Sm1 OX Server マニュアル

Edition : auto generated by oxgentexi on 30 January 2008

OpenXM.org

1 SM1 函数

この節では sm1 の ox サーバ ox_sm1_forAsir とのインタフェース関数を解説する. これらの関数はファイル 'sm1.rr' で定義されている. 'sm1.rr' は '$(OpenXM_HOME)/lib/asir-contrib' にある. システム sm1 は微分作用素環で計算するためのシステムである. 計算代数幾何のいろいろな不変量の計算が微分作用素の計算に帰着する. sm1 についての文書は OpenXM/doc/kan96xx にある.

とこに断りがないかぎりこの節のすべての関数は, 有理数係数の式を入力としてうけつけない. すべての多項式の係数は整数でないといけない.

空間 $X := \mathbf{C} \setminus \{0, 1\} = \mathbf{C} \setminus V(x(x-1))$ のドラームコホモロジ群達の次元を計算してみよう. X は平面に二つの穴をあけた空間であるので, 点 $x = 0$, $x = 1$ のまわりをまわる二つのループが 1 次元のホモロジー群の空間をはる. したがって, 1 次元ドラームコホモロジ群の次元は 2 である. sm1 は 0 次元のコホモロジ群の次元および 1 次元のコホモロジ群の次元を答える.

```
[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 サーバ

1.1.1 ox_sm1_forAsir

ox_sm1_forAsir
:: asir のための sm1 サーバ.

- サーバ ox_sm1_forAsir は asir よりコマンド sm1.start で起動される sm1 サーバである.

  標準的設定では,
  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)
  であり, これらのマクロファイルは, 一般には current directory, $(LOAD_SM1_PATH), $(OpenXM_HOME)/lib/sm1, /usr/local/lib/sm1 の順番でさがされる.
- プログラマーのためのノート: sm1 マクロを読み込んで自分独自のサーバを作るには次のファイルも見よ '$(OpenXM_HOME)/src/kxx/oxserver00.c', '$(OpenXM_HOME)/src/kxx/sm1stackmachine.c'

1.2 函数一覧

1.2.1 sm1.start

sm1.start()
:: localhost で ox_sm1_forAsir をスタートする.

return 整数

- localhost で ox_sm1_forAsir をスタートする. サーバ ox_sm1_forAsir の識別番号を戻す.
- Xm_noX = 1 とおくとサーバ ox_sm1_forAsir をデバッグ用のウィンドウなしに起動できる.
- コマンド ord を用いて変数順序を正しく設定しておく必要がある. たとえば, 変数 x と dx 上の微分作用素環 (dx は $\partial/\partial x$ に対応) で計算しているとき, sm1 サーバは式を印刷したとき, 変数 dx は右側に集めれ変数 x は左側にあつめられていると仮定している. 次の例では, 変数 cc を sm1 での計算のために用いてはいけない.
- a より z のなかで, d と o を除いたもの, それから, x0, ..., x20, y0, ..., y20, z0, ..., z20 は, デフォールトで微分作用素環の変数として使える (cf. Sm1_ord_list in sm1).
- 識別番号は static Sm1_proc に格納される. この識別番号は関数 sm1.get_Sm1_proc() でとりだすことができる.

```
[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
```

参照 ox_launch, sm1.push_int0, sm1.push_poly0, ord

1.2.2 sm1.sm1

sm1.sm1(*p*,*s*)
:: サーバ sm1 にコマンド列 s を実行してくれるようにたのむ.

return なし

p 数

s 文字列

- 識別番号 p の sm1 サーバにコマンド列 s を実行してくれるように頼む. (次の例では, 識別番号 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]
```

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

1.2.3 sm1.push_int0

sm1.push_int0(*p*,*f*)
:: オブジェクト f を識別子 p のサーバへ送る.

