Sm1 OX Server

Edition : auto generated by oxgentexi on 28 April 2024

OpenXM.org

1 SM1

sm1 ox `ox_sm1_forAsir` . `sm1.rr` . `sm1.rr` `$(OpenXM_HOME)/lib/asir-contrib` . `sm1` . . `sm1` `OpenXM/doc/kan96xx` .

, sm1 server windows . cygwin , OpenXM/misc/packages/Windows sm1 windows .

, . .

$X := \mathbf{C} \setminus \{0,1\} = \mathbf{C} \setminus V(x(x-1))$. X , $x=0, x=1$ 1. , 1 2 . `sm1` 0 1 .

```
[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

1.1.1 ox_sm1_forAsir

`ox_sm1_forAsir`

:: `asir` `sm1` .

- `ox_sm1_forAsir` `asir` `sm1.start` `sm1` .

  ,

  `ox_sm1_forAsir` = `$(OpenXM_HOME)/lib/sm1/bin/ox_sm1` + `$(OpenXM_HOME)/lib/sm1/callsm1.sm1` (macro file)
  + `$(OpenXM_HOME)/lib/sm1/callsm1b.sm1` (macro file)
  , , current directory, `$(LOAD_SM1_PATH)`, `$(OpenXM_HOME)/lib/sm1`, `/usr/local/lib/sm1` .
- : `sm1` `$(OpenXM_HOME)/src/kxx/oxserver00.c`, `$(OpenXM_HOME)/src/kxx/sm1stackmachine.c`

1.2

1.2.1 sm1.start

`sm1.start()`

:: localhost `ox_sm1_forAsir` .

return

- localhost `ox_sm1_forAsir` . `ox_sm1_forAsir` .
- `Xm_noX = 1` `ox_sm1_forAsir` .
- `ord` . , `x` `dx` (`dx` $\partial/\partial x$) , `sm1` , `dx` `x` . , `cc` `sm1` .
- `a` `z` , `d` `o` , , `x0, ..., x20, y0, ..., y20, z0, ..., z20` , (cf. `Sm1_ord_list` in `sm1`).

- static Sm1_proc . sm1.get_Sm1_proc() .

```
[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*)
:: sm1 *s* .

return

p

s

- *p* sm1 *s* . (, 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*)
:: *f* *p* .

return

p

f

- type(*f*) 2 () , *f* (type == 7) , ox_push_cmo .
- type(*f*) 0 (zero) , , 32 bit . ox_push_cmo(P,0) CMO_NULL , , 32 bit .
- sm1 , 32 bit bignum . type(*f*) 1 (), 32 bit integer . ox_push_cmo(*p*,1234) bignum 1234 sm1 .

- ox_push_cmo .

```
[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*,needBack=*n*,ring_var=*r*)
:: *v* *f* .

sm1.gb_d([*f*,*v*,*w*]|proc=*p*)
:: *v* *f* . .

return

p, *q*, *r*

f, *v*, *w*

- *v* *f* .
- Weight *w* . , graded reverse lexicographic order .
- sm1.gb *f* (*w*) (*w*) .
- sm1.gb_d . . [, , ,] .
- Term order , (SST Section 1.2). h .
- *q* , 3 , . , . *r* , dehomogenize (h 1).
- Reduced in_w , sm1.auto_reduce(1) .
- needBack 1 , [groebner basis, initial, gb,1,all, [groebner basis, backward transformation]] . (sm1 getAttribute)
- ring_var ring_var_for_asir . sm1 ring . reduction .
- difftop=1 , . option . difftop differential term order over position . [gb, init w.r.t w, init].

```
[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]]
```

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

```
[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]]
```

$m = x^a y^b \partial_x^c \partial_y^d$ $m' = x^{a'} y^{b'} \partial_x^{c'} \partial_y^{d'}$ weight vector (dx,dy,x,y) = (50,2,1,0) (m $50c+2d+a > 50c'+2d'+a'$ m') reverse lexicographic order ($50c+2d+a = 50c'+2d'+a'$ reverse lexicographic order).

