Sm1 OX Server ヤ≪

Edition : auto generated by oxgentexi on 8 June 2017

OpenXM.org

1 SM1 醇

c sm1 ox 泣若 ox_sm1_forAsir ゃ潟帥 c 若拷∽違茹 h . ∽違 <ゃ 'sm1.rr' y 臂 . 'sm1.rr' '$(OpenXM_HOME)/lib/asir-contrib' . 激鴻 sm1 小簸 違 ц 膊 激鴻 c . 荐膊篁 f 医江簸 紊 膊繻 簸 膊 鍵. sm1 ゃ 吾 OpenXM/doc/kan96xx .

, sm1 server windows ゃ 絽 . cygwin 医 c 純若鴻潟若潟潟ゃ , OpenXM/misc/packages/Windows 紊眼 sm1 泣若 windows c 簸.

鴻 ∽違, 遺違 ュ ゃ . 鴻 綛 違 贋違 c .

腥咲 $X := \mathbf{C} \setminus \{0,1\} = \mathbf{C} \setminus V(x(x-1))$ 若潟≪ 悟召 荐膊 帥. X 抗≪ ゃ 腥咲 c , $x = 0, x = 1$ 障 障篋ゃ 若 1 罨≦ ≪ 吾主召 . c , 1 罨≦ 若潟≪ 悟召 2 c . sm1 0 罨≦ 潟≪ 悟召 1 罨≦ 潟≪ 悟召 罨≦膈.

```
[283] sm1.deRham([x*(x-1),[x]]);
[1,2]
```

The author of sm1 : Nobuki Takayama, takayama@math.sci.kobe-u.ac.jp

The author of sm1 packages : Toshinori Oaku, oaku@twcu.ac.jp

Reference: [SST] Saito, M., Sturmfels, B., Takayama, N., Grobner Deformations of Hypergeometric Differential Equations, 1999, Springer. http://www.math.kobe-u.ac.jp/KAN

1.1 ox_sm1_forAsir 泣若

1.1.1 ox_sm1_forAsir

ox_sm1_forAsir
:: asir sm1 泣若.

- 泣若 ox_sm1_forAsir asir 潟潟 sm1.start ц 儀 sm1 泣若 c .
  罔羣荐 c ,
  ox_sm1_forAsir = '$(OpenXM_HOME)/lib/sm1/bin/ox_sm1' + '$(OpenXM_HOME)/lib/sm1/callsm1.s
  (macro file)
  + '$(OpenXM_HOME)/lib/sm1/callsm1b.sm1' (macro file)
  c , <ゃ , 筝 current directory, $(LOAD_SM1_PATH), $(OpenXM_HOME)/lib/sm1, /usr/local/lib/sm1 c .
- 違 若 若: sm1 茯 粋 昭 ц 泣 若 簸 罨 < <ゃ 荀 '$(OpenXM_HOME)/src/kxx/oxserver00.c', '$(OpenXM_HOME)/src/kxx/sm1stackmachine.c'

1.2 醇遺荀

1.2.1 sm1.start

sm1.start()
:: localhost ox_sm1_forAsir 鴻帥若.

return 贋

- localhost ox_sm1_forAsir 鴻帥若. 泣若 ox_sm1_forAsir ヨ 激祉.
- Xm_noX = 1 泣若 ox_sm1_forAsir 亥 c 潟 儀 c .

- 潟潟 ord 育綺罩 c 荐 綽荀 . , 紊 x dx 箏 小簸 (dx $\partial/\partial x$ 綽) ц膊 , sm1 泣若 医激 , 紊 dx 喝眼 紊 x 窪眼 ゃ 皿絎 . 罷< c , 紊 cc sm1 c 膊 .
- a z , d o ゃ, , x0, ..., x20, y0, ..., y20, z0, ..., z20 , 若 y 小簸 違 違 簸帥 (cf. Sm1_ord_list in sm1).
- 荔ヨ 激 static Sm1_proc 主. ヨ 激 ∽ sm1.get_Sm1_proc() c c .

```
[260] ord([da,a,db,b]);
[da,a,db,b,dx,dy,dz,x,y,z,dt,ds,t,s,u,v,w,
......... omit ..................
]
[261] a*da;
a*da
[262] cc*dcc;
dcc*cc
[263] sm1.mul(da,a,[a]);
a*da+1
[264] sm1.mul(a,da,[a]);
a*da
```

ox_launch, sm1.push_int0, sm1.push_poly0, ord

1.2.2 sm1.sm1

sm1.sm1(*p*,*s*)
:: 泣若 sm1 潟潟 *s* 絎茵 .

return

p

s 絖

- 荔ヨ *p* sm1 泣若 潟潟 *s* 絎茵 若. (罷< c , 荔ヨ 0)

```
[261] sm1.sm1(0," ( (x-1)^2 ) . ");
0
[262] ox_pop_string(0);
x^2-2*x+1
[263] sm1.sm1(0," [(x*(x-1))  [(x)]] deRham ");
0
[264] ox_pop_string(0);
[1 , 2]
```

sm1.start, ox_push_int0, sm1.push_poly0, sm1.get_Sm1_proc().

1.2.3 sm1.push_int0

sm1.push_int0(*p*,*f*)
:: 吾 c *f* 荔ュ *p* 泣若檎.

return

p

f 吾c

- `type(f)` 2 (絽医綣) , *f* 絖 (type == 7) , `ox_push_cmo` 泣若檎.
- `type(f)` 0 (zero) , 泣若箏c , 32 bit 贋違 B. `ox_push_cmo(P,0)` 泣若 `CMO_NULL` , 泣若眼c , 32 bit 贋違c .
- `sm1` 贋違, 32 bit 贋違 bignum c . `type(f)` 1 () , ∞違 32 bit integer 泣若 . `ox_push_cmo(p,1234)` bignum 1234 `sm1` 泣若 絵.
- 翫 `ox_push_cmo` 若水 若喝冴.

```
[219] P=sm1.start();
0
[220] sm1.push_int0(P,x*dx+1);
0
[221] A=ox_pop_cmo(P);
x*dx+1
[223] type(A);
7   (string)
[271] sm1.push_int0(0,[x*(x-1),[x]]);
0
[272] ox_execute_string(0," deRham ");
0
[273] ox_pop_cmo(0);
[1,2]
```

Reference
ox_push_cmo

1.2.4 sm1.gb

`sm1.gb([f,v,w]|proc=p,sorted=q,dehomogenize=r)`
:: *v* 箏 小簸 違 *f* 違 阪荐膊.