return　　なし

p　　数

f　　オブジェクト

- type(*f*) が 2 (再帰多項式) のとき, f は文字列 (type == 7) に変換されて, ox_push_cmo を用いてサーバへ送られる.
- type(*f*) が 0 (zero) のときは, サーバ上では, 32 bit 整数と解釈される. なお ox_push_cmo(P,0) はサーバに対して CMO_NULL をおくるので, サーバ側では, 32 bit 整数を受け取るわけではない.
- sm1 の整数は, 32 bit 整数と bignum にわけることができる. type(*f*) が 1 (数) のとき, この関数は 32 bit integer をサーバにおくる. ox_push_cmo(*p*,1234) は bignum の 1234 を sm1 サーバにおくることに注意しよう.
- その他の場合には ox_push_cmo をデータ型の変換なしに呼び出す.

```
[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 上の微分作用素環において f のグレブナ基底を計算する.

`sm1.gb_d([f,v,w]|proc=p)`
:: v 上の微分作用素環において f のグレブナ基底を計算する. 結果を分散多項式のリストで戻す.

return リスト

p, q, r 数

f, v, w リスト

- v 上の微分作用素環において f のグレブナ基底を計算する.
- Weight ベクトル w は省略してよい. 省略した場合, graded reverse lexicographic order をつかってブレブナ基底を計算する.
- sm1.gb の戻り値は f のグレブナ基底およびイニシャルモノミアル (w がないとき) または イニシァル多項式 (w が与えらたとき) のリストである.
- sm1.gb_d は結果を分散多項式のリストで戻す. 多項式の中に現れるモノミアルはグレブナ基底を計算するときに与えらた順序でソートされている. 戻り値は [変数名のリスト, 順序をきめる行列, グレブナ基底, イニシャルモノミアルまたはイニシァル多項式] である.
- Term order でない順序が与えられた場合は, 同次化ワイル代数でグレブナ基底が計算される (SST の本の Section 1.2 を見よ). 同次化変数 h が結果に加わる.
- オプショナル変数 q がセットされているときは, 3 番目の戻り値として, グレブナ基底およびイニシァルのリストが与えられた順序でソートされたモノミアルの和として戻される. いまのところこの多項式は, 文字列で表現される. オプショナル変数 r がセットされているときは, 戻り多項式は dehomogenize される (すなわち h に 1 が代入される).

```
[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]]
```

上の例において, 集合 $\{x\partial_x + y\partial_y - 1, y^2\partial_y^2 + 2\}$ は $1 \leq \partial_y \leq \partial_x \leq y \leq x \leq \cdots$ であるような graded reverse lexicographic order に関するグレブナ基底である. 集合 $\{x\partial_x, y^2\partial_y\}$ はグレブナ基底の各元に対する leading monomial (initial monomial) である.

```
[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]]
```

上の例において二つのモノミアル $m = x^a y^b \partial_x^c \partial_y^d$ および $m' = x^{a'} y^{b'} \partial_x^{c'} \partial_y^{d'}$ は最初に weight vector `(dx,dy,x,y) = (50,2,1,0)` を用いて比較される (つまり m は $50c+2d+a > 50c'+2d'+a'$ のとき m' より大きい) 次にこの比較で勝負がつかないときは reverse lexicographic order で比較される (つまり $50c+2d+a = 50c'+2d'+a'$ のとき reverse lexicographic order で比較される).

