Risa/Asir 終結式計算パッケージ f_res 説明書

利用説明書
1.0 版
2005 年 6 月

by Kenji Fujiwara and Masayuki Noro

1 関数マニュアル

1.1 概要

`f_res` パッケージは, 多変数多項式集合に対し, dense な係数をもつとして multipolynomial resultant を計算する `f_res.mres`, sparse な係数を持つ場合に sparse resultant を計算する `f_res.sres`, Dixon の方法により resultant を計算する `f_res.dres` および, 付随する関数を実装している. 実際には, これらは真の resultant の多項式倍を返す場合があるが, 消去イデアルに属する多項式を一つ求めたい場合には, グレブナー基底による消去に比較して効率がよい場合がある.

これらの方法においては, 線形計画法, 凸包, mixed volume の計算などが必要となるが, これらについてはフリーソフトである `cddlib` および `MixedVol` を利用した. これらは OpenXM サーバ `ox_sres` としてまとめられている. これは, ソースディストリビューションでは, 自動的には make されないが, '`OpenXM/src/ox_cdd`' において make, make install することにより, asir のライブラリディレクトリにインストールされる. これを利用して上で述べた resultant を計算する asir 関数が, '`OpenXM/src/asir-contrib/packages/f_res/f_res.rr`' にある. これを load することで, 次節以降で述べる機能が使えるようになる. なお, 線形計画法および凸包計算は, `gmp` による厳密計算を行うものと, 浮動小数による近似計算で行うものの 2 通りが用意されている. 後者の方が高速だが, 誤差が生ずる場合がある. この選択は, `f_res.gmp()`, `f_res.float()` を呼び出すことで行う.

1.2 Notation

このマニュアルでは点をリストで, support や polytope をリストのリストで表す. つまり, 点 (1,1) はリスト `[1,1]` で表し, 点 {(0,0), (1,0), (0,1) } からなる polytope をリストのリスト `[[0,0],[1,0],[0,1] ]` で表す.

1.3 主な関数

1.3.1 f_res.mres, f_res.mresM

`f_res.mres(`*Equations*`, `*Vars*` )`
:: Multipolynomial resultant の多項式倍を返す

`f_res.mresM(`*Equations*`, `*Vars*` )`
:: 行列式が `f_res.mres` が返す値になるような行列を返す

return

`f_res.mres`
多項式もしくは 0

`f_res.mresM`
行列

Equaitons 多項式のリスト

Vars 変数のリスト.

オプション

`rsc` 任意

`rowidx` 配列

`colidx` 配列

`p` 素数

`sub` リスト

- *Equations* の成分の多項式による不定元を *Vars* としたとき斉次多項式の場合の方法で `f_res.mres` は resultant の多項式倍を, `f_res.mresM` は resultant の多項式倍を行列式にもつ行列を返す.
- *Equations* の成分の多項式は内部で自動的に斉次化されているから, 斉次多項式である必要はない.
- Rank Submatrix Construction を行ないたいときはオプション `rsc` を 1 に設定する. その場合, この関数は内部で関数 `f_res.submatrix` を呼び出しているので, そのためのオプションはすべて受け付ける.

```
[0] F0 = a1*x + a2*y + a3$
[1] F1 = b1*x + b2*y + b3$
[2] F2 = c1*x^2 + c2*y^2 + c3 + c4*x*y + c5*x + c6*y$
[3] f_res.mresM( [F0,F1,F2], [x,y] );
[ 0 0 0 a2 a3 a1 ]
[ 0 a2 a3 0 a1 0 ]
[ a2 a3 0 a1 0 0 ]
[ 0 b2 b3 0 b1 0 ]
[ b2 b3 0 b1 0 0 ]
[ c2 c6 c3 c4 c5 c1 ]
[4] R = f_res.mres( [F0,F1,F2], [x,y] );
(-c3*b2^2+c6*b3*b2-c2*b3^2)*a1^3+(((2*c3*b2-c6*b3)*b1-c5*b3*b2+c4*b3^2)*a2+((-c
6*b2+2*c2*b3)*b1+c5*b2^2-c4*b3*b2)*a3)*a1^2+((-c3*b1^2+c5*b3*b1-c1*b3^2)*a2^2+(
c6*b1^2+(-c5*b2-c4*b3)*b1+2*c1*b3*b2)*a3*a2+(-c2*b1^2+c4*b2*b1-c1*b2^2)*a3^2)*a
1
[5] fctr( R );
[[-1,1],[a1,1],[(c3*b2^2-c6*b3*b2+c2*b3^2)*a1^2+(((-2*c3*b2+c6*b3)*b1+c5*b3*b2-
c4*b3^2)*a2+((c6*b2-2*c2*b3)*b1-c5*b2^2+c4*b3*b2)*a3)*a1+(c3*b1^2-c5*b3*b1+c1*b
3^2)*a2^2+(-c6*b1^2+(c5*b2+c4*b3)*b1-2*c1*b3*b2)*a3*a2+(c2*b1^2-c4*b2*b1+c1*b2^
2)*a3^2,1]]
```