`sm1.gb_d([f,v,w]|proc=p)`
:: *v* 箏 小簸 違 *f* 違 阪荐膊. 腔e綣 鴻ф祉.

return 鴻

p, q, r

f, v, w 鴻

- *v* 箏 小簸 違 *f* 違 阪荐膊.
- Weight *w* ャ . ャ翫, graded reverse lexicographic order ゃc 阪荐膊.
- `sm1.gb` 祉ゃ *f* 違 阪潟ゃ激c ≪≪ (*w*) 障 ゃ激< 綣 (*w* 箏) 鴻c .
- `sm1.gb_d` e綣 鴻ф祉. 紊綣 賢 障≪≪ 違 阪荐膊 綺c純若 . 祉ゃ [紊医鴻, 綺茵, 違 阪, ゃ激c ≪≪ 障 ゃ激< 綣] c .
- Term order c 綺箏翫, 罨≦ ゃ 撮違c違 阪荐膊 (SST Section 1.2 荀). 罨≦紊 h 腔 .
- 激c *q* 祉 , 3 祉ゃ , 違 阪潟ゃ激< 鴻 箏綺c純若≪≪ 祉. 障 綣, 絖ц ; 障. 激c *r* 祉 , 祉紊綣 dehomogenize (h 1 篁e ャ).
- Reduced 違 弱阪障 in_w 荐膊 , ∞違 茵 sm1.auto_reduce(1) 絎茵 .

```
[293] sm1.gb([[x*dx+y*dy-1,x*y*dx*dy-2],[x,y]]);
[[x*dx+y*dy-1,y^2*dy^2+2],[x*dx,y^2*dy^2]]
```

筝 , $\{x\partial_x + y\partial_y - 1, y^2\partial_y^2 + 2\}$ $1 \leq \partial_y \leq \partial_x \leq y \leq x \leq \cdots$ c graded reverse lexicographic order ≪違 阪 c . $\{x\partial_x, y^2\partial_y\}$ 違 阪 絲障 leading monomial (initial monomial) c .

```
[294] sm1.gb([[dx^2+dy^2-4,dx*dy-1],[x,y],[[dx,50,dy,2,x,1]]]);
[[dx+dy^3-4*dy,-dy^4+4*dy^2-1],[dx,-dy^4]]
```

筝 や ≪≪ $m = x^a y^b \partial_x^c \partial_y^d$ $m' = x^{a'} y^{b'} \partial_x^{c'} \partial_y^{d'}$ weight vector `(dx,dy,x,y) = (50,2,1,0)` 莠 (や障 m $50c+2d+a > 50c'+2d'+a'$ m' 紊 c) 罹< 莠 y 莢や reverse lexicographic order ф莠 (や障 $50c+2d+a = 50c'+2d'+a'$ reverse lexicographic order ф莠).

```
[294] F=sm1.gb([[dx^2+dy^2-4,dx*dy-1],[x,y],[[dx,50,dy,2,x,1]]]|sorted=1);
      map(print,F[2][0])$
      map(print,F[2][1])$
[595]
   sm1.gb([["dx*(x*dx +y*dy-2)-1","dy*(x*dx + y*dy -2)-1"],
             [x,y],[[dx,1,x,-1],[dy,1]]]);

[[x*dx^2+(y*dy-h^2)*dx-h^3,x*dy*dx+y*dy^2-h^2*dy-h^3,h^3*dx-h^3*dy],
 [x*dx^2+(y*dy-h^2)*dx,x*dy*dx+y*dy^2-h^2*dy-h^3,h^3*dx]]

[596]
   sm1.gb_d([["dx (x dx +y dy-2)-1","dy (x dx + y dy -2)-1"],
             "x,y",[[dx,1,x,-1],[dy,1]]]);
[[[e0,x,y,H,E,dx,dy,h],
 [[0,-1,0,0,0,1,0,0],[0,0,0,0,0,0,1,0],[1,0,0,0,0,0,0,0],
  [0,1,1,1,1,1,1,0],[0,0,0,0,0,0,-1,0],[0,0,0,0,0,-1,0,0],
  [0,0,0,0,-1,0,0,0],[0,0,0,-1,0,0,0,0],[0,0,-1,0,0,0,0,0],
  [0,0,0,0,0,0,0,1]]],
[[(1)*<<0,0,1,0,0,1,1,0>>+(1)*<<0,1,0,0,0,2,0,0>>+(-1)*<<0,0,0,0,0,1,0,2>>+(-1)*
<<0,0,0,0,0,0,0,3>>,(1)*<<0,0,1,0,0,0,2,0>>+(1)*<<0,1,0,0,0,1,1,0>>+(-1)*<<0,0,0
,0,0,0,1,2>>+(-1)*<<0,0,0,0,0,0,0,3>>,(1)*<<0,0,0,0,0,1,0,3>>+(-1)*<<0,0,0,0,0,0
,1,3>>],
 [(1)*<<0,0,1,0,0,1,1,0>>+(1)*<<0,1,0,0,0,2,0,0>>+(-1)*<<0,0,0,0,0,1,0,2>>,(1)*<
<0,0,1,0,0,0,2,0>>+(1)*<<0,1,0,0,0,1,1,0>>+(-1)*<<0,0,0,0,0,0,1,2>>+(-1)*<<0,0,0
,0,0,0,0,3>>,(1)*<<0,0,0,0,0,1,0,3>>]]]
```

sm1.auto_reduce, sm1.reduction, sm1.rat_to_p

1.2.5 sm1.deRham

sm1.deRham([*f*,*v*]|proc=*p*)

:: 腥咲 C^n - (the zero set of f=0) 若潟≪ 悟召 荐膊 泣若 若.

return 鴻

p

f 絖 障 紊綣

v 鴻

- 醇違 X = C^n \ V(*f*) 若潟≪ 悟召 荐膊. , [dim H^0(X,C), dim H^1(X,C), dim H^2(X,C), ..., dim H^n(X,C)] 祉.
- *v* 違 鴻. n = `length(v)` c .
- `sm1.deRham` 膊罘 羣紊ч 戎 . `sm1.deRham(0,[x*y*z*(x+y+z-1)*(x-y),[x,y,z]])` 膊c 絽吾 ぇ紊c .
- b-∞違 鴻合茹f , `ox_asir` `ox_sm1_forAsir` 筴睡 鴻c . 潟潟 `sm1(0,"[(parse) (oxasir.sm1) pushfile] extension");` , `ox_asir` 篆<≪吾ャ若 若 . 潟潟 `ox_asir_forAsir` 鴻帥若 茵 .
- `sm1.deRham` `ox_reset(sm1.get_Sm1_proc());` т賢 , 篁ュ sm1 泣若罔羣≪若 ャ篋 筴翫 , 潟潟 `ox_shutdown(sm1.get_Sm1_proc());` , `ox_sm1_forAsir` 箏 shutdown 鴻帥若鴻絎 c .

```
[332] sm1.deRham([x^3-y^2,[x,y]]);
[1,1,0]
[333] sm1.deRham([x*(x-1),[x]]);
[1,2]
```

`sm1.start`, `deRham` (sm1 command)

`Algorithm:`

Oaku, Takayama, An algorithm for de Rham cohomology groups of the complement of an affine variety via D-module computation, Journal of pure and applied algebra 139 (1999), 201–233.

