taji_alc.snoether(x^3+1,x^18-2*x^14+x^10-x^8+2*x^4-1|switch=10);
[[x^4+x^3+x^2+x+1, [[-2*x^2-x-2],50]], [x^4-x^3+x^2-x+1, [[-2*x^3+4*x^2-x-2],50]], [x^2+1, [[-x+1,8*x+5],32]], [x+1, [[-6,-39],320]], [x-1, [[2,-24,67],320]]]
```

参照

ChangeLog

1.2.3 taji_alc.laurent_expansion

`taji_alc.laurent_expansion(num, den)`
 :: 有理関数 num/den の極におけるローラン展開の主要部の係数を求める.

return [[因子, ローラン展開の係数],...] なるリスト.
 ローラン展開の係数は, 高位の係数から順に並べたリスト.

num (有理関数の分子の) 多項式

den (有理関数の分母の) 多項式
 または (有理関数の分母を $\mathbb{Q}$ 上で既約分解した) [[因子, 重複度],...] なるリスト

switch オプション指定

case 0 : ローラン展開の係数を [有理数係数多項式,...] なるリストで返す.
 case 1 : ローラン展開の係数を [整数係数化リスト,...] なるリストで返す.
 case 10 : ローラン展開の係数を [[整数係数多項式,...], 整数] なるリストで返す.
 case 20 : ローラン展開の係数を [[整数係数化リスト,...], 整数] なるリストで返す.
 default : case 0

- `taji_alc.laurent_expansion()` は, `taji_alc.snoother()` を使って, ローラン展開の係数を求める.
- `taji_alc.laurent_expansion()` では, $\mathbb{C}$ 上の 1 点に注目するのではなく, $\mathbb{Q}$ 上での既約因子自体に注目してローラン展開の係数を求める. 戻り値の係数リストの各成分は, その因子の全ての零点が共通に満たすローラン展開の係数多項式である. 従って, 1 点ごとのローラン展開の係数をさらに求めたい場合には, 求めたローラン展開の係数多項式に因子の零点 (即ち特異点) の値を代入する必要がある.

```
[354] taji_alc.laurent_expansion(x, (x-1)^3);
[[x-1, [1, 1, 0]]]
[356] taji_alc.laurent_expansion(x^5+x^4+x^3+x^2+x+1, (x^4+1)^3);
[[x^4+1, [1/64*x^2+1/64*x, 1/16*x^3+1/16*x^2-3/128*x-5/128, -5/128*x^3-1/8*x^2-3/16*x]]]
```

参照 Section 1.2.2 [taji_alc.snoother], page 3

ChangeLog

1.2.4 taji_alc.residue

`taji_alc.residue(num, den)`
 :: 有理関数 num/den の極における留数を求める.

`return` [[因子, 留数],...] なるリスト

`num` (有理関数の分子の) 多項式

`den` (有理関数の分母の) 多項式
 または (有理関数の分母を $\mathbb{Q}$ 上で既約分解した) [[因子, 重複度],...] なるリスト

`switch` オプション指定
 case 0 : 留数を有理数係数多項式で返す.
 case 1 : 留数を整数係数化リストで返す.
 default : case 0

`pole` オプション指定
 [因子,...] なるオプションリスト

- `taji_alc.residue()` は, `den` を $\mathbb{Q}$ 上で既約分解し, 各因子の零点 (即ち有理関数の極) における留数を返す.
- オプションで `pole` を指定すればその因子のみの留数を返す. 指定が不適当だと 0 を返す.
- `taji_alc.residue()` で採用しているアルゴリズムでは, $\mathbb{C}$ 上の 1 点に注目するのではなく, $\mathbb{Q}$ 上での既約因子自体に注目して留数を求める. 戻り値の留数は, その因子の全ての零点が共通に満たす留数多項式である. 従って, 1 点ごとの留数値をさらに求めたい場合には, 求めた留数多項式に因子の零点 (即ち特異点) の値を代入する必要がある.