```
[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>>]]]
```

参照 sm1.reduction, sm1.rat_to_p

1.2.5 sm1.deRham

sm1.deRham([*f*,*v*]|proc=*p*)
:: 空間 C^n - (the zero set of f=0) のドラームコホモロジ群の次元を計算してくれるようにサーバに頼む.

return リスト

p 数

f 文字列 または 多項式

v リスト

- この函数は空間 X = C^n \ V(f) のドラームコホモロジ群の次元を計算する. すなわち, [dim H^0(X,C), dim H^1(X,C), dim H^2(X,C), ..., dim H^n(X,C)] を戻す.
- v は変数のリスト. n = length(v) である.
- sm1.deRham は計算機の資源を大量に使用する. たとえば sm1.deRham(0,[x*y*z*(x+y+z-1)*(x-y),[x,y,z]]) の計算すらすでに非常に大変である.
- b-関数の根を効率よく解析するには, ox_asir が ox_sm1_forAsir より使用されるべきである. コマンド
  sm1(0,"[(parse) (oxasir.sm1) pushfile] extension"); を用いて, ox_asir との通信モジュールをあらかじめロードしておくとよい. このコマンドは ox_asir_forAsir のスタート時に自動的に実行されている.
- sm1.deRham を ox_reset(sm1.get_Sm1_proc()); で中断すると, 以後 sm1 サーバが非標準モードに入り予期しない動作をする場合があるので, コマンド

ox_shutdown(sm1.get_Sm1_proc()); で, ox_sm1_forAsir を一時 shutdown してリスタートした方が安全である.

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

参照 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*)
:: 多項式の集合 f のヒルベルト多項式を計算する.

hilbert_polynomial(*f*,*v*)
:: 多項式の集合 f のヒルベルト多項式を計算する.

return 多項式

p 数

f, v リスト

- 多項式の集合 f の変数 v にかんするヒルベルト多項式 h(k) を計算する.
- h(k) = dim_Q F_k/I \cap F_k ここで F_k は次数が k 以下であるような多項式の集合である. I は多項式の集合 f で生成されるイデアルである.
- sm1.hilbert にかんするノート: 効率よく計算するには f はモノミアルの集合にした方がいい. 実際, この函数はまず f のグレブナ基底を計算し, それからその initial monomial 達のヒルベルト多項式を計算する. したがって, 入力 f がすでにグレブナ基底だとこの函数のなかでもう一度グレブナ基底の計算がおこなわれる. これは時間の無駄であるし, sm1 の多項式グレブナ基底計算は asir より遅い.

```
[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
```

参照 sm1.start, sm1.gb, longname

1.2.7 sm1.genericAnn

sm1.genericAnn([*f*,*v*]|proc=*p*)
:: *f*^s のみたす微分方程式全体をもとめる. *v* は変数のリストである. ここで, s は *v*[0] であり, *f* は変数 rest(*v*) 上の多項式である.

return リスト

p 数

f 多項式

v リスト

- この函数は, *f*^s のみたす微分方程式全体をもとめる. *v* は変数のリストである. ここで, s は *v*[0] であり, *f* は変数 rest(*v*) 上の多項式である.

```
[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]
```

参照 sm1.start

1.2.8 sm1.wTensor0

sm1.wTensor0([*f*,*g*,*v*,*w*]|proc=*p*)
:: *f* と *g* の D-module としての 0 次テンソル積を計算する.

return リスト

p 数

f, *g*, *v*, *w* リスト

- *f* と *g* の D-加群としての 0 次テンソル積を計算する.
- *v* は変数のリストである. *w* は weight のリストである. 整数 *w*[i] は変数 *v*[i] の weight である.
- sm1.wTensor0 は ox_sm1 の wRestriction0 をよんでいる. wRestriction0 は, generic な weight ベクトル *w* をもとにして制限を計算している. Weight ベクトル *w* が generic でないと計算がエラーで停止する.
- F および G を *f* と *g* それぞれの解とする. 直観的にいえば, 0 次のテンソル積は 関数 FG のみたす微分方程式系である.
- 入力 *f*, *g* が D の左イデアルであっても, 一般に, 出力は自由加群 D^r の部分加群である.

```
[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 リスト

f 多項式

g, *v*, *w* リスト

p 数 (ox_sm1 のプロセス番号)

- この函数は f を homogenized ワイル代数において, 多項式集合 g で簡単化 (reduce) する; つまり, この函数は, f に割算アルゴリズムを適用する. 変数集合は v で指定する. w は順序を指定するための ウエイトベクトルであり, 省略してもよい. `sm1.reduction_noH` は, Weyl algebra 用.
- 戻り値は次の形をしている: [r,c0,[c1,...,cm],g] ここで g=[g1, ..., gm] であり, c0 f + c1 g1 + ... + cm gm = r がなりたつ. r/c0 が normal form である.
- この函数は, 低次項にあらわれる reducible な項も簡単化する.
- 函数 sm1.reduction_d(P,F,G) および sm1.reduction_noH_d(P,F,G) は, 分散多項式用である.

```
[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]]
```

参照 `sm1.start`, `d_true_nf`

1.2.10 sm1.xml_tree_to_prefix_string

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

:: XML で書かれた OpenMath の木表現 s を前置記法になおす.