1.2.6 `sm1.hilbert`

`sm1.hilbert([`*f*`,`*v*`]|proc=`*p*`)`
:: 紊綣 *f* 紊綣荐膊.

`hilbert_polynomial(`*f*`,`*v*`)`
:: 紊綣 *f* 紊綣荐膊.

return 紊綣

p

f, *v* 鴻

- 紊綣 *f* *v* 紊綣 h(k) 荐膊.
- h(k) = dim_Q F_k/I \cap F_k F_k 違 k 篁ヤ c 紊綣 c . I 綣 *f* x ゃ≪ c .
- sm1.hilbert 若: 合荐膊 *f* ≪≪ 鴻. 絎, 醇違 障 *f* 違 阪荐膊, initial monomial 紊綣荐膊. c , ュ *f* c 違 阪 醇違 c 箏綺 違 阪 膊 . ♂c , `sm1` 紊綣違 阪荐膊 `asir` .

```
[346] load("katsura")$
[351] A=hilbert_polynomial(katsura(5),[u0,u1,u2,u3,u4,u5]);
32
```

```
[279] load("katsura")$
```

```
[280] A=gr(katsura(5),[u0,u1,u2,u3,u4,u5],0)$
[281] dp_ord();
0
[282] B=map(dp_ht,map(dp_ptod,A,[u0,u1,u2,u3,u4,u5]));
[(1)*<<1,0,0,0,0,0>>,(1)*<<0,0,0,2,0,0>>,(1)*<<0,0,1,1,0,0>>,(1)*<<0,0,2,0,0,0>>,
 (1)*<<0,1,1,0,0,0>>,(1)*<<0,2,0,0,0,0>>,(1)*<<0,0,0,1,1,1>>,(1)*<<0,0,0,1,2,0>>,
 (1)*<<0,0,1,0,2,0>>,(1)*<<0,1,0,0,2,0>>,(1)*<<0,1,0,1,1,0>>,(1)*<<0,0,0,0,2,2>>,
  (1)*<<0,0,1,0,1,2>>,(1)*<<0,1,0,0,1,2>>,(1)*<<0,1,0,1,0,2>>,(1)*<<0,0,0,0,3,1>>,
  (1)*<<0,0,0,0,4,0>>,(1)*<<0,0,0,0,1,4>>,(1)*<<0,0,0,1,0,4>>,(1)*<<0,0,1,0,0,4>>,
 (1)*<<0,1,0,0,0,4>>,(1)*<<0,0,0,0,0,6>>]
[283] C=map(dp_dtop,B,[u0,u1,u2,u3,u4,u5]);
[u0,u3^2,u3*u2,u2^2,u2*u1,u1^2,u5*u4*u3,u4^2*u3,u4^2*u2,u4^2*u1,u4*u3*u1,
 u5^2*u4^2,u5^2*u4*u2,u5^2*u4*u1,u5^2*u3*u1,u5*u4^3,u4^4,u5^4*u4,u5^4*u3,
 u5^4*u2,u5^4*u1,u5^6]
[284] sm1.hilbert([C,[u0,u1,u2,u3,u4,u5]]);
32
```

sm1.start, sm1.gb, longname

1.2.7 sm1.genericAnn

sm1.genericAnn([*f*,*v*]|proc=*p*)

:: *f*^s 帥緇 合綣 . *v* 違 鴻c . , s *v*[0] c , *f* rest(*v*) 箏 綣c .

return 鴻

p

f 紊綣

v 鴻

- 醇違, *f*^s 帥緇 合綣 . *v* 違 鴻c . , s *v*[0] c , *f* rest(*v*) 箏 綣c .

```
[595] sm1.genericAnn([x^3+y^3+z^3,[s,x,y,z]]);
[-x*dx-y*dy-z*dz+3*s,z^2*dy-y^2*dz,z^2*dx-x^2*dz,y^2*dx-x^2*dy]
```

sm1.start

1.2.8 sm1.wTensor0

sm1.wTensor0([*f*,*g*,*v*,*w*]|proc=*p*)

:: *f* *g* D-module 0 罨<潟純 荐膊.

return 鴻

p

f, *g*, *v*, *w* 鴻

- *f* *g* D-臂ゃ 0 罨<潟純 荐膊.
- *v* 違 鴻c . *w* weight 鴻c . 贋 *w*[i] *v*[i] weight c .
- sm1.wTensor0 ox_sm1 wRestriction0 c . wRestriction0 , generic weight *w* 狗荐膊 . Weight *w* generic c 膊 若y 罩≪.
- F G *f* *g* B . 頑T , 0 罨< 潟純 ∞ FG 帥緇 合綣膤祉c .

- ュ f, g D 窪ゃ≪ c c , 箏 , 阪 怨臂 D^r 臂ゃ c .

```
[258] sm1.wTensor0([[x*dx -1, y*dy -4],[dx+dy,dx-dy^2],[x,y],[1,2]]);
[[-y*x*dx-y*x*dy+4*x+y],[5*x*dx^2+5*x*dx+2*y*dy^2+(-2*y-6)*dy+3],
 [-25*x*dx+(-5*y*x-2*y^2)*dy^2+((5*y+15)*x+2*y^2+16*y)*dy-20*x-8*y-15],
 [y^2*dy^2+(-y^2-8*y)*dy+4*y+20]]
```

1.2.9 sm1.reduction

`sm1.reduction([`*f*`,`*g*`,`*v*`,`*w*`]|proc=`*p*`)`

::

return 鴻

f 紊綣

g, v, w 鴻

p (ox_sm1 祉合)

- 醇違 *f* homogenized ゃ 撮違 , 紊綣 *g* x 亜 (reduce) ; ゃ障, 醇違, *f* 牙≪ 眼 冴 . 紊育 *v* ф絎. *w* 綺絎 ゃ c , ャ . `sm1.reduction_noH` , Weyl algebra .
- 祉ゃ 就 : [r,c0,[c1,...,cm],g] *g*=[g1, ..., gm] c , c0 f + c1 g1 + ... + cm gm = r . r/c0 normal form c .
- 醇違, 簸罨♂ reducible 脊≦.
- 醇 sm1.reduction_d(P,F,G) sm1.reduction_noH_d(P,F,G) , e綣 c .

```
[259] sm1.reduction([x^2+y^2-4,[y^4-4*y^2+1,x+y^3-4*y],[x,y]]);
[x^2+y^2-4,1,[0,0],[y^4-4*y^2+1,x+y^3-4*y]]
[260] sm1.reduction([x^2+y^2-4,[y^4-4*y^2+1,x+y^3-4*y],[x,y],[[x,1]]]);
[0,1,[-y^2+4,-x+y^3-4*y],[y^4-4*y^2+1,x+y^3-4*y]]
```