1.3.2 `f_res.indexof`

`f_res.indexof(`*Element*`,` *List* `)`

:: リスト中に要素が最初に現れる位置を返す

Element 検索したい要素

List 検索対象のリスト

return *List* で最初に現れる *Element* のインデックス番号. *List* に *Element* が現れない場合は整数 -1.

- *List* で最初に現れる *Element* のインデックス番号を返す. *List* に *Element* が現れない場合は -1 を返す.

- *Element* の型は何であっても構わない.
- 関数 `flist` と組み合わせると, ある関数が Asir に入っているかが分かる.

```
[0] f_res.indexof( 2, [1,2,3] );
1
[1] f_res.indexof( 4, [1,2,3] );
-1
[2] f_res.indexof( "nd_det", flist() );
31
[3] f_res.indexof( "nd_Det", flist() );
-1
```

1.3.3 f_res.listadd

`f_res.listadd(`*A*`, `*B*` )`
:: リストをベクトルと見て和を求める

A
B リスト

return リスト

- ベクトルの和のようにリスト A とリスト B の和を求める.
- リスト A とリスト B の長さは等しくなくてはいけない.

```
[0] f_res.listadd( [1,2,3], [4,5,6] );
[5,7,9]
[1] f_res.listadd( [a,b,c], [d,e,f] );
[a+d,b+e,c+f]
```

1.3.4 f_res.start

`f_res.start(`*N*`)`
:: `ox_sres` を起動する

N 任意

return 整数

- パラメータ N が 1 のときは GMP 版, それ以外のときは浮動小数版の新しい OpenXM サーバ `ox_sres` を起動し, 他の関数で使われるサーバに設定する.
- 実行ファイルが見つからないときはデバッグモードに入る.
- 返される整数は通信のための識別子 .

1.3.5 f_res.float

`f_res.float()`
:: `ox_sres` を起動する

return 整数

- 浮動小数版の OpenXM サーバ `ox_sres` が存在しないときは起動し, 他の関数で使われるサーバに設定する.
- 実行ファイルが見つからないときはデバッグモードに入る.
- すでに存在している場合は他の関数で使われるサーバに設定するだけで新たに起動はしない.
- 返される整数は通信のための識別子 .

1.3.6 f_res.gmp

`f_res.gmp()`
:: `ox_sres` を起動する

return 整数

- GMP 版の OpenXM サーバ `ox_sres` が存在しないときは起動し, 他の関数で使われるサーバに設定する.
- 実行ファイルが見つからないときはデバッグモードに入る.
- すでに存在している場合は他の関数で使われるサーバに設定するだけで新たに起動はしない.
- 返される整数は通信のための識別子.

1.3.7 f_res.conv

`f_res.conv(`*List*`)`
:: polytope の凸閉包を求める

return リストのリスト

List 点を表すリストのリスト

- *List* で与えられる polytope の凸閉包を求める.
- OpenXM サーバ `ox_sres` が存在しないときは浮動小数版を起動する.
- 点の座標は整数しか受け付けない.

```
[0] f_res.conv( [ [1,1],[0,0],[0,2],[2,0],[2,2] ] );
[[0,0],[0,2],[2,0],[2,2]]
```

1.3.8 f_res.support

`f_res.support(`*Equation*`,`*Vars*`)`
:: 多項式の support を返す

return リストのリスト

Equation 多項式

Vars 不定元のリスト

- 不定元を *Vars* としたときの多項式 *Equation* の support をリストのリストとして返す.

```
[0] f_res.support( x^2 + x*y + y^2, [x,y] );
[[0,2],[1,1],[2,0]]
[1] f_res.support( x^2 + x*y + y^2, [x,y,z] );
[[0,2,0],[1,1,0],[2,0,0]]
```

1.3.9 f_res.np

`f_res.np(`*Equation*`,`*Vars*`)`
:: Newton polytope を返す

return リストのリスト

Equation 多項式

Vars 不定元のリスト

- 不定元を *Vars* としたときの多項式 *Equation* の Newton polytope をリストのリストとして返す.
- OpenXM サーバ `ox_sres` が存在しないときは浮動小数版を起動する.

```
[0] f_res.np( x^2 + x*y + y^2, [x,y] );
[[0,2],[2,0]]
[1] f_res.np( x^2 + x*y + y^2, [x,y,z] );
[[0,2,0],[2,0,0]]
```

1.3.10 f_res.msum

`f_res.msum(`*Polytopes*`)`
:: polytope たちの Minkowski sum を返す

return リストのリスト

Polytopes リストのリストのリスト

オプション

conv 任意.