return String

p Number

s String

- XML で書かれた OpenMath の木表現 s を前置記法になおす.
- この函数は om_* に将来移すべきである.
- `om_xml_to_cmo(OpenMath Tree Expression)` は CMO_TREE を戻す. `asir` はこの CMO をまだサポートしていない.
- `java` の実行環境が必要. (たとえば, /usr/local/jdk1.1.8/bin をコマンドサーチパスに入れるなど.)

```
[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))
```

参照　　om_*, OpenXM/src/OpenMath, eval_str

1.2.11 sm1.syz

sm1.syz([*f*,*v*,*w*]|proc=*p*)
　　:: v 上の微分作用素環において f の syzygy を計算する.

return　　リスト

p　　数

f, v, w　　リスト

- 戻り値は次の形をしている: $[s,[g, m, t]]$. ここで s は f の v を変数とする微分作用素環における syzygy である. g は f の weight vector w に関するグレブナ基底である. m は入力行列 f をグレブナ基底 g へ変換する行列である. t はグレブナ基底 g の syzygy である. まとめると, 次の等式がなりたつ: $g = m\ f$, $s\ f = 0$.
- Weight ベクトル w は省略してよい. 省略した場合, graded reverse lexicographic order をつかってブレブナ基底を計算する.
- Term order でない順序が与えられた場合は, 同次化ワイル代数でグレブナ基底が計算される (SST の本の Section 1.2 を見よ). 同次化変数 h が結果に加わる.

```
[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 サーバ に f かける g を v 上の微分作用素環でやってくれるように頼む.

return 多項式またはリスト

p 数

f, g 多項式またはリスト

v リスト

- sm1 サーバ に f かける g を v 上の微分作用素環でやってくれるように頼む.
- `sm1.mul_h` は homogenized Weyl 代数用.
- BUG: `sm1.mul(p0*dp0,1,[p0])` は `dp0*p0+1` を戻す. d 変数が後ろにくるような変数順序がはいっていないと, この関数は正しい答えを戻さない.

```
[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` に f の distraction を計算してもらう.

return リスト

p 数

f 多項式

v,x,d,s リスト

- 識別子 p の `sm1` サーバに, f の distraction を v 上の微分作用素環で計算してもらう.
- x , d は, それぞれ, distract すべき x 変数, d 変数のリスト. Distraction したら, s を変数として結果を表す.
- Distraction というのは x*dx を x で置き換えることである. 詳しくは Saito, Sturmfels, Takayama : Grobner Deformations of Hypergeometric Differential Equations の page 68 を見よ.

```
[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
```

参照 distraction2(sm1),

1.2.14 sm1.gkz

sm1.gkz([*A*,*B*]|proc=*p*)
:: 行列 A とパラメータ B に付随した GKZ 系 (A-hypergeometric system) をもどす.

return リスト

p 数

A, B リスト

- 行列 A とパラメータ B に付随した GKZ 系 (A-hypergeometric system) をもどす.

```
[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*)
:: F_1 または F_D に対応する方程式系を戻す.

return リスト

p 数

a リスト

- Appell の関数 F_1 および その n 変数化である Lauricella の関数 F_D(a,b1,b2,...,bn,c;x1,...,xn) のみたす微分方程式系を戻す. ここで, a =(a,c,b1,...,bn). パラメータは有理数でもよい.
- sm1 の関数 appell1 をよぶわけでないので, パラメータが有理数や文字式の場合も正しく動く.

```
[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*`)`
:: F_4 または F_C に対応する方程式系を戻す.

return リスト

p 数

a リスト

- Appell の関数 F_4 および その n 変数化である Lauricella の関数 F_C(a,b,c1,c2,...,cn;x1,...,xn) のみたす微分方程式系を戻す. ここで, a =(a,b,c1,...,cn). パラメータは有理数でもよい.
- sm1 の関数 appell1 をよぶわけでないので, パラメータが有理数や文字式の場合も正しく動く.

```
[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*`)`
:: 微分方程式系 a の holonomic rank を戻す.

return 数

p 数

a リスト

- 微分方程式系 *a* の, generic point での正則解の次元を戻す. この次元を, holonomic rank と呼ぶ.
- *a* は微分作用素のリストと変数のリストよりなる.
- *a* が regular holonomic のときは `sm1.rrank` も holonomic rank を戻す. いっぱんにこの関数の方が `sm1.rank` より早い.