`sm1.start`, `d_true_nf`

1.2.10 sm1.xml_tree_to_prefix_string

`sm1.xml_tree_to_prefix_string(`*s*`|proc=`*p*`)`

:: XML ф吾 OpenMath ; *s* 臀 羈 .

return String

p Number

s String

- XML ф吾 OpenMath ; *s* 臀 羈 .
- 醇違 om_* ョ皿鴻 c .
- `om_xml_to_cmo(OpenMath Tree Expression)` CMO_TREE 祉. `asir` CMO 障泣 若 .
- `java` 茵医綽荀. (, /usr/local/jdk1.1.8/bin 潟潟泣若鴻 ャ .)

```
[263] load("om");
1
[270] F=om_xml(x^4-1);
control: wait OX
```

```
Trying to connect to the server... Done.
<OMOBJ><OMA><OMS name="plus" cd="basic"/><OMA>
<OMS name="times" cd="basic"/><OMA>
<OMS name="power" cd="basic"/><OMV name="x"/><OMI>4</OMI></OMA>
<OMI>1</OMI></OMA><OMA><OMS name="times" cd="basic"/><OMA>
<OMS name="power" cd="basic"/><OMV name="x"/><OMI>0</OMI></OMA>
<OMI>-1</OMI></OMA></OMA></OMOBJ>
[271] sm1.xml_tree_to_prefix_string(F);
basic_plus(basic_times(basic_power(x,4),1),basic_times(basic_power(x,0),-1))
```

om_*, OpenXM/src/OpenMath, eval_str

1.2.11 sm1.syz

sm1.syz([f,v,w]|proc=p)
:: v 箏 小簸 違 f syzygy 荐膊.

return 鴻

p

f, *v*, *w* 鴻

- 祉ゃ 就 : [s,[g, m, t]]. s f v 紊違 緇 簸 違 syzygy c . g f weight vector w ≪違 阪c . m ュ茵 f 違 阪 g 後茵c . t 違 阪 g syzygy c . 障 , 罨< 綣 : g = m f , s f = 0.
- Weight w ャ . ャ翫, graded reverse lexicographic order ゃc 阪荐膊.
- Term order c 綺箏翫, 罨≦ ゃ 撮違c違 阪荐膊 (SST Section 1.2 荀). 罨≦紊 h 腔 .

```
[293] sm1.syz([[x*dx+y*dy-1,x*y*dx*dy-2],[x,y]]);
[[[y*x*dy*dx-2,-x*dx-y*dy+1]],    generators of the syzygy
 [[[x*dx+y*dy-1],[y^2*dy^2+2]],   grobner basis
  [[1,0],[y*dy,-1]],              transformation matrix
 [[y*x*dy*dx-2,-x*dx-y*dy+1]]]]
[294]sm1.syz([[x^2*dx^2+x*dx+y^2*dy^2+y*dy-4,x*y*dx*dy-1],[x,y],[[dx,-1,x,1]]]);
[[[y*x*dy*dx-1,-x^2*dx^2-x*dx-y^2*dy^2-y*dy+4]], generators of the syzygy
 [[[x^2*dx^2+h^2*x*dx+y^2*dy^2+h^2*y*dy-4*h^4],[y*x*dy*dx-h^4], GB
  [h^4*x*dx+y^3*dy^3+3*h^2*y^2*dy^2-3*h^4*y*dy]],
 [[1,0],[0,1],[y*dy,-x*dx]],     transformation matrix
 [[y*x*dy*dx-h^4,-x^2*dx^2-h^2*x*dx-y^2*dy^2-h^2*y*dy+4*h^4]]]]
```

1.2.12 sm1.mul

sm1.mul(f,g,v|proc=p)
:: sm1 泣若 f g v 箏 小簸 違c c 若.

return 紊綣障 鴻

p

f, *g* 紊綣障 鴻

v 鴻

- sm1 泣若 *f* *g* *v* 等 小篏 違c c 若.
- `sm1.mul_h` homogenized Weyl 篁f亥.
- BUG: `sm1.mul(p0*dp0,1,[p0])` `dp0*p0+1` 祉. d 紊違緇 育綺 c , ∞違膈祉 .

```
[277] sm1.mul(dx,x,[x]);
x*dx+1
[278] sm1.mul([x,y],[1,2],[x,y]);
x+2*y
[279] sm1.mul([[1,2],[3,4]],[[x,y],[1,2]],[x,y]);
[[x+2,y+4],[3*x+4,3*y+8]]
```

1.2.13 sm1.distraction

`sm1.distraction([`*f*`,`*v*`,`*x*`,`*d*`,`*s*`]|proc=`*p*`)`
:: `sm1` *f* distraction 荐膊 .

return 鴻

p

f 紊綣

v,x,d,s 鴻

- 荔ュ *p* `sm1` 泣若, *f* distraction *v* 等 小篏 違ц膊 .
- *x* , *d* , , distract 鴻 x 紊, d 紊違 鴻. Distraction , *s* 紊違 茵 .
- Distraction x*dx x x舟 c . 荅潟 Saito, Sturmfels, Takayama : Grobner Deformations of Hypergeometric Differential Equations page 68 荀.

```
[280] sm1.distraction([x*dx,[x],[x],[dx],[x]]);
x
[281] sm1.distraction([dx^2,[x],[x],[dx],[x]]);
x^2-x
[282] sm1.distraction([x^2,[x],[x],[dx],[x]]);
x^2+3*x+2
[283] fctr(@);
[[1,1],[x+1,1],[x+2,1]]
[284] sm1.distraction([x*dx*y+x^2*dx^2*dy,[x,y],[x],[dx],[x]]);
(x^2-x)*dy+x*y
```

`distraction2(sm1)`,

1.2.14 sm1.gkz

`sm1.gkz([`*A*`,`*B*`]|proc=`*p*`)`
:: 茵 *A* <若 *B* GKZ 膤 (A-hypergeometric system) .

return 鴻

p

A, *B* 鴻

- 茵 *A* <若 *B* GKZ 膤 (A-hypergeometric system) .

```
[280] sm1.gkz([  [[1,1,1,1],[0,1,3,4]],  [0,2] ]);
[[x4*dx4+x3*dx3+x2*dx2+x1*dx1,4*x4*dx4+3*x3*dx3+x2*dx2-2,
 -dx1*dx4+dx2*dx3,-dx2^2*dx4+dx1*dx3^2,dx1^2*dx3-dx2^3,-dx2*dx4^2+dx3^3],
 [x1,x2,x3,x4]]
```

1.2.15 sm1.mgkz

sm1.mgkz([*A*,*W*,*B*]|proc=*p*)
:: 茵 *A*, weight *W* <若 *B* modified GKZ 膤 (A-hypergeometric system) .