- *Polytopes* の成分である polytope による Minkowski sum 内のすべての lattice points を求める.
- *conv* が 1 のときは Minkowski sum の凸閉包を返す. OpenXM サーバ `ox_sres` が存在しないときは浮動小数版を起動する.

```
[0] Q1 = [[0,0],[1,0],[0,1]]$
[1] Q2 = [[0,0],[1,0],[0,1],[1,1]]$
[2] f_res.msum( [Q1,Q1] );
[[0,0],[0,1],[0,2],[1,0],[1,1],[2,0]]
[3] f_res.msum( [Q1,Q1] | conv=1 );
[[0,0],[0,2],[2,0]]
[4] f_res.msum( [Q1,Q1,Q1] | conv=1 );
[[0,0],[0,3],[3,0]]
[5] f_res.msum( [Q1,Q2] );
[[0,0],[0,1],[0,2],[1,0],[1,1],[1,2],[2,0],[2,1]]
[6] f_res.msum( [Q1,Q2] | conv=1 );
[[0,0],[0,2],[1,2],[2,0],[2,1]]
```

1.3.11 f_res.mvol

`f_res.mvol(`*Polytopes*`)`
:: polytope たちの mixed volume を求める

return 整数

Polytopes リストのリストのリスト

- varPolytopes の成分である polytope による mixed volume を求める.
- Mixed volume の定義から polytope の次元と数は等しい必要がある.
- OpenXM サーバ `ox_sres` が存在しないときは浮動小数版を起動する.

```
[0] Q1 = [[0,0],[1,0],[0,1]]$
```

```
[1] Q2 = [[0,0],[1,0],[0,1],[1,1]]$
[2] f_res.mvol( [Q1,Q1] );
1
[3] f_res.mvol( [Q1,Q2] );
2
[4] f_res.mvol( [Q2,Q2] );
2
```

1.3.12 f_res.sres

`f_res.sres(`*Equations*`,`*Vars*`)`
:: sparse resultant の多項式倍を返す

return 多項式

Equations 多項式のリスト

Vars 不定元のリスト

オプション

v リスト

p 素数

sub リスト

- *Equations* の成分の多項式による不定元を *Vars* としたとき Incremental algorithm で計算した resultant の多項式倍を返す.
- オプション *v* は v-distance を表すリストで, 定義されていない場合は [11,12,13,. . .]$ が使われる.
- 行列の rank の計算は GF(p) 上で行なわれ, 行列の中の不定元にはオプションで *sub* で指定されるリストの要素が前から順に代入され評価される. ここで p はオプションの p である. 素数 p が指定されていない場合は 65521 が使われ, リスト *sub* が指定されていない場合は 53,59,. . . の素数が使われる.
- OpenXM サーバ `ox_sres` が存在しないときは浮動小数版を起動する.

```
[0] F0 = a1*x + a2*y + a3$
[1] F1 = b1*x + b2*y + b3$
[2] F2 = c1*x^2 + c2*y^2 + c3 + c4*x*y + c5*x + c6*y$
[3] R = f_res.sres( [F0,F1,F2], [x,y] );
(c3*b2^3-c6*b3*b2^2+c2*b3^2*b2)*a1^2+(((-2*c3*b2^2+c6*b3*b2)*b1+c5*b3*b2^2-c4*b
3^2*b2)*a2+((c6*b2^2-2*c2*b3*b2)*b1-c5*b2^3+c4*b3*b2^2)*a3)*a1+(c3*b2*b1^2-c5*b
3*b2*b1+c1*b3^2*b2)*a2^2+(-c6*b2*b1^2+(c5*b2^2+c4*b3*b2)*b1-2*c1*b3*b2^2)*a3*a2
+(c2*b2*b1^2-c4*b2^2*b1+c1*b2^3)*a3^2
[4] fctr( R );
[[1,1],[b2,1],[(c3*b2^2-c6*b3*b2+c2*b3^2)*a1^2+(((-2*c3*b2+c6*b3)*b1+c5*b3*b2-c
4*b3^2)*a2+((c6*b2-2*c2*b3)*b1-c5*b2^2+c4*b3*b2)*a3)*a1+(c3*b1^2-c5*b3*b1+c1*b3
^2)*a2^2+(-c6*b1^2+(c5*b2+c4*b3)*b1-2*c1*b3*b2)*a3*a2+(c2*b1^2-c4*b2*b1+c1*b2^2
)*a3^2,1]]
```

