Sm1 OX Server Manual

OpenXM.org

1 SM1 Functions

This chapter describes interface functions for sm1 ox server `ox_sm1_forAsir`. These interface functions are defined in the file 'sm1.rr'. The file 'sm1.rr' is at '$(OpenXM_HOME)/lib/asir/contrib-packages'. The system sm1 is a system to compute in the ring of differential operators. Many constructions of invariants in the computational algebraic geometry reduce to constructions in the ring of differential operators. Documents on sm1 are in the directory `OpenXM/doc/kan96xx`.

All the coefficients of input polynomials should be integers for most functions in this section. Other functions accept rational numbers as inputs and it will be explicitly noted in each explanation of these functions.

Let us evaluate the dimensions of the de Rham cohomology groups of $X := \mathbf{C} \setminus \{0, 1\} = \mathbf{C} \setminus V(x(x-1))$. The space X is a two punctured plane, so two loops that encircles the points $x = 0$ and $x = 1$ respectively spans the first homology group. Hence, the dimension of the first de Rham cohomology group is 2. sm1 answers the dimensions of the 0th and the first cohomology groups.

```
[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` Server

1.1.1 `ox_sm1_forAsir`

`ox_sm1_forAsir`
     :: sm1 server for `asir`.

- `ox_sm1_forAsir` is the sm1 server started from asir by the command `sm1.start`. In the standard setting,
  `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)
  The macro files 'callsm1.sm1' and 'callsm1b.sm1' are searched from current directory, $(LOAD_SM1_PATH), $(OpenXM_HOME)/lib/sm1, /usr/local/lib/sm1 in this order.

- Note for programmers: See the files '$(OpenXM_HOME)/src/kxx/oxserver00.c', '$(OpenXM_HOME)/src/kxx/sm1stackmachine.c' to build your own server by reading sm1 macros.

1.2 Functions

1.2.1 sm1.start

`sm1.start()`

:: Start `ox_sm1_forAsir` on the localhost.

return   Integer

- Start `ox_sm1_forAsir` on the localhost. It returns the descriptor of `ox_sm1_forAsir`.
- Set `Xm_noX = 1` to start `ox_sm1_forAsir` without a debug window.
- You might have to set suitable orders of variable by the command `ord`. For example, when you are working in the ring of differential operators on the variable `x` and `dx` (`dx` stands for $\partial/\partial x$), `sm1` server assumes that the variable `dx` is collected to the right and the variable `x` is collected to the left in the printed expression. In the example below, you must not use the variable `cc` for computation in `sm1`.
- The variables from `a` to `z` except `d` and `o` and `x0`, ..., `x20`, `y0`, ..., `y20`, `z0`, ..., `z20` can be used as variables for ring of differential operators in default. (cf. `Sm1_ord_list` in `sm1`).
- The descriptor is stored in `static Sm1_proc`. The descriptor can be obtained by the function `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
  ```

`Reference`

ox_launch, sm1.push_int0, sm1.push_poly0, ord

1.2.2 sm1.sm1

`sm1.sm1(p,s)`

:: ask the `sm1` server to execute the command string *s*.

return   Void

p   Number

s   String

- It asks the sm1 server of the descriptor number p to execute the command string s. (In the next example, the descriptor number is 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]
```

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

1.2.3 `sm1.push_int0`

`sm1.push_int0(p,f)`
        :: push the object f to the server with the descriptor number p.

return   Void

p   Number

f   Object

- When `type(f)` is 2 (recursive polynomial), f is converted to a string (type == 7) and is sent to the server by `ox_push_cmo`.
- When `type(f)` is 0 (zero), it is translated to the 32 bit integer zero on the server. Note that `ox_push_cmo(p,0)` sends `CMO_NULL` to the server. In other words, the server does not get the 32 bit integer 0 nor the bignum 0.
- `sm1` integers are classfied into the 32 bit integer and the bignum. When `type(f)` is 1 (number), it is translated to the 32 bit integer on the server. Note that `ox_push_cmo(p,1234)` send the bignum 1234 to the `sm1` server.
- In other cases, `ox_push_cmo` is called without data conversion.

```
[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)`
:: computes the Grobner basis of f in the ring of differential operators with the variable v.