```
[219] taji_alc.residue(1,x^4+1);
      [[x^4+1,-1/4*x]]
```

この例で言うと, 求めた留数多項式 $-1/4 \cdot x$ に, x^4+1 の (4 つある) 零点をそれぞれ代入したものが個別の留数値である.

- `den` は, リストでの入力が見やすい. (多項式で入力すると, 簡約化の処理が生じるため重くなる.) ただしその場合には, 既約チェック, 有理式の約分, 整数係数化は行わないので注意する. 入力値はユーザ側が責任をもつ.

```
[221] taji_alc.residue(x^8, [[x^3-x-1,3]]);
      [[x^3-x-1,-2243/12167*x^2+2801/12167*x+5551/12167]]
[222] taji_alc.residue(x^2+x, [[x+1,3],[x-1,3],[x^2+3*x-1,2]]);
      [[x^2+3*x-1,-284/4563*x-311/1521],[x-1,89/432],[x+1,7/432]]
[223] taji_alc.residue(x^2+x, [[x+1,3],[x-1,3],[x^2+3*x-1,2]]|switch=1)
      ;
      [[x^2+3*x-1,-284*x-933,4563],[x-1,[89,432]],[x+1,[7,432]]]
[234] taji_alc.residue(x^2+x, [[x+1,3],[x-1,3],[x^2+3*x-1,2]]|switch=1,
      pole=[x+1]);
      [[x+1,[7,432]]]
[225] taji_alc.residue(x^3+1,x^18-2*x^14+x^10-x^8+2*x^4-1);
      [[x^4+x^3+x^2+x+1,-1/25*x^2-1/50*x-1/25],[x^4-x^3+x^2-x+1,-1/25*x^3+2/
      25*x^2-1/50*x-1/25],[x^2+1,1/4*x+5/32],[x+1,-39/320],[x-1,67/320]]
```

参照

ChangeLog

1.2.5 taji_alc.invpow

taji_alc.invpow(poly, f, m)

:: 剰余体 $\mathbb{Q}[x]/\langle f \rangle$ 上での $poly$ の逆元の m 乗を求める.

return 逆冪

poly 多項式

f $\mathbb{Q}$ 上で既約な多項式

m 自然数

switch オプション指定

case 0 : 逆冪を有理数係数多項式で返す.

case 1 : 逆冪を整数係数化リストで返す.

default : case 0

- $poly$ と f は互いに素でなければならない.
- アルゴリズムの骨格は繰り返し 2 乗法である. そこに最小多項式の理論を応用して高速化している.

```
[236] taji_alc.invpow(3*x^2-1,x^3-x-1,1);
```

```
-6/23*x^2+9/23*x+4/23
```

```
[237] taji_alc.invpow(3*x^2-1,x^3-x-1,1|switch=1);
```

```
[-6*x^2+9*x+4,23]
```

```
[238] taji_alc.invpow(3*x^2-1,x^3-x-1,30|switch=1);
```

```
[1857324483*x^2-2100154824*x-477264412,266635235464391245607]
```

参照

ChangeLog

1.2.6 taji_alc.rem_formula

taji_alc.rem_formula(polylist)

:: 多項式 $f(x)$ を与えたときの剰余公式を求める.

return switch および 説明文を参照

polylist $f(x)$ を Q 上で既約分解した $[[\text{因子}, \text{重複度}, \text{零点の記号}], \dots]$ なるリスト

switch オプション指定

case 0 : x の冪で整理し, リストで返す.

case 10 : $f(x)$ の冪で整理し, リストで返す. (一因子の場合のみ対応)

case 20 : x の冪で整理し, symbolic な表現で返す.

default : case 0

- アルゴリズムは, エルミートの補間剰余を用いている.
- 剰余公式の表現方法はいくつか考えられるため, switch で選択式とした.
- switch=0 の戻り値の見方を述べる. 例として, $f(x)=f_1(x)^{m_1}f_2(x)^{m_2}$ を考える. 入力は $[[f_1(x), m_1, z_1], [f_2(x), m_2, z_2]]$ となる. そのとき戻り値は, $[r_{f_1}(x, z_1), r_{f_2}(x, z_2)]$ なるリストで返される. これは, 剰余公式が
$$r(x) = r_{f_1}(x, z_1) + r_{f_2}(x, z_2)$$
 なる形で与えられることを意味している. 各成分の $r_{f_i}(x, z_i)$ は, $[p^{(m_i-1)}(z_i)]$ の係数となる x と z_i の多項式, $\dots, p^{(0)}(z_i)$ の係数となる x と z_i の多項式 なるリストである.
- switch=10 の戻り値の見方を述べる. 例として, $f(x)=f_1(x)^m$ を考える. 入力は $[[f_1(x), m, z]]$ となる. そのとき戻り値は, $[r_{(m-1)}(x, z), \dots, r_0(x, z)]$ なるリストで返される. 各成分は, 剰余公式を
$$r(x) = r_{m-1}(x, z)f_1(x)^{m-1} + \dots + r_0(x, z)$$
 のように $f_1(x)$ の冪で展開したときの各係数を意味している. 各成分の $r_i(x, z)$ は, $[p^{(m-1)}(z)]$ の係数となる x と z の多項式, $\dots, p^{(0)}(z)$ の係数となる x と z の多項式 なるリストである.
- switch=20 の戻り値の見方を述べる. symbolic な出力の $p^{(m)}(z)$ は, $p(x)$ の m 階の導関数に z を代入した値という意味である.
- 戻り値は, 与えた因子の全ての零点を代入したものの和として見る. これは因子が 2 次以上の多項式の場合に関係してくる. 例えば,