return 鴻

p

A, *W*, *B* 鴻

- 茵 *A*, weight vector *W* <若 *B* modified GKZ 膤 (A-hypergeometric system) .
- http://arxiv.org/abs/0707.0043

```
[280] sm1.mgkz([ [[1,2,3]], [1,2,1], [a/2]]);
[[6*x3*dx3+4*x2*dx2+2*x1*dx1-a,-x4*dx4+x3*dx3+2*x2*dx2+x1*dx1,
  -dx2+dx1^2,-x4^2*dx3+dx1*dx2],[x1,x2,x3,x4]]

Modified A-hypergeometric system for
A=(1,2,3), w=(1,2,1), beta=(a/2).
```

1.2.16 sm1.appell1

sm1.appell1(a|proc=*p*)
:: F_1 障 F_D 綽合綣膤祉祉.

return 鴻

p

a 鴻

- Appell ∽ F_1 n 紊医 c Lauricella ∽ F_D(a,b1,b2,...,bn,c;x1,...,xn) 帥緇 合綣膤祉祉. , a =(a,c,b1,...,bn). <若帥 違 c .
- sm1 ∽ appell1 吟 c , <若帥違絖綣 翫 罩 c .

```
[281] sm1.appell1([1,2,3,4]);
[[((-x1+1)*x2*dx1-3*x2)*dx2+(-x1^2+x1)*dx1^2+(-5*x1+2)*dx1-3,
  (-x2^2+x2)*dx2^2+((-x1*x2+x1)*dx1-6*x2+2)*dx2-4*x1*dx1-4,
  ((-x2+x1)*dx1+3)*dx2-4*dx1],       equations
 [x1,x2]]                            the list of variables

[282] sm1.gb(@);
[[((-x2+x1)*dx1+3)*dx2-4*dx1,((-x1+1)*x2*dx1-3*x2)*dx2+(-x1^2+x1)*dx1^2
```

```
 +(-5*x1+2)*dx1-3,(-x2^2+x2)*dx2^2+((-x2^2+x1)*dx1-3*x2+2)*dx2
 +(-4*x2-4*x1)*dx1-4,
 (x2^3+(-x1-1)*x2^2+x1*x2)*dx2^2+((-x1*x2+x1^2)*dx1+6*x2^2
+(-3*x1-2)*x2+2*x1)*dx2-4*x1^2*dx1+4*x2-4*x1],
 [x1*dx1*dx2,-x1^2*dx1^2,-x2^2*dx1*dx2,-x1*x2^2*dx2^2]]

[283] sm1.rank(sm1.appell1([1/2,3,5,-1/3]));
3

[285] Mu=2$ Beta = 1/3$
[287] sm1.rank(sm1.appell1([Mu+Beta,Mu+1,Beta,Beta,Beta]));
4
```

1.2.17 sm1.appell4

`sm1.appell4(a|proc=`*p*`)`
:: F_4 障 F_C 綽合綣膤祉祉.

return 鴻

p

a 鴻

- Appell ∞ F_4 n 紊医 c Lauricella ∞ F_C(a,b,c1,c2,...,cn;x1,...,xn) 帥緇 合綣膤祉祉. , a =(a,b,c1,...,cn). <若帥 違 c .
- sm1 ∞ appell1 吟 c , <若帥違絖綣 翫 罩 c .

```
[281] sm1.appell4([1,2,3,4]);
  [[-x2^2*dx2^2+(-2*x1*x2*dx1-4*x2)*dx2+(-x1^2+x1)*dx1^2+(-4*x1+3)*dx1-2,
  (-x2^2+x2)*dx2^2+(-2*x1*x2*dx1-4*x2+4)*dx2-x1^2*dx1^2-4*x1*dx1-2],
                                                          equations
    [x1,x2]]                                  the list of variables

[282] sm1.rank(@);
4
```

1.2.18 sm1.rank

`sm1.rank(a|proc=`*p*`)`
:: 緇 合綣膤 *a* holonomic rank 祉.

return

p

a 鴻

- 緇 合綣膤 *a* , generic point c 茹 c 祉. , holonomic rank 若.

- *a* 小簸 鴻 違 鴻 .
- *a* regular holonomic `sm1.rrank` holonomic rank 祉. c宴 ∞違 鴻 `sm1.rank` .

```
[284] sm1.gkz([ [[1,1,1,1],[0,1,3,4]], [0,2] ]);
[[x4*dx4+x3*dx3+x2*dx2+x1*dx1,4*x4*dx4+3*x3*dx3+x2*dx2-2,
  -dx1*dx4+dx2*dx3, -dx2^2*dx4+dx1*dx3^2,dx1^2*dx3-dx2^3,-dx2*dx4^2+dx3^3],
 [x1,x2,x3,x4]]
[285] sm1.rrank(@);
4

[286] sm1.gkz([ [[1,1,1,1],[0,1,3,4]], [1,2]]);
[[x4*dx4+x3*dx3+x2*dx2+x1*dx1-1,4*x4*dx4+3*x3*dx3+x2*dx2-2,
 -dx1*dx4+dx2*dx3,-dx2^2*dx4+dx1*dx3^2,dx1^2*dx3-dx2^3,-dx2*dx4^2+dx3^3],
 [x1,x2,x3,x4]]
[287] sm1.rrank(@);
5
```

1.2.19 sm1.auto_reduce

`sm1.auto_reduce(`*s*`|proc=`*p*`)`
:: "AutoReduce" *s* |紡.

祉

p

s

- *s* 1 , 篁ュ荐膊違 阪 鴻, reduced 違 阪 .
- *s* 0 , 荐膊違 阪 reduced 違 阪 . < 若 .

1.2.20 sm1.slope

`sm1.slope(`*ii*`,`*v*`,`*f_filtration*`,`*v_filtration*`|proc=`*p*`)`
:: 緇 合綣膤 *ii* slope 祉.

return

p

ii 鴻 (合綣)

v 鴻 (紊)

f_filtration 鴻 (weight vector)

v_filtration
鴻 (weight vector)

- `sm1.slope` 緇 合綣膤 *ii* V filtration *v_filtration* φ紡劼綛渇≪ 何c (geomeric) slope 荐膊.
- *v* 違 鴻.

- 祉ゃ, 鴻 鴻c. 鴻 1 荀膣 slope, 膃 2 荀膣, weight vector 綽 microcharacteristic variety bihomogeneous c .