`sm1.gb_d([f,v,w]|proc=p)`
:: computes the Grobner basis of f in the ring of differential operators with the variable v. The result will be returned as a list of distributed polynomials.

return   List

p, q, r   Number

f, v, w   List

- It returns the Grobner basis of the set of polynomials f in the ring of deferential operators with the variables v.
- The weight vectors are given by w, which can be omitted. If w is not given, the graded reverse lexicographic order will be used to compute Grobner basis.
- The return value of `sm1.gb` is the list of the Grobner basis of f and the initial terms (when w is not given) or initial ideal (when w is given).
- `sm1.gb_d` returns the results by a list of distributed polynomials. Monomials in each distributed polynomial are ordered in the given order. The return value consists of [variable names, order matrix, grobner basis in districted polynomials, initial monomials or initial polynomials].
- When a non-term order is given, the Grobner basis is computed in the homogenized Weyl algebra (See Section 1.2 of the book of SST). The homogenization variable h is automatically added.
- When the optional variable q is set, `sm1.gb` returns, as the third return value, a list of the Grobner basis and the initial ideal with sums of monomials sorted by the given order. Each polynomial is expressed as a string temporally for now. When the optional variable r is set to one, the polynomials are dehomogenized (,i.e., h is set to 1).

```
[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]]
```

In the example above, the set $\{x\partial_x + y\partial_y - 1, y^2\partial_y^2 + 2\}$ is the Gröbner basis of the input with respect to the graded reverse lexicographic order such that $1 \le \partial_y \le \partial_x \le y \le x \le \cdots$. The set $\{x\partial_x, y^2\partial_y\}$ is the leading monomials (the initial monominals) of the Gröbner basis.

```
[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]]
```

In the example above, two monomials $m = x^a y^b \partial_x^c \partial_y^d$ and $m' = x^{a'} y^{b'} \partial_x^{c'} \partial_y^{d'}$ are firstly compared by the weight vector `(dx,dy,x,y) = (50,2,1,0)` (i.e., m is larger than m' if $50c + 2d + a > 50c' + 2d' + a'$) and when the comparison is tie, then these are compared by the reverse lexicographic order (i.e., if $50c + 2d + a = 50c' + 2d' + a'$, then use the reverse lexicogrpahic 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>>]]]
```

Reference
        sm1.reduction, sm1.rat_to_p

1.2.5 sm1.deRham

sm1.deRham([f,v]|proc=p)
        :: ask the server to evaluate the dimensions of the de Rham cohomology groups
        of C^n - (the zero set of f=0).

return   List

p Number

f String or polynomial

v List

- It returns the dimensions of the de Rham cohomology groups of X = C^n \ V(f).
  In other words, it returns [dim H^0(X,C), dim H^1(X,C), dim H^2(X,C), ..., dim
  H^n(X,C)].

- v is a list of variables. n = length(v).

- sm1.deRham requires huge computer resources. For example, sm1.deRham(0,[x*y*z*(x+y+z-
  1)*(x-y),[x,y,z]]) is already very hard.

- To efficiently analyze the roots of b-function, `ox_asir` should be used from `ox_sm1_forAsir`. It is recommended to load the communication module for `ox_asir` by the command
  `sm1(0,"[(parse) (oxasir.sm1) pushfile] extension");` This command is automatically executed when `ox_sm1_forAsir` is started.
- If you make an interruption to the function `sm1.deRham` by `ox_reset(sm1.get_Sm1_proc());`, the server might get out of the standard mode. So, it is strongly recommended to execute the command `ox_shutdown(sm1.get_Sm1_proc());` to interrupt and restart the server.

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

`Reference`
        `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)`
        :: ask the server to compute the Hilbert polynomial for the set of polynomials *f*.

`hilbert_polynomial(f,v)`
        :: ask the server to compute the Hilbert polynomial for the set of polynomials *f*.

return   Polynomial

p   Number

f, v   List

- It returns the Hilbert polynomial h(k) of the set of polynomials *f* with respect to the set of variables *v*.
- h(k) = dim_Q F_k/I \cap F_k where F_k the set of polynomials of which degree is less than or equal to k and I is the ideal generated by the set of polynomials *f*.
- Note for sm1.hilbert: For an efficient computation, it is preferable that the set of polynomials *f* is a set of monomials. In fact, this function firstly compute a Grobner basis of *f*, and then compute the Hilbert polynomial of the initial monomials of the basis. If the input *f* is already a Grobner basis, a Grobner basis is recomputed in this function, which is a waste of time and Grobner basis computation in the ring of polynomials in `sm1` is slower than in `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
```

Reference
        sm1.start, sm1.gb, longname

1.2.7 sm1.genericAnn

sm1.genericAnn([*f*,*v*]|proc=*p*)
        :: It computes the annihilating ideal for f^s. *v* is the list of variables. Here, s
        is *v*[0] and *f* is a polynomial in the variables rest(*v*).