```
[228] taji_alc.rem_formula([[x^2+1, 1, z]]);
      [[-1/2*z*x+1/2]]
```

の正しい見方は, x^2+1 の零点を a_1, a_2 とおいたときに, z に a_1 と a_2 を代入した,

$r(x)=(-1/2*a_1*x+1/2)+(-1/2*a_2*x+1/2)$ である. しかし出力では, 零点の和の部分の便宜上省略して返す.

```
[583] taji_alc.rem_formula([[x-1, 1, z1], [x-2, 1, z2]]);
      [[-x+2], [x-1]]
```

```

[584] taji_alc.rem_formula([[x-1,1,z1],[x-2,1,z2]]|switch=20);
(-p^(0)(z1)+p^(0)(z2))*x+2*p^(0)(z1)-p^(0)(z2)

[587] taji_alc.rem_formula([[x-1,2,z1]]);
[[x-1,1]]
[588] taji_alc.rem_formula([[x-1,2,z1]]|switch=20);
p^(1)(z1)*x-p^(1)(z1)+p^(0)(z1)

[494] taji_alc.rem_formula([[x-1,3,z1]]|switch=20);
1/2*p^(2)(z1)*x^2+(-p^(2)(z1)+p^(1)(z1))*x+1/2*p^(2)(z1)-p^(1)(z1)+p^(0)(z1)

[229] taji_alc.rem_formula([[x+1,2,z1],[x^3-x-1,1,z2]]);
[[-x^4-x^3+x^2+2*x+1,-2*x^4-3*x^3+2*x^2+5*x+3], [(-1/23*z2^2-10/23*z2+16/23)*x^4+(-12/23*z2^2-5/23*z2+31/23)*x^3+(-5/23*z2^2+19/23*z2-12/23)*x^2+(22/23*z2^2+13/23*z2-53/23)*x+16/23*z2^2-1/23*z2-26/23]]
[230] taji_alc.rem_formula([[x+1,2,z1],[x^3-x-1,1,z2]]|switch=20);
(-1/23*p^(0)(z2)*z2^2-10/23*p^(0)(z2)*z2-2*p^(0)(z1)+16/23*p^(0)(z2)-p^(1)(z1))*x^4+(-12/23*p^(0)(z2)*z2^2-5/23*p^(0)(z2)*z2-3*p^(0)(z1)+31/23*p^(0)(z2)-p^(1)(z1))*x^3+(-5/23*p^(0)(z2)*z2^2+19/23*p^(0)(z2)*z2+2*p^(0)(z1)-12/23*p^(0)(z2)+p^(1)(z1))*x^2+(22/23*p^(0)(z2)*z2^2+13/23*p^(0)(z2)*z2+5*p^(0)(z1)-53/23*p^(0)(z2)+2*p^(1)(z1))*x+16/23*p^(0)(z2)*z2^2-1/23*p^(0)(z2)*z2+3*p^(0)(z1)-26/23*p^(0)(z2)+p^(1)(z1)