Algorithm: "A.Assi, F.J.Castro-Jimenez and J.M.Granger, How to calculate the slopes of a D-module, Compositio Math, 104, 1-17, 1996" 帥. Slope s' ャ 臂 c, 膃 激莢 , 違 , Slope 偽絲上 -s' 祉. ゃ障 pF+qV micro 号y 罕筴 gap c , -s'=q/p 祉. 菴 c s=1-1/s' slope 若c. 茹c O(s) . s 1<= s 羣. r=s-1=-1/s' kappa=1/r=-s' c. 違 Borel and Laplace 紊 1/Gamma(1+m*r) factor, exp(-tau^kappa) 戎.

```
[284] A= sm1.gkz([  [[1,2,3]],  [-3] ]);


[285] sm1.slope(A[0],A[1],[0,0,0,1,1,1],[0,0,-1,0,0,1]);

[286] A2 = sm1.gkz([ [[1,1,1,0],[2,-3,1,-3]], [1,0]]);
     (* This is an interesting example given by Laura Matusevich,
        June 9, 2001 *)

[287] sm1.slope(A2[0],A2[1],[0,0,0,0,1,1,1,1],[0,0,0,-1,0,0,0,1]);


     sm.gb
```

1.2.21 sm1.ahg

`sm1.ahg(A)`
: It idential with sm1.gkz(A).

1.2.22 sm1.bfunction

`sm1.bfunction(F)`
: It computes the global b-function of F.

Description:

It no longer calls sm1's original bfunction. Instead, it calls asir "bfct".

Algorithm:

M.Noro, Mathematical Software, icms 2002, pp.147–157.

Example:

```
sm1.bfunction(x^2-y^3);
```

1.2.23 sm1.call_sm1

`sm1.call_sm1(F)`
: It executes F on the sm1 server. See also sm1.

1.2.24 sm1.ecart_homogenize01Ideal

`sm1.ecart_homogenize01Ideal(A)`
: It (0,1)-homogenizes the ideal $A[0]$. Note that it is not an elementwise homogenization.

Example:

```
input1
  F=[(1-x)*dx+1]$ FF=[F,"x,y"]$
  sm1.ecart_homogenize01Ideal(FF);
intput2
  F=sm1.appell1([1,2,3,4]);
  sm1.ecart_homogenize01Ideal(F);
```

1.2.25 sm1.ecartd_gb

`sm1.ecartd_gb(A)`

: It returns a standard basis of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. If the option rv="dp" (return_value="dp") is given, the answer is returned in distributed polynomials.

Note. Functions in the category ecart changes the global environment in the sm1 server. If you interrupted these functions, run sm1.ecartd_gb with a small input and terminate it normally. Then, the global environment is reset to the normal state. Reference. G. Granger, T. Oaku, N. Takayama, Tangent cone algorithm for homogeized differential operators, 2005.

Example:

```
input1
  F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  sm1.ecartd_gb(FF);
output1
  [[(-2*x-2*y+2)*dx+h,(-2*x-2*y+2)*dy+h],[(-2*x-2*y+2)*dx,(-2*x-2*y+2)*dy]]
input2
  F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]],["noAutoHomogenize",1]]$
  sm1.ecartd_gb(FF);
input3
  F=[[x^2,y+x],[x+y,y^3], [2*x^2+x*y,y+x+x*y^3]]$
  FF=[F,"x,y",[[dx,1,dy,1],[x, -1, y, -1,dx, 1, dy, 1]],
            ["degreeShift",[[0,1],[-3,1]]]]$
  sm1.ecartd_gb(FF);
```

1.2.26 sm1.ecartd_gb_oxRingStructure

`sm1.ecartd_gb_oxRingStructure()`

: It returns the oxRingStructure of the most recent ecartd_gb computation.

1.2.27 sm1.ecartd_isSameIdeal_h

`sm1.ecartd_isSameIdeal_h(F)`

: Here, F=[II,JJ,V]. It compares two ideals II and JJ in h[0,1](D)_alg.

Example:

```
input
  II=[(1-x)^2*dx+h*(1-x)]$ JJ = [(1-x)*dx+h]$
  V=[x]$
  sm1.ecartd_isSameIdeal_h([II,JJ,V]);
```

1.2.28 sm1.ecartd_reduction

`sm1.ecartd_reduction(F,A)`
: It returns a reduced form of F in terms of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. When the output is G, $G[3]$ is F and G[0]-(G[1]*A-sum(k,G[2][k]*G[3][k]))=0 holds. F must be (0,1)-hohomogenized (see sm1.ecart_homogenize01Ideal). This function does not check if the given order is admissible for the ecart reduction. To do this check, use sm1.ecartd_gb.

Example:

```
input
  F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  G=sm1.ecartd_reduction(dx+dy,FF);
  G[0]-(G[1]*(dx+dy)+G[2][0]*F[0]+G[2][1]*F[1]);
```

1.2.29 sm1.ecartd_reduction_noh

`sm1.ecartd_reduction_noh(F,A)`
: It returns a reduced form of F in terms of A by using a tangent cone algorithm. h[0,1](D)-homogenization is NOT used. $A[0]$ must not contain the variable h.

Example:

```
F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  sm1.ecartd_reduction_noh(dx+dy,FF);
```

1.2.30 sm1.ecartd_syz

`sm1.ecartd_syz(A)`
: It returns a syzygy of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. If the option rv="dp" (return_value="dp") is given, the answer is returned in distributed polynomials. The return value is in the format [s,[g,m,t]]. s is the generator of the syzygies, g is the Grobner basis, m is the translation matrix from the generators to g. t is the syzygy of g.

Example:

```
input1
  F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
```

```
 FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
 sm1.ecartd_syz(FF);
input2
 F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
 FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]],["noAutoHomogenize",1]]$
 sm1.ecartd_syz(FF);
```

1.2.31 sm1.gb_oxRingStructure

`sm1.gb_oxRingStructure()`
: It returns the oxRingStructure of the most recent gb computation.

1.2.32 sm1.gb_reduction

`sm1.gb_reduction(F,A)`
: It returns a reduced form of F in terms of A by using a normal form algorithm. h[1,1](D)-homogenization is used.

Example:

```
input
  F=[2*(h-x-y)*dx+h^2,2*(h-x-y)*dy+h^2]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]]]$
  sm1.gb_reduction((h-x-y)^2*dx*dy,FF);
```

1.2.33 sm1.gb_reduction_noh

`sm1.gb_reduction_noh(F,A)`
: It returns a reduced form of F in terms of A by using a normal form algorithm.

Example:

```
input
  F=[2*dx+1,2*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1]]]$
  sm1.gb_reduction_noh((1-x-y)^2*dx*dy,FF);
```

1.2.34 sm1.generalized_bfunction

`sm1.generalized_bfunction(I,V,VD,W)`
: It computes the generalized b-function (indicial equation) of I with respect to the weight W.

Description:

It no longer calls sm1's original function. Instead, it calls asir "generic_bfct".

Example:

```
sm1.generalized_bfunction([x^2*dx^2-1/2,dy^2],[x,y],[dx,dy],[-1,0,1,0]);
```

1.2.35 sm1.integration

`sm1.integration(I,V,R)`
: It computes the integration of I as a D-module to the set defined by R. V is the list of variables. When the optional variable *degree*=d is given, only the integrations from 0 to d are computed. Note that, in case of vector input, INTEGRATION VARIABLES MUST APPEAR FIRST in the list of variable V. We are using wbfRoots to get the roots of b-functions, so we can use only generic weight vector for now.