return	List
p	Number
f	Polynomial
v	List

  • This function computes the annihilating ideal for f^s. *v* is the list of variables. Here,
    s is *v*[0] and *f* is a polynomial in the variables 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]
```

Reference
        sm1.start

1.2.8 sm1.wTensor0

sm1.wTensor0([*f*,*g*,*v*,*w*]|proc=*p*)
        :: It computes the D-module theoretic 0-th tensor product of *f* and *g*.

return   List

p   Number

f, g, v, w   List

- It returns the D-module theoretic 0-th tensor product of *f* and *g*.

- *v* is a list of variables. *w* is a list of weights. The integer *w*[i] is the weight of the variable *v*[i].

- `sm1.wTensor0` calls `wRestriction0` of `ox_sm1`, which requires a generic weight vector *w* to compute the restriction. If *w* is not generic, the computation fails.

- Let F and G be solutions of *f* and *g* respectively. Intuitively speaking, the 0-th tensor product is a system of differential equations which annihilates the function FG.

- The answer is a submodule of a free module D^r in general even if the inputs *f* and *g* are left ideals of D.

```
[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   List

f   Polynomial

g, v, w   List

p   Number (the process number of ox_sm1)

- It reduces *f* by the set of polynomial *g* in the homogenized Weyl algebra; it applies the division algorithm to *f*. The set of variables is *v* and *w* is weight vectors to determine the order, which can be ommited. `sm1.reduction_noH` is for the Weyl algebra.

- The return value is of the form [r,c0,[c1,...,cm],[g1,...gm]] where *g*=[g1, ..., gm] and c0 f + c1 g1 + ... + cm gm = r. r/c0 is the normal form.

- The function reduction reduces reducible terms that appear in lower order terms.

- The functions sm1.reduction_d(P,F,G) and sm1.reduction_noH_d(P,F,G) are for distributed polynomials.

```
[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]]
```

Reference
          sm1.start, d_true_nf

1.2.10 `sm1.xml_tree_to_prefix_string`

`sm1.xml_tree_to_prefix_string(s|proc=p)`
          :: Translate OpenMath Tree Expression s in XML to a prefix notation.

return   String

p   Number

s   String

- It translate OpenMath Tree Expression s in XML to a prefix notation.
- This function should be moved to om_* in a future.
- `om_xml_to_cmo(OpenMath Tree Expression)` returns CMO_TREE. asir has not yet understood this CMO.
- `java` execution environment is required. (For example, `/usr/local/jdk1.1.8/bin` should be in the command search path.)

```
[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))
```

Reference
          `om_*`, `OpenXM/src/OpenMath`, `eval_str`

1.2.11 `sm1.syz`

`sm1.syz([f,v,w]|proc=p)`
          :: computes the syzygy of f in the ring of differential operators with the variable v.

return   List

p   Number

f, v, w   List

- The return values is of the form $[s,[g, m, t]]$. Here s is the syzygy of f in the ring of differential operators with the variable v. g is a Groebner basis of f with the weight vector w, and m is a matrix that translates the input matrix f to the Gr\"obner basis g. t is the syzygy of the Gr\"obner basis g. In summary, $g = m\,f$ and $s\,f = 0$ hold as matrices.

- The weight vectors are given by w, which can be omitted. If w is not given, the graded reverse lexicographic order will be used to compute Grobner basis.

- When a non-term order is given, the Grobner basis is computed in the homogenized Weyl algebra (See Section 1.2 of the book of SST). The homogenization variable h is automatically added.

```
[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)`
:: ask the sm1 server to multiply f and g in the ring of differential operators over v.

return   Polynomial or List

p   Number

f, g   Polynomial or List

v   List

- Ask the sm1 server to multiply f and g in the ring of differential operators over v.
- `sm1.mul_h` is for homogenized Weyl algebra.
- BUG: `sm1.mul(p0*dp0,1,[p0])` returns `dp0*p0+1`. A variable order such that d-variables come after non-d-variables is necessary for the correct computation.

```
[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)`
:: ask the `sm1` server to compute the distraction of f.

return   List

p	Number
f	Polynomial
v,x,d,s	List

- It asks the `sm1` server of the descriptor number p to compute the distraction of f in the ring of differential operators with variables v.
- x is a list of x-variables and d is that of d-variables to be distracted. s is a list of variables to express the distracted f.
- Distraction is roughly speaking to replace x*dx by a single variable x. See Saito, Sturmfels, Takayama : Grobner Deformations of Hypergeometric Differential Equations at page 68 for details.

```
[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
```

`Reference`
`        distraction2(sm1),`

1.2.14 sm1.gkz

`sm1.gkz([A,B]|proc=p)`
          :: Returns the GKZ system (A-hypergeometric system) associated to the matrix A with the parameter vector B.