[231] taji_alc.rem_formula([[x^3-x-1,2,z]]|switch=10);
[[[(3/23*z^2-4/23)*x^2+(-1/23*z+3/23)*x-4/23*z^2+3/23*z+4/23,(162/529*z^2-174/529*z-108/529)*x^2+(-105/529*z^2+54/529*z+70/529)*x-108/529*z^2+116/529*z+72/529], [(-6/23*z^2+9/23*z+4/23)*x^2+(9/23*z^2-2/23*z-6/23)*x+4/23*z^2-6/23*z+5/23]]]

```

参照

ChangeLog


```
184*z+180,529]]],[[x^3-x-1,[-105*x^2+(-23*z+54)*x+69*z+70,529]]],[[x^3
-x-1,[(69*z+162)*x^2-174*x-92*z-108,529]]]]
```

```
[277] taji_alc.solve_ode_cp(x^2-4,z,0);
[[[x+2,1/2],[x-2,1/2]],[[x+2,-1/4],[x-2,1/4]]]
[278] taji_alc.solve_ode_cp(x^2-4,z,0|data=[1,-1]);
[[x+2,3/4],[x-2,1/4]]
[279] taji_alc.solve_ode_cp(x^2-4,z,0|data=[c0,c1]);
[[x+2,1/2*c0-1/4*c1],[x-2,1/2*c0+1/4*c1]]
```

参照

ChangeLog

1.2.8 taji_alc.solve_ode_cp_ps

`taji_alc.solve_ode_cp_ps(poly, var, exppoly)`
 :: 有理数係数の線形常微分方程式のコーシー問題
 $Pu(z) = f(z), u^{(0)}(0) = c_0, \dots, u^{(n-1)}(0) = c_{n-1}$
 の特殊解を求める.
 ただし, 非斉次形のみを対象としているので, $f(z) \neq 0$ とする.

return 指数多項式リスト

poly 多項式 (P の特性多項式)
 または (P の特性多項式を Q 上で既約分解した) [[因子, 重複度],...] なるリスト

var 不定元 (関数の独立変数)

exppoly $f(z)$ の指数多項式リスト

switch オプション指定
 case 0 : 指数多項式リストの 2 番目の成分を有理数係数多項式で返す.
 case 1 : 指数多項式リストの 2 番目の成分を整数係数化リストで返す.
 default : case 0

switch2 オプション指定
 case 0 : コーシー問題の特殊解を返す.
 case 1 : 簡単な形の特殊解を返す.
 default : case 0

- 変数は 2 種類必要 (特性多項式の変数と関数の独立変数). `poly` の不定元と `var` の不定元が衝突しないよう注意.

```
[345] taji_alc.solve_ode_cp_ps((x-2)*(x+3), z, [[x-1, 1]]);
[[x+3, 1/20], [x-1, -1/4], [x-2, 1/5]]
[346] taji_alc.solve_ode_cp_ps((x-2)*(x+3), z, [[x-1, 1]] | switch2=1);
[[x-1, -1/4]]
[347] taji_alc.solve_ode_cp_ps((x-2)*(x+3), z, [[x-2, 1]]);
[[x+3, 1/25], [x-2, 1/5*z-1/25]]
[348] taji_alc.solve_ode_cp_ps((x-2)*(x+3), z, [[x-2, 1]] | switch2=1);
[[x-2, 1/5*z-1/25]]
[349] taji_alc.solve_ode_cp_ps((x-2)*(x+3), z, [[x+1, 1], [x-2, 1]] | switch2=1);
[[x+1, -1/6], [x-2, 1/5*z+2/75]]