1.3.13 f_res.dres, f_res.dresM

`f_res.dres(`*Equations*`,`*Vars*`)`
:: Dixon resultant を返す

`f_res.dresM(`*Equations*`,`*Vars*`)`
:: 行列式が Dixon resultant になるような行列を返す

return

`f_res.dres`
多項式

`f_res.dresM`
行列

Equaitons 多項式のリスト

Vars 不定元のリスト

オプション

norsc 任意

rowidx 配列

colidx 配列

p 素数

sub リスト

- *Equations* の 成分の多項式による不定元を *Vars* としたとき Dixon の方法で `f_res.dres` は resultant の多項式倍を, `f_res.dresM` は resultant の多項式倍を行列式にもつ行列を返す.
- Rank Submatrix Construction を行ないたくないときはオプション *norsc* を 1 に設定する.
- この関数は内部で関数 `f_res.submatrix` を呼び出しているので, そのためのオプションはすべて受け付ける.

```
[0] F0 = a1*x + a2*y + a3$
[1] F1 = b1*x + b2*y + b3$
[2] F2 = c1*x^2 + c2*y^2 + c3 + c4*x*y + c5*x + c6*y$
[3] f_res.dresM( [F0,F1,F2], [x,y] );
[ c1*b3*a2-c1*b2*a3 -c2*b3*a1+c4*b3*a2+(c2*b1-c4*b2)*a3 (c3*b2-c6*b3)*a1+(-c3*b
1+c5*b3)*a2+(c6*b1-c5*b2)*a3 ]
[ 0 -c2*b2*a1+c2*b1*a2 -c2*b3*a1+c2*b1*a3 ]
[ -c1*b2*a1+c1*b1*a2 -c4*b2*a1+c4*b1*a2 -c4*b3*a1+c1*b3*a2+(c4*b1-c1*b2)*a3 ]
[4] R = dres( [F0,F1,F2], [x,y] );
(-c3*c2*c1*b2^3+c6*c2*c1*b3*b2^2-c2^2*c1*b3^2*b2)*a1^3+(((3*c3*c2*c1*b2^2-2*c6*
c2*c1*b3*b2+c2^2*c1*b3^2)*b1-c5*c2*c1*b3*b2^2+c4*c2*c1*b3^2*b2)*a2+((-c6*c2*c1*
b2^2+2*c2^2*c1*b3*b2)*b1+c5*c2*c1*b2^3-c4*c2*c1*b3*b2^2)*a3)*a1^2+(((-3*c3*c2*c
1*b2+c6*c2*c1*b3)*b1^2+(2*c5*c2*c1*b3*b2-c4*c2*c1*b3^2)*b1-c2*c1^2*b3^2*b2)*a2^
2+((2*c6*c2*c1*b2-2*c2^2*c1*b3)*b1^2-2*c5*c2*c1*b2^2*b1+2*c2*c1^2*b3*b2^2)*a3*a
2+(-c2^2*c1*b2*b1^2+c4*c2*c1*b2^2*b1-c2*c1^2*b2^3)*a3^2)*a1+(c3*c2*c1*b1^3-c5*c
2*c1*b3*b1^2+c2*c1^2*b3^2*b1)*a2^3+(-c6*c2*c1*b1^3+(c5*c2*c1*b2+c4*c2*c1*b3)*b1
^2-2*c2*c1^2*b3*b2*b1)*a3*a2^2+(c2^2*c1*b1^3-c4*c2*c1*b2*b1^2+c2*c1^2*b2^2*b1)*
```

```
a3^2*a2
[5] fctr(R);
[[-1,1],[c2,1],[c1,1],[b2*a1-b1*a2,1],[(c3*b2^2-c6*b3*b2+c2*b3^2)*a1^2+(((-2*c3
*b2+c6*b3)*b1+c5*b3*b2-c4*b3^2)*a2+((c6*b2-2*c2*b3)*b1-c5*b2^2+c4*b3*b2)*a3)*a1
+(c3*b1^2-c5*b3*b1+c1*b3^2)*a2^2+(-c6*b1^2+(c5*b2+c4*b3)*b1-2*c1*b3*b2)*a3*a2+(
c2*b1^2-c4*b2*b1+c1*b2^2)*a3^2,1]]
```