$return$	List
p	Number
A, B	List

- Returns the GKZ hypergeometric system (A-hypergeometric system) associated to the matrix

```
[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.appell1

`sm1.appell1(a|proc=p)`
          :: Returns the Appell hypergeometric system F_1 or F_D.

return   List

p   Number

a   List

- Returns the hypergeometric system for the Lauricella function F_D(a,b1,b2,...,bn,c;x1,...,xn) where a =(a,c,b1,...,bn). When n=2, the Lauricella function is called the Appell function F_1. The parameters a, c, b1, ..., bn may be rational numbers.

- It does not call sm1 function appell1. As a concequence, when parameters are rational or symbolic, this function also works as well as integral parameters.

```
[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
```

1.2.16 `sm1.appell4`

`sm1.appell4(a|proc=p)`
        :: Returns the Appell hypergeometric system F_4 or F_C.

return   List

p   Number

a   List

- Returns the hypergeometric system for the Lauricella function F_4(a,b,c1,c2,...,cn;x1,...,xn) where a =(a,b,c1,...,cn). When n=2, the Lauricella function is called the Appell function F_4. The parameters a, b, c1, ..., cn may be rational numbers.

-

- It does not call sm1 function appell4. As a concequence, when parameters are rational or symbolic, this function also works as well as integral parameters.

```
[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
```

1.2.17 `sm1.rank`

`sm1.rank(a|proc=p)`
            :: Returns the holonomic rank of the system of differential equations a.

return	Number
p	Number
a	List

- It evaluates the dimension of the space of holomorphic solutions at a generic point of the system of differential equations a. The dimension is called the holonomic rank.
- a is a list consisting of a list of differential equations and a list of variables.
- `sm1.rrank` returns the holonomic rank when a is regular holonomic. It is generally faster than `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
```

1.2.18 `sm1.auto_reduce`

`sm1.auto_reduce(s|proc=p)`
            :: Set the flag "AutoReduce" to s.

return	Number
p	Number
s	Number

- If *s* is 1, then all Grobner bases to be computed will be the reduced Grobner bases.
- If *s* is 0, then Grobner bases are not necessarily the reduced Grobner bases. This is the default.

1.2.19 `sm1.slope`

`sm1.slope(ii,v,f_filtration,v_filtration|proc=p)`
:: Returns the slopes of differential equations *ii*.

return	List
p	Number
ii	List (equations)
v	List (variables)
f_filtration	List (weight vector)
v_filtration	
	List (weight vector)

- `sm1.slope` returns the (geometric) slopes of the system of differential equations *ii* along the hyperplane specified by the V filtration *v_filtration*.
- *v* is a list of variables.
- The return value is a list of lists. The first entry of each list is the slope and the second entry is the weight vector for which the microcharacteristic variety is not bihomogeneous.

Algorithm: see "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" Note that the signs of the slopes are negative, but the absolute values of the slopes are returned.

```
    [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]);
```

```
Reference
        sm.gb
```

1.2.20 `sm1.ahg`

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

1.2.21 `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);
```

1.2.22 `sm1.call_sm1`

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

1.2.23 `sm1.ecart_homogenize01Ideal`

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

   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);
```

1.2.24 `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.

   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);
```

1.2.25 `sm1.ecartd_gb_oxRingStructure`

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

1.2.26 `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]);
```

1.2.27 `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.

   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]]]$
  sm1.ecartd_reduction(dx+dy,FF);
```

1.2.28 `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);
```

1.2.29 `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);
```

1.2.30 `sm1.gb_oxRingStructure`

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

1.2.31 `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);
```

1.2.32 `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);
```

1.2.33 `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]);
```

1.2.34 `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);
```

1.2.35 `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.langl.gov

Example:
```
sm1.restriction([dx^2-x,dy^2-1],[x,y],[y]);
```

1.2.36 `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]]);
```

1.2.37 `sm1.ahg`

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

1.2.38 `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);
```

1.2.39 `sm1.call_sm1`

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

1.2.40 `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);
```

1.2.41 `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.

   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);
```

1.2.42 `sm1.ecartd_gb_oxRingStructure`

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

1.2.43 `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]);
```

1.2.44 `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.

   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]]]$
sm1.ecartd_reduction(dx+dy,FF);
```

1.2.45 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);
```

1.2.46 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);
```

1.2.47 sm1.gb_oxRingStructure

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

1.2.48 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);
```

1.2.49 `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);
```

1.2.50 `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]);
```

1.2.51 `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);
```

1.2.52 `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.langl.gov

Example:

```
sm1.restriction([dx^2-x,dy^2-1],[x,y],[y]);
```

1.2.53 `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