[350] taji_alc.solve_ode_cp_ps((x^3-x-1)*(x-3)^2, z, [[x-3, 2], [x-1, 3*z^2+1]]);
[[x-1, [-6*z^2-36*z-119, 8]], [x^3-x-1, [42291*x^2+55504*x+32313, 12167]], [x-3, [4232*z^2-4278*z-4295, 97336]]]
```

参照

ChangeLog

1.2.9 taji_alc.fbt

`taji_alc.fbt(num, den, var)`
 :: 有理関数 num/den が定める代数的局所コホモロジー類のフーリエ・ボレル変換を行う.

`return` [指数多項式リスト,...] なるリスト

`num` (有理関数の分子の) 多項式

`den` (有理関数の分母の) 多項式
 または (有理関数の分母を $\mathbb{Q}$ 上で既約分解した) [[因子, 重複度],...] なるリスト

`var` 不定元 (像の独立変数)

`switch` オプション指定
 case 0 : 指数多項式リストの 2 番目の成分を有理数係数多項式で返す.
 case 1 : 指数多項式リストの 2 番目の成分を整数係数化リストで返す.
 default : case 0

- 変数は 2 種類必要 (代数的局所コホモロジー類の変数と像の独立変数). num/den の不定元と `var` の不定元が衝突しないよう注意.
- `taji_alc.fbt()` は, $\text{Res}(\text{Rat} * \exp(z * x))$ なる形の有理形関数の留数を求める. この有理形関数の留数は指数多項式となるため, 指数多項式リストで返す.
- 内部のアルゴリズムは `taji_alc.residue()` とほぼ同じであり, 実際に `taji_alc.residue()` を呼び出して計算を行っている.

```
[235] taji_alc.fbt(1, (x^3-x-1)^3, z);
[[x^3-x-1, (9/529*z^2-81/529*z-4905/12167)*x^2+(-27/1058*z^2-9/529*z+45
63/12167)*x+11/1058*z^2+135/529*z+3270/12167]]
```

参照 Section 1.2.4 [`taji_alc.residue`], page 6

ChangeLog

1.2.10 taji_alc.invfbt

taji_alc.invfbt(exppoly, var)

:: 指数多項式の逆フーリエ・ボレル変換を行う.

return 有理関数

exppoly 指数多項式リスト

var 不定元 (指数多項式の独立変数)

switch オプション指定

case 0 : 有理関数で返す.

case 1 : 有理関数を [分子, 分母を Q 上で既約分解したリスト] なるリストで返す.

default : case 0

- 変数は 2 種類必要 (代数的数の最小多項式の変数と指数多項式の独立変数). 衝突しないよう注意.
- taji_alc.invfbt() は, exppoly を, $\text{Res}(\text{Rat} \cdot \exp(z \cdot x))$ なる形の留数表示に変換し, Rat 部分を返す.
- taji_alc.fbt() の逆演算である.

```
[8] taji_alc.invfbt([[x^3-x-1, 2*x^2*z^2+x*z+1], [x^2+1, z*x+z^2]], z|switch=1);
[3*x^14+14*x^12+39*x^11+33*x^10+179*x^9+206*x^8+350*x^7+223*x^6+126*x^5+176*x^4+107*x^3+101*x^2+15*x-4, [[x^2+1, 3], [x^3-x-1, 3]]]
```

```
[9] taji_alc.fbt(3*x^14+14*x^12+39*x^11+33*x^10+179*x^9+206*x^8+350*x^7+223*x^6+126*x^5+176*x^4+107*x^3+101*x^2+15*x-4, [[x^2+1, 3], [x^3-x-1, 3]], z);
[[x^3-x-1, 2*z^2*x^2+z*x+1], [x^2+1, z*x+z^2]]
```

参照 Section 1.2.9 [taji_alc.fbt], page 13

ChangeLog

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

1	1 変数代数的局所コホモロジー類	1
	Index	15

Table of Contents

1	1 変数代数的局所コホモロジー類	1
1.1	1 変数代数的局所コホモロジー類用のパッケージ <code>taji_alc</code> について	1
1.2	1 変数代数的局所コホモロジー類用の関数	1
1.2.1	<code>taji_alc.cpfid</code>	1
1.2.2	<code>taji_alc.snoether</code>	3
1.2.3	<code>taji_alc.laurent_expansion</code>	5
1.2.4	<code>taji_alc.residue</code>	6
1.2.5	<code>taji_alc.invpow</code>	7
1.2.6	<code>taji_alc.rem_formula</code>	8
1.2.7	<code>taji_alc.solve_ode_cp</code>	10
1.2.8	<code>taji_alc.solve_ode_cp_ps</code>	12
1.2.9	<code>taji_alc.fbt</code>	13
1.2.10	<code>taji_alc.invfbt</code>	14
	Index	15