```
[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>>]]]

[1834] sm1.gb([[dx^2-x,dx],[x]] | needBack=1);
[[dx,dx^2-x,1],[dx,dx^2,1],gb,1,all,[[dx,dx^2-x,1],[[0,1],[1,0],[-dx,dx^2-x]]]]

// computation in R^2
[1834] F=[[[dx^2-dy,0],[x*dx+2*y*dy+1/2,0]],[x,y],[[dx,1,dy,1]]];
[1835] sm1.gb([F,[x,y],[[dx,1,dy,1]]] | difftop=1);
```

sm1.gb sm1. . sm1 .

```
if ((P_sm1=sm1.get_Sm1_proc()) < 0) P_sm1=sm1.start();
//sm1.sm1(P_sm1," 13 /gb.characteristic set ");
sm1.sm1(P_sm1," [(TraceLift) 13] system_variable ");
sm1.sm1(P_sm1," [(StopDegree) 27] /gb.options set ");
```

`sm1.auto_reduce`, `sm1.reduction`, `sm1.rat_to_p`

1.2.5 sm1.deRham

`sm1.deRham([f,v]|proc=p)`
:: C^n - (the zero set of *f*=0) .

return

p

f

v

- X = C^n \ V(*f*) . , [dim H^0(X,C), dim H^1(X,C), dim H^2(X,C), ..., dim H^n(X,C)] .
- *v* . n = `length(v)` .
- `sm1.deRham` . `sm1.deRham(0,[x*y*z*(x+y+z-1)*(x-y),[x,y,z]])` .
- b-, `ox_asir` `ox_sm1_forAsir` . `sm1(0,"[(parse) (oxasir.sm1) pushfile] extension");` , `ox_asir` . `ox_asir_forAsir` .
- `sm1.deRham` `ox_reset(sm1.get_Sm1_proc());` , sm1 , `ox_shutdown(sm1.get_Sm1_proc());` , `ox_sm1_forAsir` shutdown .

```
[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)`
:: *f* .

`hilbert_polynomial(f,v)`
:: *f* .

return

p

f, *v*

- *f* *v* h(k) .
- h(k) = dim_Q F_k/I \cap F_k F_k k . I *f* .
- sm1.hilbert : *f* . , *f* , initial monomial . , *f* . , `sm1` `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 . *v* . , s *v*[0] , *f* `rest`(*v*) .

return

p

f

v

- , *f*^s . *v* . , s *v*[0] , *f* `rest`(*v*) .

```
[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 .

return

p

f, *g*, *v*, *w*

- *f* *g* D- 0 .

- *v* . *w* weight . *w*[i] *v*[i] weight .
- `sm1.wTensor0` `ox_sm1` `wRestriction0` . `wRestriction0` , generic weight *w* . Weight *w* generic .
- F G *f* *g* . , 0 FG .
- *f*, *g* D , , D^r .

```
[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)
sm1.reduction([f,g,v]|proc=p)
sm1.reduction([f,g]|proc=p,ring_var=r)
sm1.reduction_verbose([f,g,v,w]|proc=p)
```
::

return

f

g, *v*, *w*

p (ox_sm1)

- *f* homogenized , *g* (reduce) ; , , *f* . *v* . *w* , . `sm1.reduction_noH` , Weyl algebra .
- : [r,c0,[c1,...,cm],g] *g*=[g1, ..., gm] , c0 f + c1 g1 + ... + cm gm = r . r/c0 normal form .
- , reducible .
- sm1.reduction_d(P,F,G) sm1.reduction_noH_d(P,F,G) , .
- mod_reduction . ox_sm1 ring_var ring . auto_reduce(1) . gb .
- reduction_verbose [r,c0,[c1,...,cm],[g1,...gm],init,order] init order r initial.