```
[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*`)`
:: フラグ "AutoReduce" を *s* に設定.

戻り値 数

p 数

s 数

- *s* が 1 のとき, 以後計算されるグレブナ基底はすべて, reduced グレブナ基底となる.
- *s* が 0 のとき, 計算されるグレブナ基底は reduced グレブナ基底とはかぎらない. こちらがデフォールト.

1.2.19 sm1.slope

`sm1.slope(`*ii*`,`*v*`,`*f_filtration*`,`*v_filtration*`|proc=`*p*`)`
:: 微分方程式系 *ii* の slope を戻す.

return 数

p 数

ii リスト (方程式)

v リスト (変数)

f_filtration リスト (weight vector)

v_filtration
: リスト (weight vector)

- `sm1.slope` は微分方程式系 *ii* の V filtration *v_filtration* で指定する超平面に沿っての (geomeric) slope を計算する.
- *v* は変数のリスト.
- 戻り値は, リストを成分とするリストである. 成分リストの第 1 要素が slope, 第 2 要素は, その weight vector に対応する microcharacteristic variety が bihomogeneous でない.

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" をみよ. Slope の本来の定義では, 符号が負となるが, このプログラムは, Slope の絶対値を戻す.

```
[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]);
```

参照 `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 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);
```

1.2.24 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

1 SM1 函数 1
Index 24

Table of Contents

1 SM1 函数 1
1.1 ox_sm1_forAsir サーバ 1
1.1.1 ox_sm1_forAsir 1
1.2 函数一覧 1
1.2.1 sm1.start 1
1.2.2 sm1.sm1 2
1.2.3 sm1.push_int0 3
1.2.4 sm1.gb 3
1.2.5 sm1.deRham 5
1.2.6 sm1.hilbert 6
1.2.7 sm1.genericAnn 7
1.2.8 sm1.wTensor0 7
1.2.9 sm1.reduction 8
1.2.10 sm1.xml_tree_to_prefix_string 8
1.2.11 sm1.syz 9
1.2.12 sm1.mul 9
1.2.13 sm1.distraction 10
1.2.14 sm1.gkz 11
1.2.15 sm1.appell1 11
1.2.16 sm1.appell4 12
1.2.17 sm1.rank 12
1.2.18 sm1.auto_reduce 13
1.2.19 sm1.slope 13
1.2.20 sm1.ahg 14
1.2.21 sm1.bfunction 14
1.2.22 sm1.call_sm1 14
1.2.23 sm1.ecart_homogenize01Ideal 15
1.2.24 sm1.ecartd_gb 15
1.2.25 sm1.ecartd_gb_oxRingStructure 15
1.2.26 sm1.ecartd_isSameIdeal_h 15
1.2.27 sm1.ecartd_reduction 16
1.2.28 sm1.ecartd_reduction_noh 16
1.2.29 sm1.ecartd_syz 16
1.2.30 sm1.gb_oxRingStructure 17
1.2.31 sm1.gb_reduction 17
1.2.32 sm1.gb_reduction_noh 17
1.2.33 sm1.generalized_bfunction 17
1.2.34 sm1.isSameIdeal_in_Dalg 18
1.2.35 sm1.restriction 18
1.2.36 sm1.saturation 18
1.2.37 sm1.ahg 18
1.2.38 sm1.bfunction 18
1.2.39 sm1.call_sm1 19

1.2.40 sm1.ecart_homogenize01Ideal 19
1.2.41 sm1.ecartd_gb 19
1.2.42 sm1.ecartd_gb_oxRingStructure 20
1.2.43 sm1.ecartd_isSameIdeal_h 20
1.2.44 sm1.ecartd_reduction 20
1.2.45 sm1.ecartd_reduction_noh 20
1.2.46 sm1.ecartd_syz 20
1.2.47 sm1.gb_oxRingStructure 21
1.2.48 sm1.gb_reduction 21
1.2.49 sm1.gb_reduction_noh 21
1.2.50 sm1.generalized_bfunction 22
1.2.51 sm1.isSameIdeal_in_Dalg 22
1.2.52 sm1.restriction 22
1.2.53 sm1.saturation 22

Index 24