`sm1.integration(I,V,R | degree=key0)`
: This function allows optional variables *degree*

Algorithm:

T.Oaku and N.Takayama, math.AG/9805006, http://www.arxiv.org

Example:

```
sm1.integration([dt - (3*t^2-x), dx + t],[t,x],[t]);
  The output [[n0,F0],[n1,F1],...] means that H^0=D^n0/F0, H^(-1)=D^n1/F1, ...
  The free basis of the vector space D^n is denoted by e0, e1, ...
```

1.2.36 sm1.isSameIdeal_in_Dalg

`sm1.isSameIdeal_in_Dalg(I,J,V)`
: It compares two ideals I and J in D_alg (algebraic D with variables V, no homogenization).

Example:

```
Input1
  II=[(1-x)^2*dx+(1-x)]$ JJ = [(1-x)*dx+1]$ V=[x]$
  sm1.isSameIdeal_in_Dalg(II,JJ,V);
```

1.2.37 sm1.laplace

`sm1.laplace(L,V,VL)`
: It returns the Laplace transformation of L for VL. V is the list of space variables. The numbers in coefficients must not be rational with a non-1 denominator. cf. ptozp

Example:

```
L1=sm1.laplace(dt-(3*t^2-x),[x,t],[t,dt]);
L2=sm1.laplace(dx+t,[x,t],[t,dt]);
sm1.restriction([L1,L2],[t,x],[t] | degree=0);
```

1.2.38 sm1.rat_to_p

`sm1.rat_to_p(F)`
: It returns the denominator of F and the numerator of F. They are returned in a list. All elements of the denominator and numerator are polynomial objects

with integer coefficients. Note that dn and nm do not regard rational numbers as a factional object and this function is necessary to send data to sm1, which accept only integers and does not accept rational numbers.

Example:

```
sm1.rat_to_p(1/2*x+1);
  [x+2,2]
sm1.rat_to_p([1/2*x,1/3*x]);
  [[3*x,2*x],6]
```

1.2.39 sm1.restriction

`sm1.restriction(`*I*`,`*V*`,`*R*`)`

: It computes the restriction of I as a D-module to the set defined by R. V is the list of variables. When the optional variable *degree*=d is given, only the restrictions from 0 to d are computed. Note that, in case of vector input, RESTRICTION VARIABLES MUST APPEAR FIRST in the list of variable V. We are using wbfRoots to get the roots of b-functions, so we can use only generic weight vector for now.

`sm1.restriction(`*I*`,`*V*`,`*R* `|` *degree*`=key0)`

: This function allows optional variables *degree*

Algorithm:

T.Oaku and N.Takayama, math.AG/9805006, http://xxx.lanl.gov

Example:

```
sm1.restriction([dx^2-x,dy^2-1],[x,y],[y]);
  The output [[n0,F0],[n1,F1],...] means that H^0=D^n0/F0, H^(-1)=D^n1/F1, ...
  The free basis of the vector space D^n is denoted by e0, e1, ...
```

1.2.40 sm1.saturation

`sm1.saturation(`*T*`)`

: $T = [I,J,V]$. It returns saturation of I with respect to J^infty. V is a list of variables.

Example:

```
sm1.saturation([[x2^2,x2*x4, x2, x4^2], [x2,x4], [x2,x4]]);
```

1.2.41 sm1.ahg

`sm1.ahg(`*A*`)`

: It idential with sm1.gkz(A).

1.2.42 sm1.bfunction

`sm1.bfunction(`*F*`)`

: It computes the global b-function of F.

Description:

It no longer calls sm1's original bfunction. Instead, it calls asir "bfct".

Algorithm:

M.Noro, Mathematical Software, icms 2002, pp.147–157.

Example:

```
sm1.bfunction(x^2-y^3);
```

1.2.43 sm1.call_sm1

`sm1.call_sm1(F)`

: It executes F on the sm1 server. See also sm1.

1.2.44 sm1.ecart_homogenize01Ideal

`sm1.ecart_homogenize01Ideal(A)`

: It (0,1)-homogenizes the ideal $A[0]$. Note that it is not an elementwise homogenization.

Example:

```
input1
  F=[(1-x)*dx+1]$ FF=[F,"x,y"]$
  sm1.ecart_homogenize01Ideal(FF);
intput2
  F=sm1.appell1([1,2,3,4]);
  sm1.ecart_homogenize01Ideal(F);
```

1.2.45 sm1.ecartd_gb

`sm1.ecartd_gb(A)`

: It returns a standard basis of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. If the option rv="dp" (return_value="dp") is given, the answer is returned in distributed polynomials.

Note. Functions in the category ecart changes the global environment in the sm1 server. If you interrupted these functions, run sm1.ecartd_gb with a small input and terminate it normally. Then, the global environment is reset to the normal state. Reference. G. Granger, T. Oaku, N. Takayama, Tangent cone algorithm for homogeized differential operators, 2005.

Example:

```
input1
  F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  sm1.ecartd_gb(FF);
output1
  [[(-2*x-2*y+2)*dx+h,(-2*x-2*y+2)*dy+h],[(-2*x-2*y+2)*dx,(-2*x-2*y+2)*dy]]
```

```
input2
  F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]],["noAutoHomogenize",1]]$
  sm1.ecartd_gb(FF);
input3
  F=[[x^2,y+x],[x+y,y^3], [2*x^2+x*y,y+x+x*y^3]]$
  FF=[F,"x,y",[[dx,1,dy,1],[x, -1, y, -1,dx, 1, dy, 1]],
            ["degreeShift",[[0,1],[-3,1]]]]$
  sm1.ecartd_gb(FF);
```

1.2.46 sm1.ecartd_gb_oxRingStructure

`sm1.ecartd_gb_oxRingStructure()`
: It returns the oxRingStructure of the most recent ecartd_gb computation.

1.2.47 sm1.ecartd_isSameIdeal_h

`sm1.ecartd_isSameIdeal_h(`*F*`)`
: Here, F=[II,JJ,V]. It compares two ideals II and JJ in h[0,1](D)_alg.

Example:

```
input
  II=[(1-x)^2*dx+h*(1-x)]$ JJ = [(1-x)*dx+h]$
  V=[x]$
  sm1.ecartd_isSameIdeal_h([II,JJ,V]);
```

1.2.48 sm1.ecartd_reduction

`sm1.ecartd_reduction(`*F*`,`*A*`)`
: It returns a reduced form of F in terms of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. When the output is G, G[3] is F and G[0]-(G[1]*A-sum(k,G[2][k]*G[3][k]))=0 holds. F must be (0,1)-hohomogenized (see sm1.ecart_homogenize01Ideal). This function does not check if the given order is admissible for the ecart reduction. To do this check, use sm1.ecartd_gb.

Example:

```
input
  F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  G=sm1.ecartd_reduction(dx+dy,FF);
  G[0]-(G[1]*(dx+dy)+G[2][0]*F[0]+G[2][1]*F[1]);
```

1.2.49 sm1.ecartd_reduction_noh

`sm1.ecartd_reduction_noh(`*F*`,`*A*`)`
: It returns a reduced form of F in terms of A by using a tangent cone algorithm. h[0,1](D)-homogenization is NOT used. A[0] must not contain the variable h.

Example:

```
F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  sm1.ecartd_reduction_noh(dx+dy,FF);
```

1.2.50 sm1.ecartd_syz

`sm1.ecartd_syz(A)`
: It returns a syzygy of A by using a tangent cone algorithm. h[0,1](D)-homogenization is used. If the option rv="dp" (return_value="dp") is given, the answer is returned in distributed polynomials. The return value is in the format [s,[g,m,t]]. s is the generator of the syzygies, g is the Grobner basis, m is the translation matrix from the generators to g. t is the syzygy of g.

Example:

```
input1
  F=[2*(1-x-y)*dx+1,2*(1-x-y)*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1]]]$
  sm1.ecartd_syz(FF);
 input2
  F=[2*(1-x-y)*dx+h,2*(1-x-y)*dy+h]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]],["noAutoHomogenize",1]]$
  sm1.ecartd_syz(FF);
```

1.2.51 sm1.gb_oxRingStructure

`sm1.gb_oxRingStructure()`
: It returns the oxRingStructure of the most recent gb computation.

1.2.52 sm1.gb_reduction

`sm1.gb_reduction(F,A)`
: It returns a reduced form of F in terms of A by using a normal form algorithm. h[1,1](D)-homogenization is used.

Example:

```
input
  F=[2*(h-x-y)*dx+h^2,2*(h-x-y)*dy+h^2]$
  FF=[F,"x,y",[[dx,1,dy,1],[x,-1,y,-1,dx,1,dy,1]]]$
  sm1.gb_reduction((h-x-y)^2*dx*dy,FF);
```

1.2.53 sm1.gb_reduction_noh

`sm1.gb_reduction_noh(F,A)`
: It returns a reduced form of F in terms of A by using a normal form algorithm.

Example:

```
input
  F=[2*dx+1,2*dy+1]$
  FF=[F,"x,y",[[dx,1,dy,1]]]$
  sm1.gb_reduction_noh((1-x-y)^2*dx*dy,FF);
```

1.2.54 sm1.generalized_bfunction

`sm1.generalized_bfunction(`*I*`,`*V*`,`*VD*`,`*W*`)`
: It computes the generalized b-function (indicial equation) of I with respect to the weight W.

Description:

It no longer calls sm1's original function. Instead, it calls asir "generic_bfct".

Example:

```
sm1.generalized_bfunction([x^2*dx^2-1/2,dy^2],[x,y],[dx,dy],[-1,0,1,0]);
```

1.2.55 sm1.integration

`sm1.integration(`*I*`,`*V*`,`*R*`)`
: It computes the integration of I as a D-module to the set defined by R. V is the list of variables. When the optional variable *degree*=d is given, only the integrations from 0 to d are computed. Note that, in case of vector input, INTEGRATION VARIABLES MUST APPEAR FIRST in the list of variable V. We are using wbfRoots to get the roots of b-functions, so we can use only generic weight vector for now.

`sm1.integration(`*I*`,`*V*`,`*R* ` | ` *degree*`=key0)`
: This function allows optional variables *degree*

Algorithm:

T.Oaku and N.Takayama, math.AG/9805006, http://www.arxiv.org

Example:

```
sm1.integration([dt - (3*t^2-x), dx + t],[t,x],[t]);
  The output [[n0,F0],[n1,F1],...] means that H^0=D^n0/F0, H^(-1)=D^n1/F1, ...
  The free basis of the vector space D^n is denoted by e0, e1, ...
```

1.2.56 sm1.isSameIdeal_in_Dalg

`sm1.isSameIdeal_in_Dalg(`*I*`,`*J*`,`*V*`)`
: It compares two ideals I and J in D_alg (algebraic D with variables V, no homogenization).

Example:

```
Input1
  II=[(1-x)^2*dx+(1-x)]$ JJ = [(1-x)*dx+1]$ V=[x]$
  sm1.isSameIdeal_in_Dalg(II,JJ,V);
```

1.2.57 sm1.laplace

`sm1.laplace(`*L*`,`*V*`,`*VL*`)`
: It returns the Laplace transformation of L for *VL*. *V* is the list of space variables. The numbers in coefficients must not be rational with a non-1 denominator. cf. ptozp

Example:

```
L1=sm1.laplace(dt-(3*t^2-x),[x,t],[t,dt]);
L2=sm1.laplace(dx+t,[x,t],[t,dt]);
sm1.restriction([L1,L2],[t,x],[t] | degree=0);
```

1.2.58 sm1.rat_to_p

`sm1.rat_to_p(`*F*`)`
: It returns the denominator of F and the numerator of F. They are returned in a list. All elements of the denominator and numerator are polynomial objects with integer coefficients. Note that dn and nm do not regard rational numbers as a factional object and this function is necessary to send data to sm1, which accept only integers and does not accept rational numbers.

Example:

```
sm1.rat_to_p(1/2*x+1);
  [x+2,2]
sm1.rat_to_p([1/2*x,1/3*x]);
  [[3*x,2*x],6]
```

1.2.59 sm1.restriction

`sm1.restriction(`*I*`,`*V*`,`*R*`)`
: It computes the restriction of *I* as a D-module to the set defined by *R*. *V* is the list of variables. When the optional variable *degree*=d is given, only the restrictions from 0 to d are computed. Note that, in case of vector input, RESTRICTION VARIABLES MUST APPEAR FIRST in the list of variable *V*. We are using wbfRoots to get the roots of b-functions, so we can use only generic weight vector for now.

`sm1.restriction(`*I*`,`*V*`,`*R* ` | ` *degree*`=key0)`
: This function allows optional variables *degree*

Algorithm:

T.Oaku and N.Takayama, math.AG/9805006, http://xxx.lanl.gov

Example:

```
sm1.restriction([dx^2-x,dy^2-1],[x,y],[y]);
  The output [[n0,F0],[n1,F1],...] means that H^0=D^n0/F0, H^(-1)=D^n1/F1, ...
  The free basis of the vector space D^n is denoted by e0, e1, ...
```

1.2.60 sm1.saturation

`sm1.saturation(T)`

: $T = [I,J,V]$. It returns saturation of I with respect to J^infty. V is a list of variables.

Example:

```
sm1.saturation([[x2^2,x2*x4, x2, x4^2], [x2,x4], [x2,x4]]);
```

Index

(Index is nonexistent)

(Index is nonexistent)

Short Contents

Table of Contents