```
[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]]

[1837] XM_debug=0$ S=sm1.syz([ [x^2-1,x^3-1,x^4-1],[x]])$
[1838] sm1.auto_reduce(1);
1
[1839] S0=sm1.gb([S[0],[x]]);
[[[-x^2-x-1,x+1,0],[x^2+1,0,-1]],[[0,x,0],[0,0,-1]]]
[1840] sm1.reduction([ [-x^4-x^3-x^2-x,x^3+x^2+x+1,-1], S0[0]]);
[[0,0,0],-1,[[x^2+1,0,0],[1,0,0]],[[-x^2-x-1,x+1,0],[x^2+1,0,-1]]]
```

```
XM_debug=0$
sm1.auto_reduce(1)$
F=[x*y-1,x^2+y^2-4]$
```

```
Weight_vec=[[x,10,y,1]]$
printf("\n\nsyz----\n")$
S=sm1.syz([F,[x,y],Weight_vec]);  // When Weight_vec is given, the TOP order is used.
// If the Weight_vec is not given, the POT order (e.g., (1,0,0)<(0,1,0)<(0,0,1)) with
Sgb=sm1.gb([S[0],[x,y],Weight_vec]);
R0=[x+y,x^2*y+x];
P=R0[0]*F[0]+R0[1]*F[1];
R=sm1.reduction_verbose([R0,Sgb[0],[x,y],Weight_vec]);

printf("\nMinimal representation=%a\n",R[0])$
printf("The initial of minimal rep=%a\n",R[4])$
printf("Order=%a\n",R[5][1][1])$
```

sm1.start, d_true_nf

1.2.10 sm1.xml_tree_to_prefix_string

sm1.xml_tree_to_prefix_string(*s*|proc=*p*)
:: XML OpenMath *s* .

return String

p Number

s String

- XML OpenMath *s* .
- om_* .
- om_xml_to_cmo(OpenMath Tree Expression) CMO_TREE . asir CMO .
- java . (, /usr/local/jdk1.1.8/bin .)

```
[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* *f* syzygy .

return

p

f, v, w

- : [*s*,[*g*, *m*, *t*]]. *s* *f* *v* syzygy . *g* *f* weight vector *w* . *m* *f* *g* . *t* *g* syzygy . , : $g = m\,f$, $s\,f = 0$.
- Weight *w* . , graded reverse lexicographic order .
- Term order , (SST Section 1.2). `h` .

```
[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 *f* *g* *v* .

return

p

f, g

v

- sm1 *f* *g* *v* .
- `sm1.mul_h` homogenized Weyl .
- BUG: `sm1.mul(p0*dp0,1,[p0])` `dp0*p0+1` . d, .

```
[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 .

return

p

f

v,x,d,s

- *p* `sm1` , *f* distraction *v* .
- *x* , *d* , , distract x , d . Distraction , *s* .
- Distraction x*dx x . Saito, Sturmfels, Takayama : Grobner Deformations of Hypergeometric Differential Equations page 68 .

```
[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)`
:: *A* *B* GKZ (A-hypergeometric system) .

return

p

A, B

- *A* *B* GKZ (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)`
:: *A*, weight *W* *B* modified GKZ (A-hypergeometric system) .

return

p

A, W, B

- *A*, weight vector *W* *B* modified GKZ (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 F_D .

return

p

a

- Appell F_1 n Lauricella F_D(a,b1,b2,...,bn,c;x1,...,xn) . , a =(a,c,b1,...,bn). .
- sm1 appell1 , .

```
[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 F_C .

return

p

a

- Appell F_4 n Lauricella F_C(a,b,c1,c2,...,cn;x1,...,xn) . , *a* =(a,b,c1,...,cn). .
- sm1 appell1 , .

```
[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)`
:: *a* holonomic rank .

return

p

a

- *a* , generic point . , holonomic rank .
- *a* .
- *a* regular holonomic `sm1.rrank` holonomic rank . `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* .

p

s

- *s* 1 , , reduced .
- *s* 0 , reduced . .

1.2.20 sm1.slope

`sm1.slope(`*ii*`,`*v*`,`*f_filtration*`,`*v_filtration*`|proc=`*p*`)`

:: *ii* slope .

return

p

ii ()

v ()

f_filtration (weight vector)

v_filtration
(weight vector)

- `sm1.slope` *ii* V filtration *v_filtration* (geomeric) slope .
- *v* .
- , . 1 slope, 2 , weight vector microcharacteristic variety bihomogeneous .

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" . Slope s' , , , Slope -s' . pF+qV microgap, -s'=q/p . s=1-1/s' slope . O(s) . s 1<= s . r=s-1=-1/s' kappa=1/r=-s' . Borel and Laplace 1/Gamma(1+m*r) factor, exp(-tau^kappa) .

```
[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