1.3.14 f_res.dixonpolynomial

`f_res.dixonpolynomial(`*Equations*`,`*Vars*`)`
:: Dixon polynomial を返す

return リスト

Equaitons 多項式のリスト

Vars 不定元のリスト

Equations の 成分の多項式による不定元を *Vars* としたときの Dixon polynomial を計算し, `[ (Dixon polynomial), (新しい変数の配列) ]` というリストを返す. 新しい変数は関数 `uc` によって生成された不定元である. 多項式の数は変数の数よりも一つ多い必要がある.

```
[0] F0 = a1*x + a2*y + a3$
[1] F1 = b1*x + b2*y + b3$
[2] F2 = c1*x^2 + c2*y^2 + c3 + c4*x*y + c5*x + c6*y$
[3] f_res.dixonpolynomial( [F0,F1,F2], [x,y] );
[(-_0*c1*b2*a1+(_0*c1*b1+c1*b3)*a2-c1*b2*a3)*x+(((-_1*c2-_0*c4)*b2-c2*b3)*a1+((
_1*c2+_0*c4)*b1+c4*b3)*a2+(c2*b1-c4*b2)*a3)*y+(c3*b2+(-_1*c2-_0*c4-c6)*b3)*a1+(
-c3*b1+(_0*c1+c5)*b3)*a2+((_1*c2+_0*c4+c6)*b1+(-_0*c1-c5)*b2)*a3,[ _0 _1 ]]
```

1.3.15 f_res.matrixdecomp

`f_res.matrixdecomp( `*Dpoly*`, `*UC*`, `*Vars*` )`
:: Dixon polynomial を行列に分解する.

return リスト

Dpoly 多項式

UC 配列

Vars リスト

- dixonpolynomial *Dpoly* を行が *UC* の monomial, 列が *Vars* の monomial で添字付けられる行列に分解する.
- 戻り値は, `[ (`*UC* の `monomial` の配列`),(`行列`),(`*Vars* の `monomial` の配列`) ]` という形で, それぞれ *sigma_P* $= V\ D\_P\ W$ の V, *D_P*, W を表す.

```
[0] F0 = a1*x + a2*y + a3$
[1] F1 = b1*x + b2*y + b3$
[2] F2 = c1*x^2 + c2*y^2 + c3 + c4*x*y + c5*x + c6*y$
[3] D = f_res.dixonpolynomial( [F0,F1,F2], [x,y] )$
[4] M = f_res.matrixdecomp( D[0], D[1], [x,y] );
[[ 1 _1 _0 ],[ c1*b3*a2-c1*b2*a3 -c2*b3*a1+c4*b3*a2+(c2*b1-c4*b2)*a3 (c3*b2-c6*
b3)*a1+(-c3*b1+c5*b3)*a2+(c6*b1-c5*b2)*a3 ]
```

```
[ 0 -c2*b2*a1+c2*b1*a2 -c2*b3*a1+c2*b1*a3 ]
[ -c1*b2*a1+c1*b1*a2 -c4*b2*a1+c4*b1*a2 -c4*b3*a1+c1*b3*a2+(c4*b1-c1*b2)*a3 ],[
 x y 1 ]]
[5] V = M[0]*M[1]$
[6] D[0] == V[0]*M[2][0]+V[1]*M[2][1]+V[2]*M[2][2];
1
```

1.3.16 f_res.submatrix

`f_res.submatrix(` *Matrix* `)`
:: 引数である行列の rank を持つ部分行列を返す.

return 行列

Matrix 行列

オプション

rowidx 配列

colidx 配列

p 素数

sub リスト

- 行列 *Matrix* の rank を持つ部分行列を返す.
- 行列の rank の計算で行列の中の不定元にはリスト *sub* の値が前から順に代入され GF(p) で評価される. ここで p はオプションの p が使われる.
- 与えられた行列が正則ではないとき部分行列は一意に定まらない. そこでどの行列を指定するかというのを配列 *rowidx*,*colidx* で行なう. 実際には行列 *Matrix*の (i,j) 成分を (*rowidx*[i],*colidx*[j]) 成分と入れ換えているだけである.
- 素数 p が指定されていない場合は 65521 が使われ, リスト *sub* が指定されていない場合は 53,59,dots の素数が使われる.

```
[0] M = newmat( 3, 3, [[1,0,0],[0,a,0],[0,b,0]] );
[ 1 0 0 ]
[ 0 a 0 ]
[ 0 b 0 ]
[1] f_res.submatrix( M );
[ 1 0 ]
[ 0 a ]
[2] f_res.submatrix( M | rowidx=ltov([0,2,1]) );
[ 1 0 ]
[ 0 b ]
```

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents